TM

Язык программирования LogicProgram

Прежде чем начать работу, прочтите это

Товарные знаки и авторские права

Торговый знак Miracle – принадлежал ЗАО НПФ «И.В.А.» и используется в качестве обозначения системы быстрой разработки – Miracle™ (имеющего также и графическое представление).

Торговый знак LogicProgram – принадлежит ЗАО НПФ «И.В.А.» и используется в качестве обозначения системы быстрой разработки – LogicProgram (LP имеет графическое представление).

Использование в руководстве название фирм и торговых марок предназначено только в качестве пояснения и ссылок на соответствующих правообладателей.

Лицензия

Представленный документ «Язык программирования LogicProgram» (закрытого акционерного общества Научно-производственной фирмы “И.В.А.”), поставляемый на электронных носителях или иным способом, лицензируются Вам в соответствии с настоящим Лицензионным соглашением ЗАО НПФ “И.В.А” (“Соглашение”).

Соглашение

Вы можете использовать представленный документ на любом компьютере или в локальной сети. В настоящем Соглашении понятие “*использовать*” означает загрузку документа на жесткий диск, сетевой сервер или другое запоминающее устройство.

Вам разрешается:

1. Изготовить необходимое количество копий документа «Язык программирования LogicProgram»;
2. Вы не имеете право вносить изменения в документ «Язык программирования LogicProgram».

Ограниченная гарантия

НПФ “И.В.А.” гарантирует:

1. документа «Язык программирования LogicProgram» содержит необходимую информацию для использования программного обеспечения – платформа «LogicProgram».

ЗА ИСКЛЮЧЕНИЕМ ОГРАНИЧЕННОЙ ГАРАНТИИ, УСТАНОВЛЕННОЙ ВЫШЕ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ДОКУМЕНТ «ЯЗЫК ПРОГРАММИРОВАНИЯ LOGICPROGRAM» ПОСТАВЛЯЮТСЯ “ТАКОЙ, КАКОЙ ОН ЕСТЬ”. ЗАО НПФ “И.В.А.” НЕ ПРЕДОСТАВЛЯЕТ НИКАКИХ ДРУГИХ ГАРАНТИЙ, ЧЕТКО ВЫРАЖЕННЫХ ИЛИ ПОДРАЗУМЕВАЮЩИХСЯ, ОТНОСИТЕЛЬНО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И (ИЛИ) ДОКУМЕНТАЦИИ И ОСОБО ОТВЕРГАЕТ ПОДРАЗУМЕВАЮЩИЕСЯ КОММЕРЧЕСКИЕ ГАРАНТИИ И ГАРАНТИИ ГОДНОСТИ ДЛЯ КОНКРЕТНОЙ ЦЕЛИ. ЗАО НПФ “И.В.А.” НЕ ГАРАНТИРУЕТ, ЧТО ДОКУМЕНТ «ЯЗЫК ПРОГРАММИРОВАНИЯ LOGICPROGRAM» БУДУТ ОТВЕЧАТЬ ВАШИМ ТРЕБОВАНИЯМ ИЛИ ПОЖЕЛАНИЯМ ИЛИ, ЧТО ДОКМЕНТАЦИЯ БУДЕТ СВОБОДНОЙ ОТ ОШИБОК. ТОЛЬКО ВЫ САМИ ОТВЕЧАЕТЕ ЗА ВЫБОР ДОКУМЕНТАЦИИ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ ДЛЯ ДОСТИЖЕНИЯ НАМЕЧЕННЫХ ВАМИ РЕЗУЛЬТАТОВ И ЗА ФАКТИЧЕСКИ ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ.

Данная гарантия дает вам особые права, но вы можете иметь и другие права, которые оговорены действующим законодательством.

Ограничение возмещения и ответственности

В случае если документ «Язык программирования LogicProgram» не соответствует ”ограниченной гарантии” ЗАО НПФ “И.В.А.”, фирма ЗАО НПФ “И.В.А.” несет полную ответственность, и Вы имеете право на получение от ЗАО НПФ «И.В.А.» исправленного документа «Язык программирования LogicProgram», в сроки и способом установленного ЗАО НПФ «И.В.А.»

Ни при каких условиях и, несмотря на недостаточность достижения главной цели при любом ограниченном средстве возмещения, предусмотренном в настоящем Соглашении, ЗАО НПФ “И.В.А.” не несет перед Вами, любым пользователем или какой-либо третьей стороной ответственности за любые косвенные, фактические, вытекающие, случайные или штрафные убытки любого рода на основании контракта, гражданского правонарушения или на иных основаниях, в том числе, без ограничений, за повреждения, потерю дохода, потерю репутации, упущенные коммерческие возможности, утрату данных и (или) потерю прибыли, потерянных накоплениях или другом специфическом, случайном или косвенном ущербе, возникающем в результате использования или невозможности использования документации, независимо от того, можно ли было предвидеть возможность таких убытков, и была ли фирма ЗАО НПФ «И.В.А.» уведомлена о возможности их возникновения.

Оглавление

Обозначения и сокращения

- Система быстрой разработки (RAD) – rapid application development (быстрая разработка приложений) — концепция создания средств разработки программных продуктов;
- Miracle (Miracle plus) – RAD система;
- LogicProgram (LP) – RAD система;
- LP-программа – конечное, пользовательское приложение (программа) созданное с использованием языка программирования LogicProgram;
- Компонент – элемент построения LP-программы;
- Объект – в рамках платформы LogicProgram понятие объекта тождественно понятию компонент;
- Кросс-платформенность – свойство платформы LogicProgram поддерживать исполнение LP-программ в разных операционных средах;
- LP-машина – механизм обеспечивающий кросс-платфоменность LP-программ;
- Скрипт – тоже самое, что и LP-программа.

Введение

Язык программирования LogicProgram

Название языка	LogicProgram
Класс языка	Компонентный
Метод разработки	Цикличный
Первая публикация	1993
Разработчик	Коллектив ЗАО НПФ «И.В.А.»
Диалекты	Miracle; Miracle Plus
Релиз	Ноябрь 2010
Типизация данных	Автоматическая; явная
Основная реализация	Платформа LogicProgram
Поддерживается на операционных средах	Windows; Linux; QNX; Java; Android
Сайт	http://www.logicprogram.ru

Основные особенности языка

Язык программирования LogicProgram предназначен для написания компьютерных программ, обеспечивающих передачу компьютеру инструкций по выполнению того или иного вычислительного процесса и организацию управления отдельными устройствами.

Язык программирования LogicProgram позволяет осуществлять разработку программ формированием структуры связей между компонентами, т.е. формировать *сеть процессов*. Формирование программного кода осуществляется не текстовым методом. Реализация поддержки кода программы осуществляется целевыми машинами платформы LogicProgram.

Язык программирования LP определяет способ передачи команд, организацию обмена информацией через использования специальных конструкции позволяющих определять и манипулировать структурами данных и процессами вычислений.

Типизация данных – автоматическая (динамическая), также может быть применена явная типизация через декларирование типа данных при описании структуры переменных, массивом и т.п.

Семантика языка ориентирована на исполнение кода приложения целевыми машинами платформы, при этом язык LP не удовлетворяет интуитивному требованию ясности, и не ориентирован на удобочитаемость человеком и представляет матрицу отношений компонентов программы в виде формализованной записи кода программы (семантическая сеть). Среда проектирования выполняет задачи по обеспечению простоты и ясности понимания структуры сети

человеком. При этом визуальное программирование в среде разработки предоставляет одну из возможных методик построения и управления сетью процессов. Вычисления в языке LogicProgram строятся на событийной модели формирующей очередь исполнения.

Программа на языке LP является исполняемой, т.е. не требует процесса компиляции или интерпретации. Данное свойство позволяет вносить изменения в исходный код и исполнять его сразу, без каких либо дополнительных действий. Наличие средств псевдокомпиляции в языке LogicProgram носит оптимизационный характер, что позволяет уменьшить размеры исполняемого кода программы.

История развития

Язык программирования LogicProgram ведет свое начало от платформы Miracle, и ориентирован на поддержку концепции быстрой разработки приложений – RAD (*rapid application development*). Основа концепции направлена на предоставление программисту средств создания программы максимально быстро, поэтапным (цикличным) методом проектирования. Минимизация времени разработки версий проектируемого решения основана на оценке результата предыдущего варианта, уточнений требований, и многократном использовании выверенных блоков программного кода. Визуальное программирование, повышает скорость разработки, снижает трудоемкость восприятия «кода» программы.

Первая реализация среды быстрой разработки – Miracle, появилась в 1993г. (http://www.miracle.ru). Отличительной особенностью платформы явилось «не текстовое» формирования кода программы. При этом формировался непосредственно код исполнения (машинный код). Визуальная среда проектирования позволяла создавать код программы без написания текстовых конструкций, непосредственно в машинным коде, выполнение которого обеспечивалось специализированной средой исполнения. Платформа была ориентирована для работы под управлением операционной системы Windows95. Поддержка среды Miracle была остановлена в 2001г.

Вторая реализация среды быстрой разработки – Miracle plus, появилась в 2007г. Отличительной особенность платформы явились «машины исполнения» для нескольких операционных сред, переводящие код программы в машинный код исполнения. Машины платформы M+ были реализованы для операционных систем: Windows, Linux. Поддержка платформы была остановлена в 2008г.

Третья реализация среды быстрой разработки – LogicProgram (http://www.logicprogram.ru), появилась в 2009г. Отличительной особенностью платформы явилось появление формальной записи проекта в текстовом виде. Формат записи XML (*eXtensible Markup Language*). Язык программирования LogicProgram ориентирован на запись отношений компонентов в виде сети связей, обеспечивающих выполнение требуемых процессов. При этом поток данных становиться полноценным механизм поддержки базовой концепции языка: «Событие→Команда».

Направленность языка

Язык программирования LogicProgram задумывался как средство, которое должно обеспечить высокий уровень разработки программы, при увеличении ее сложности. Метод разработки – цикличный, позволяющий формировать прототип программы небольшими законченными циклами, производить уточнение требований и производить следующий цикл проектирования. Данная методика не требует полной формализации проекта, и позволяет начинать программирования с набором «не четких» требований.

Обеспечение качества программы, при увеличении ее сложности обеспечивается методикой программирования на базе готовых, не изменяемых компонентов. Основной задачей программиста является формирование поведения программы путем перевода используемых компонентов в необходимое состояние. При этом управление инфраструктурными требованиями, относительно корректности работы программы в операционной среде, правил работы с внешними источниками, и т.п. решается непосредственно платформой LP и часто скрыто от программиста. Среда проектирования LP-программ реализует концепцию компонентного, сборочного программирования. В рамках данной концепции программист оперирует не алгоритмом исполнения функций (процедурный подход) и не устанавливает иерархию взаимодействия объектов (объектно-ориентированный подход), а управляет состоянием компонентов через сеть возможных отношений между ними. Данный подход позволяет манипулировать небольшим набором отношений и экземпляров компонентов, даже при общем росте сложности алгоритма работы программы.

Формат записи кода программы оформлен в виде XML файла. Формат исполняемого кода оформлен в виде XML файла, но в отличие от формата проекта не имеет различных значений ориентированных на программиста.

Определение компонента

Компонентное программирование является подходом, где понятие «компонента» носит инженерное (конструкторское) значение, при котором парадигма разработки программного обеспечения позволяет реализовать «*сборочную модель*» проектирования.

Компонент это специализированное программное решение, имеющее однозначный, неизменяемый интерфейс взаимодействия, позволяющего организовывать необходимую конструкцию программного кода. Все компоненты в платформе LogicProgram разделены на два класса:

- *Интерфейсные (визуальные)* – предназначенные для организации взаимодействия с пользователем;
- *специализированные* – предназначенные для решения различных задач обработки данных и управления программой.

Библиотека компонентов может быть дополнена через API платформы LP. Разработка новых компонент может быть произведена различными языками программирования.

В зависимости от типа, каждый экземпляр компонента в проекте может иметь собственные настройки (конфигурацию) предоставляющие требуемый уровень многообразия использования компонента платформы.

Основное свойства компонента – многократное решение поставленных целей без доступа во внутреннее устройство объекта.

Спецификация компонента

Каждая компонента платформы LogicProgram имеет формализованный интерфейс, состоящего из следующих частей:

Тип интерфейса	Описание
События	интерфейс, обеспечивающий предоставление в систему информации об изменении состояния экземпляра компонента
Команды	интерфейс, обеспечивающий управление экземпляром компонента
Входные данные	интерфейс, обеспечивающий передачу данных в структуру экземпляра компонента
Выходные данные	интерфейс, обеспечивающий получение данных из структуры экземпляра компонента

Следует помнить, что компоненты могут предоставлять не все типы интерфейсов, это зависит от назначения конкретного компонента системы.

Программирование на языке LP требует иного взгляда на ход разработки программы, так как сборка готового решения основано на конструировании из набора готовых (*неизменяемых*) компонент. Данный подход отличается от подходов принятых в других парадигмах разработок «ПО». Основа «сборочной» парадигмы базируется на формировании алгоритма работы программы, небольшими шагами, при этом развитие программы не требует рассмотрения всего комплекса связей компонентов. Поэтапное развитие «поведения» программы основано на дополнении процессов в уже имеющуюся структуру сети решения. Основа парадигмы – *связь* между компонентами.

Таблица 1 Сравнительная таблица парадигм разработки

Подход к разработке «ПО»	Основа парадигмы
Процедурный	действие
Объектно-ориентированный	взаимодействие
Компонентный, сборочный	связь

Преимущества компонентного, сборочного программирования:

- повторное использование компонент;
- повышение надежности системы за счет верификации компонент;
- снижение стоимости разработки за счет повторного использования сети компонентов (процесса);
- при необходимости простая замена компонент в работающих системах.

Компонентный, сборочный подход отличается инженерной философией проектирования структуры программ. Данный подход Основанной на том, что компонент это относительно самостоятельная часть проектируемой системы, из которых состоит вся конструкция программы. При этом отдельные части одного и того же назначения могут быть заменены между собой без применения каких либо дополнительной реорганизации.

Особенности среды программирования

Отличительная особенность платформы LogicProgram – предоставление нескольких методов формирования структуры кода проекта, где визуальный подход является основным механизмом проектирования. Наличие формальной записи проекта на языке LogicProgram предоставляет возможность использования нескольких методик программирования.

Все методики построены на едином принципе - выборе свойств интерфейсов компонентов, между которыми необходимо установить связь. Полный список интерфейсов взаимодействия с компонентом:

Интерфейсы управления состоянием компонента	События Команды
Связи по данным между компонентами	Приемник Источник

В зависимости от типа компонента, некоторые интерфейсы могут быть не доступны.

Платформа LP предоставляет следующие методики программирования:

- *базовая* - проектирование сети процессов программы, через формирование связи отношений от выбранного объекта. Т.е. когда программист осуществляет выбор в дереве проекта необходимого компонента, требуемый интерфейс: (событие/команда/источники или приемника данных), конкретную характеристику и далее для установки требуемой связи выбирает из дополнительного окна нужную ответную часть;

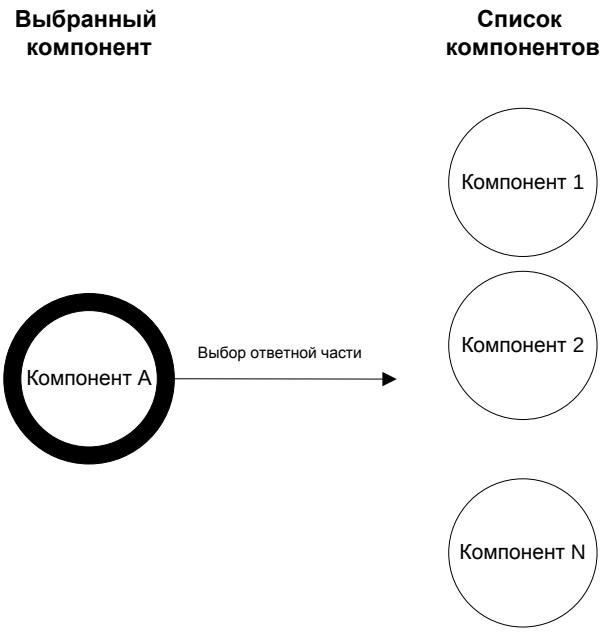


Рисунок 1 Схема выбора ответной части для установки связи

Подход программирования в *базовой* методике, ориентирован на работу с мышью, и предоставляет список интерфейсов компонента их характеристики и установленные связи в рамках одного окна. Т.е. программисту предоставляется «обзор» всех отношений компонента одновременно. Данный способ формирования связей между компонентами предоставляет обзорный подход по выбору свойств, команд, потоков данных в ответном компоненте отношений.

- *вспомогательная* - проектирование сети процессов программы, через установку связи между двумя выбранными компонентами Т.е. когда из списка проекта выбирается необходимый компонент, его интерфейс (события/команды/источника или приемника данных) конкретная характеристика, тоже для второго компонента, после чего устанавливается связь между ними.

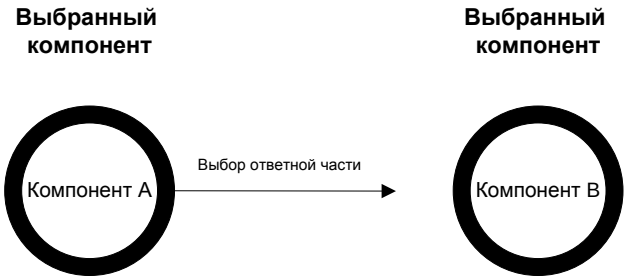


Рисунок 2 Схема установки связи между двумя выбранными компонентами

Подход в *вспомогательной* методике, ориентирован на работу с клавиатурой, через предоставление двух списочков характеристик компонентов проекта одновременно. Способ организации работы во «вспомогательном режиме» подобен работе с консольным файловым менеджером, и ориентирован на управление связями через функциональные клавиши. Дерево каталогов проекта программы представлено в виде списка соответствующего уровня, вплоть до характеристик интерфейсов компонента. Данный метод предоставляет механизм для скоростного программирования в LogicProgram.

Несмотря на разность методик программирования, все они взаимозаменяемые и ориентированы на формирование и управление необходимой структуры сети отношений между компонентами.

Наличие средств отображения выбранного участка сети в виде графической схемы, обеспечивает программиста дополнительным механизмом контроля структуры программы. В качестве средства отображения используется среда Microsoft Visio.

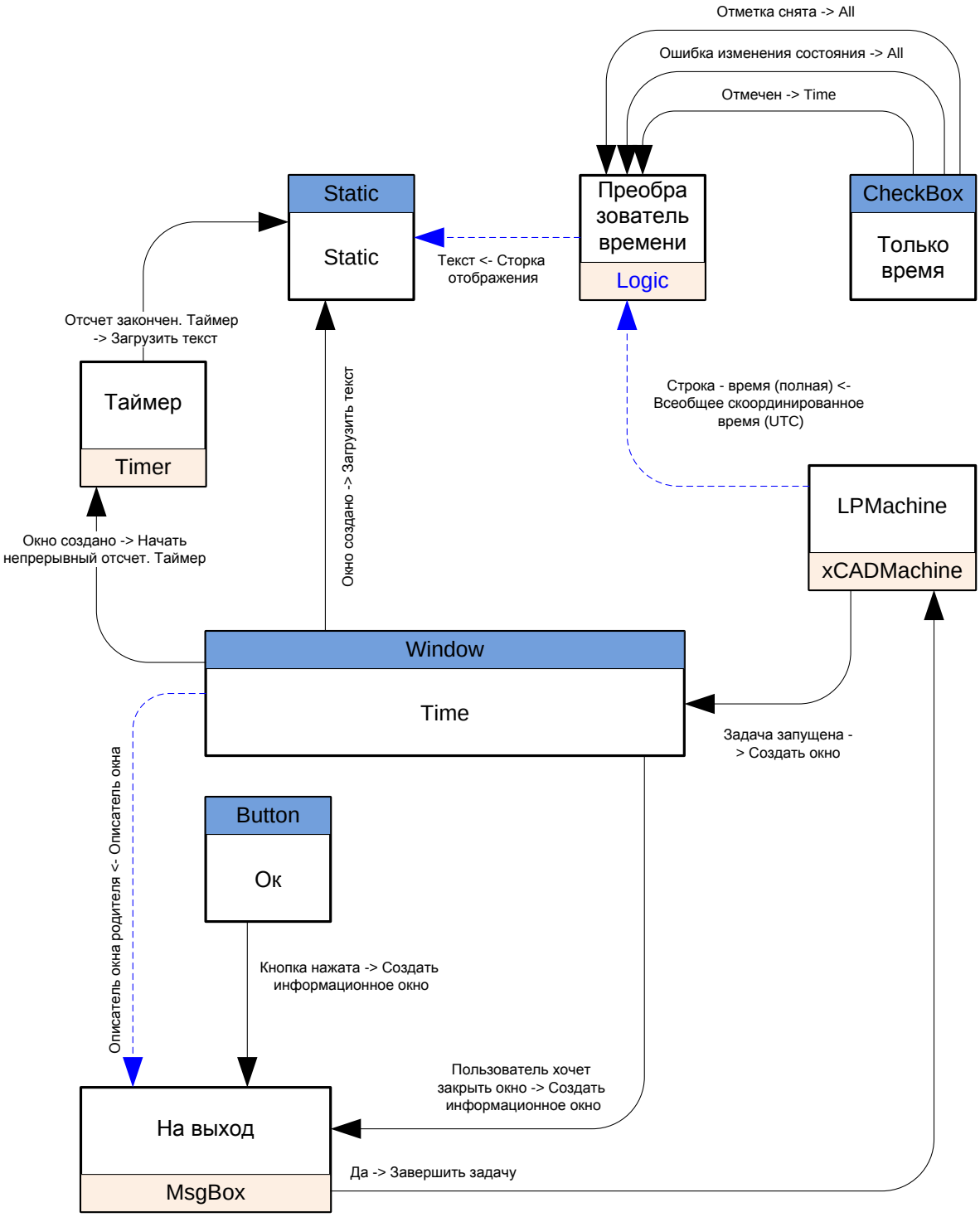


Рисунок 3 Пример графического отображения сети связи компонентов программы

Легенда: прямая линия – связь управления, пунктирная линия – связь по данным.

Структура дерева проекта

Уровень в дереве проекта	Назначение
Корень	Характеристики проекта, его свойства
Интерфейсные объекты	Сборка всех визуальных объектов
Компонент «окно»	Компонент окно, в рамках которого размещаются все остальные интерфейсные компоненты. Количество окон не регламентируется
1-n Интерфейсный компонент	Какой либо интерфейсный компонент
Специальные объекты	Сборка всех специальных объектов
Подкаталог	Возможная сборка компонентов в каталоге (оформляется программистом)
1-n Специальный компонент	Какой либо специальный компонент

Структура компонентов

Уровень в структуре компонента	Назначение
Компонент	Компонент любого типа
Интерфейс	Интерфейс взаимодействия. Один из 4-х типов интерфейсов. В зависимости от назначения компонента не все интерфейсы могут быть представлены
Характеристика	Характеристика интерфейса – механизм взаимодействия с компонентом. Назначение характеристики меняется в зависимости от типа интерфейса. Характеристика может быть как командой, событием так и способом взаимодействием с компонентом по данным.
Связь	Установленная связь с характеристикой интерфейса. Связь обозначается как название характеристики, или как название объекта ответа. В списке характеристик может быть более одной связи. В этом случае их последовательность исполнения определяет очередь в списке.

Свойства языка LogicProgram

- Минимизация времени разработки;
- Методика программирования позволяет включать в процесс разработки программ, специалистов разного уровня подготовки в области программирования, но в первую очередь методика направлена на прикладных специалистов, не имеющих специальной подготовки в области программирования;
- Проектирование циклами, где полученный прототип используется для уточнения требований и является базой для формирования следующей итерации программы;
- Структура связей между экземплярами компонентов формирует сеть процессов;
- Сеть процессов формируется из связей управления: «Команда→Событие», так и связей данных «Источник→Приемник»;
- Последовательность выполнения связей от первой команды до последнего события называется процессом;
- Изменение состояния компонентов основывается на факте активизации цепочек связей, последовательность выполнения которых обусловлена очередью их формирования (процессом);
- Параллельное выполнение процессов в рамках единой очереди исполнения;
- Поддерживается методика программирования, основанная на управлении потоков данных;
- Визуальная среда проектирования обеспечивает управление структурой программы;

- Портируемость кода исполнения, т.е. LP-программа (без модификации) может исполняться в различных операционных средах.

Развитие языка LogicProgram

Дальнейшее развитие языка программирования LP направлено на предоставление средств компиляции проекта программы в двоичный исполняемый код, что обеспечит повышение производительности программы на этапе ее запуска, и оптимизации исполнения кода. Компиляция программ будет выполнена в едином формате, который будет поддерживаться всеми целевыми машинами платформы. Скомпилированная программа будет представлять структуру, для исполнения которой целевые машины не будет нужно проводить процессы организационного характера. Единый компилятор LP-программы будет оптимизирован для каждой машины исполнения, с тем, чтобы обеспечить использование специфичных требований операционных сред эксплуатации.

Среда разработки

Платформа LogicProgram предлагает инструментальную среду визуального проектирования программ на базе методики сборочного, компонентного программирования. Основная цель системы предоставить разработчикам современную среду компонентного программирования для нескольких операционных сред: Windows, Linux, QNX и для исполнения под JAVA. Основа программирования платформы LP — не текстовые, а визуальные средства проектирования конечных пользовательских решений на базе готовых компонентов.

В общем представлении проект системы, а также компоненты разработанные средствами платформы LP, представляют матрицу (сеть) заданных отношений между экземплярами компонентов. Компонентное программирование на языке платформы LP имеется свой жизненный цикл (итерации) с особенностями проектирования отдельных фаз, что обеспечивает реализацию эволюционного подхода к проектированию программы. Методология тестирования компонентных конфигураций (связей) базируется на проведении этапа тестирования, как конкретных взаимосвязей компонентов, так и на выявлении «пресечений» процессов при возможном ходе работы программы.

В состав пакета платформа "LogicProgram", входят среда проектирования приложений: "LP-Studio", и набор LP-машин, обеспечивающих исполнение LP-программ для разных операционных сред. Среда проектирования работает под управлением операционной среды Windows.

Основные идеи

Основная идея платформы LogicProgram, это предоставить программисту механизм программирования, в рамках которого основное внимание уделяется «предметным» свойствам программы, и исключаются требования по инфраструктурному обслуживанию операционной среды, архитектуры исполнения и т.п.

Методика проектирования направлена на создание небольших, законченных шагов программы – итераций. Каждая итерация программы является завершенным решением, каждая следующая итерация – развитием предыдущего варианта. При этом внимание программиста направлено на рассмотрение конкретного участка разработки, как самостоятельной части программы.

Достижение единства цели конечной программы не требует формализации «полного» алгоритма работы, и может строиться на интеграции нескольких наборов алгоритмов, каждый из которых ориентирован на решение требуемой задачи в виде самостоятельных решений (процессов). Структура процесса формируется набором связей в них. Методика проектирования процессами позволяет управлять малым набором связей, даже при функциональном росте сети программы.

Многократное использование не только компонентов платформы, но и процессов повышает качество программы, и снижает время на её разработку.

Парадигма платформы LogicProgram это простота проектирования, простота модификации, при высоком качестве исполнения.

Основы программирования

Язык программирования LogicProgram ориентирован на формирования алгоритма работы программы в виде сети отношений компонентов. Полученная матрица связей содержит возможные связи между компонентами, но не отражает фактическую последовательность исполнения, так как это зависит от точки запуска в процессе, потоков и структуры данных и т.п.

Форма записи кода программы не ориентирована на удобство восприятия человеком, для этой цели код программы предоставляется в среде разработки в виде дерева компонентов проекта и списка связей между ними. Вспомогательные средства предоставляют механизм для управления структурой кода программы, включая графическое представление сети процессов.

Что нужно для изучения языка LP

Для начала работы с языком LogicProgram необязательно иметь огромный опыт программирования, более того применение навыков процедурного, объектно-ориентированного программирования может только усложнить работу и скрыть все преимущества сборочного программирования. Конечно, в ходе работы вам придется оперировать такими понятиями как переменная, массив, типизация данных, цикл и т.п., однако все эти механизмы не являются обязательными для разработки программ.

Основа компонентного, сборочного программирования базируется на достаточно простой модели поведения элементов программы: когда какой либо объект изменяет свое состояние (например: *пользователь нажал на кнопку в окне программы*), то в системе появляется событие о данном изменении. Если вам необходимо, чтобы данное действие имело продолжение (например: *завершилась работа программы*), то необходимо установить связь с командой которую нужно исполнить с этим событием. Как видно из представленного описания, программирование в платформе LP основано на управлении тем, что именно должна сделать программа.

Для работы с компонентами типа Web-сервер, HTML вам также необходимо знать основы данных технологий (например: HTML, CSS).

Тем, кто не имеет опыта программирования, изучение языка LogicProgram не должно вызвать сильных затруднений, так как язык записи кода программы полностью скрыт от человека. Т.е. нет необходимости изучать правила оформления текста программы, все, что вам надо понимать это то, что именно вы хотите получить от программы и определиться, с помощью каких компонентов собрать требуемое решение.

При росте опыта проектирования программ в LogicProgram, вы сможете использовать более тонкие механизмы программирования, что дополнит ваши программы компонентами собственной разработки, сформировать части программных решений для многократного использования в других проектах и что самое главное, при необходимости легко вносить изменения в уже функционирующие программы.

Тот, кто имеет опыт программирования, может достигнуть результатов, в более короткие сроки, при этом дополняя платформу новыми компонентами написанных на языках высокого уровня, а также интегрировать обработки на LP в собственных системах, что позволит более гибко вносить изменения в функционирование разрабатываемых решений, в том числе и привлекая специалистов из предметных областей. В качестве пояснения будут использоваться пример на языке C++, демонстрирующего работу с LogicProgram.

Основное отличие языка LP от других языков заключено в иной концепции проектирования, при этом объектно-ориентированный подход, основанный на событийной модели очень близок к сборочной модели разработки программ, отличающейся в большей степени отсутствием языковых конструкций в записи команд управления.

Соглашения

Для комментирования хода программирования, код программы в данной книге будет представлен графическими примитивами, получаемыми в Microsoft Visio. Отношения, между которыми, могут быть обозначены в виде связей (обозначены линиями). Связь по управлению, когда событие об изменении состояния одного компонента является действием по активизации команды другого компонента, обозначается сплошной линией. Связь по данным, кода один компонент предоставляет данные (хранимые значения) другому компоненту, обозначается прерывистой линией. Острие линии указывает направление действия.

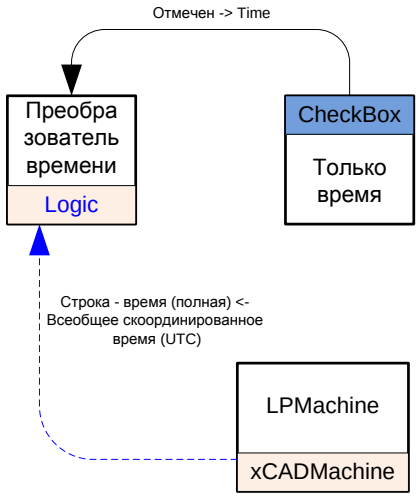


Рисунок 4 Пример графического представления отношений между компонентами

Платформа LogicProgram оперирует двумя типами компонентов: «*интерфейсные*» и «*специальные*». Графические примитивы для каждого типа компонентов разные. Для «визуальных компонентов» системное название в графическом примитиве, отражается в верхней шапке. Для «специальных компонентов» системное название отражается в нижней части. В «теле» примитива отражается название экземпляра компонента заданное программистом. Также связи могут иметь подписи, отражающие название «событий→команды» или «источник→приемник» данных.

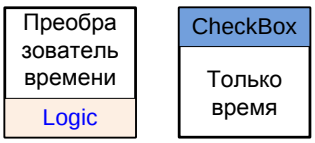


Рисунок 5 Пример граф. примитивов: "Специальный" и "Визуальный"

Следует помнить, что графическое представление сети связей компонентов программы не отражает последовательность их исполнения, а только обозначает заданные связи между экземплярами компонентов.

Структура программы состоит из набора компонентов. Включение компонента в состав решения формирует экземпляр компонента, при необходимости вы можете одновременно использовать более одного экземпляра компонента в решении. В данном документе понятие «компонент в программе» означает экземпляр компонента в дереве проекта.

Современное программирование базируется не только на работе с клавиатурой, но и на работе с манипулятором «мышь». Вся терминология, принятая в этой области достаточна стандартная: «выбор», «перенести», «двойной выбор правой кнопкой» и используется в тексте с учетом того, что она понятна и не требует более точных указаний.

Механизм записи

Среда проектирования LP-Studio обеспечивает запись кода программы на языке LP. Для программиста нет необходимости изучать синтаксис языка, так как для формирования кода программы он оперирует связями, представленными в виде списков характеристик интерфейсов компонента. Среда программирования LP-Studio осуществляет формирование дерева проекта и управление списками связей между компонентами, где сам компонент может иметь следующие типы связей:

- *События* – отражает список событий, которые посылает в систему компонент при изменении своего состояния;
- *Команды* – отражает список команд, посредством которых можно поменять состояние компонента;
- *Источник данных* – отражает список информационных источников компонента;
- *Приемник данных* – отражает список каналов приема информации в компонент.

Запись кода отношений между компонентами производиться через механизм выбора нужных характеристик компонентов. После создания, связь отражается в списке характеристик интерфейса. Для описания таких связей в данном документе, будет использоваться следующий вид:

«*С: (имя компонента) характеристика →(имя компонента) характеристика*»

Или

«*Д: (имя компонента) характеристика →(имя компонента) характеристика*»

Начальная буква в каждой строке указывает тип связи: «**С**» - событие, следовательно, вторая часть в записи – Команда, для записи управления. «**Д**» - связь по данным, где первая часть записи это источник, а вторая часть записи, это приемник данных.

На уровне файла проекта осуществляется запись соответствующих строк кода в формате языка LogicProgram.

Понятие проекта

Проект программы это набор компонентов и установленные отношения между ними. Структура проекта может включать как интерфейсные, так и специальные компоненты. При этом имеется возможность импорта набора компонентов оформленных ранее как часть проекта. Подробности см. «*Дополнительное свойство временной папки*»

Следует помнить, что основное отличие интерфейсных и специальных компонентов заключается в их назначении, для организации взаимодействия программы с человеком используются интерфейсные компоненты, оформленные в рамках базового объекта – *окно*. Количество окон в проекте не ограничено. Специальные компоненты позволяют организовать работу по обработке данных, расчетам и т.п., при этом некоторые специальные компоненты могут предоставлять собственные диалоговые окна взаимодействия с человеком или работать как расширение интерфейсных объектов.

Написание кода

Программирование на языке LogicProgram не требует формирование строковых конструкций описывающих алгоритм работы программы. Проектирование кода оформляется операцией установки взаимоотношений между компонентами. Установленная связь записывается в виде специализированных тегов в файл проекта, однако программисту нет необходимости оперировать структурой программы на уровне текста.

Программирование на LP это проектирование требуемого алгоритма работы программы с использованием механизма управления состоянием требуемых компонентов.

Пример: нужно создать программу (Пример-1), которая отображает текущую дату и время на компьютере пользователя.

Как видно из задачи, программа должна иметь интерфейс (Окно), в рамках которого будет отображаться текущее время. Для этой цели набор компонентов будет следующим:

- *Окно* – компонент для организации взаимодействия с пользователем;
- *Не редактируемый текст* – компонент, принадлежащий компоненту окно, нужен для вывода текущего времени;
- *LPMachine* – специальный компонент, обеспечивает взаимодействие программы с машиной исполнения её кода;
- *Таймер* – специальный компонент, предоставляющий отсчет указанного промежутка времени.

Устанавливаем связь между компонентами:

Первый процесс, обеспечивающий запуск программы:

- *С: (LPMachine) Задача запущена→(Окно) Создать* окно – т.е. первое событие в программе активирует вывод окна интерфейса;
- *С: (Окно) Окно создано→(Не редактируемый текс) Загрузить текст* – по событию о создании окна, производим загрузку текущего времени в компонент отображения;
- *С: (Окно) Окно создано→(Таймер) Начать непрерывный отчет* – по событию о создании окна, активируем работу таймера в режиме постоянного отчета;

  Следует обратить внимание на то, что на одно событие – «*Окно создано*» последовательно активируются две команды. Последовательность команд в списке события влияет на ход работы программы, так как последовательность команд задает очередь их исполнения.
- *Д: (LPMachine) Локальное время→(Не редактируемый текс) Загрузить текст* – указываем компоненту отвечающему за отображения времени источник данных предоставляющего требуемую информацию;

  Также следует организовать смену текущего значения в компоненте - *Не редактируемый текст,* с тем, чтобы время в интерфейсе «шло». Для этой цели требуется компонент, который с частой в 01 секунду, выдавал бы событие о том, что время изменилось. Используем для этой цели компонент *Таймер*.

  Второй процесс, обеспечивающий смену отображения значения текущего времени:
- *С: (Таймер) Отсчет закончен→(Не редактируемый текс) Загрузить текст* – связь, обеспечивающая команду загрузки нового значения времени, с частотой в одну секунду.

  И последней процесс, обеспечивающий завершение программы:
- *С: (Окно) Пользователь хочет закрыть окно→(LPMachine) Завершить задачу* – по событию о том, что пользователь нажал на пиктограмму закрытия окна (в шапке окна), производим завершение работ машины исполнения проекта программы.

Следует помнить, что разбиение данного примера на процессы, необходимо только для пояснения алгоритма работы. Поддержка проектирования в рамках процессов, как механизмов организации событий в программе не поддерживается.

Графическое представление программы со всей структурой связей «Пример-1»:

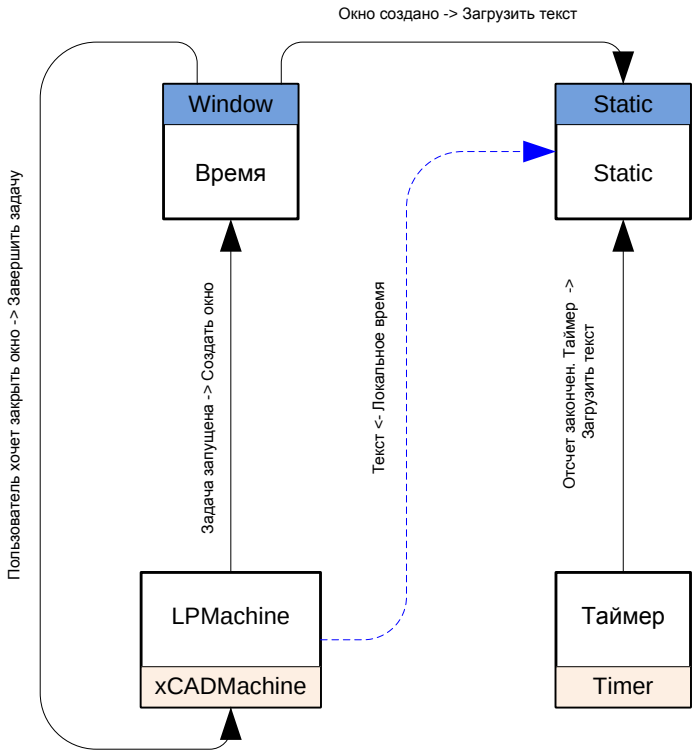


Рисунок 6 Схема программы "Пример-1"

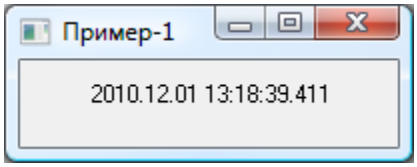


Рисунок 7 Представление программы "Пример-1" пользователю

В итоге размер проекта: 2368 байт, размер скрипта: 2270 байт. Количество компонентов: 4, связей управления:5 связей по данным: 1

Код проекта (*приводиться как пример записи структуры программы на языке LogicProgram*):

```
<?xml version="1.0" encoding="windows-1251"?>
<Project Version="1" CreateDate="1 декабря 2010 г." Type="1" Name="Пример: 1" ModifyDate="1 декабря 2010 г.">
	<DesignTime WASzX="800" WASzY="600"></DesignTime>
	<Objects MaxFldId="5" MaxObjId="5">
		<Interface>
			<Object uuid="{9AC53E5B-3E72-4118-86D1-EF70C74C5639}" Id="1" Sel="0">
				<Prop>
					<X>336</X>
					<Y>230</Y>
					<W>189</W>
					<H>43</H>
					<HAlign>0</HAlign>
					<VAlign>0</VAlign>
					<SysMenu>1</SysMenu>
					<Resize>1</Resize>
					<MinBtn>1</MinBtn>
					<MaxBtn>1</MaxBtn>
					<VType>3</VType>
					<Modal>0</Modal>
```

```
                                        <Type>1</Type>
                                        <Title>Пример-1</Title>
                                        <Visible>1</Visible>
                                        <BGColor>4294967295</BGColor>
                                        <EditBGColor>4294967295</EditBGColor>
                                        <EditTXColor>4294967295</EditTXColor>
                                        <StaticBGColor>4294967295</StaticBGColor>
                                        <StaticTXColor>4294967295</StaticTXColor>
                                        <ButtonBGColor>4294967295</ButtonBGColor>
                                        <ButtonTXColor>4294967295</ButtonTXColor>
                                        <ListBGColor>4294967295</ListBGColor>
                                        <ListTXColor>4294967295</ListTXColor>
                                        <ScrollBGColor>4294967295</ScrollBGColor>
                                </Prop>
                                <Object uuid="{E7D70D40-331A-4B00-AE1E-4A3BAC320D23}" Id="2" Sel="1">
                                        <Prop>
                                                <X>4</X>
                                                <Y>10</Y>
                                                <W>182</W>
                                                <H>20</H>
                                                <HAlign>0</HAlign>
                                                <VAlign>0</VAlign>
                                                <Visible>1</Visible>
                                                <Title>Static</Title>
                                                <Align>2</Align>
                                                <StaticBGColor>4294967295</StaticBGColor>
                                                <StaticTXColor>4294967295</StaticTXColor>
                                        </Prop>
                                        <DataIn PropNum="1">
                                                <Link ObjId="3" PropNum="1"></Link>
                                        </DataIn>
                                </Object>
                                <Event PropNum="1">
                                        <Link ObjId="2" PropNum="4"></Link>
                                </Event>
                                <Event PropNum="4">
                                        <Link ObjId="3" PropNum="1"></Link>
                                </Event>
                        </Object>
                </Interface>
                <Special>
                        <Object uuid="{0C863284-C2FC-4ED5-BDD4-B4666E91B088}" Id="3">
                                <Event PropNum="1">
                                        <Link ObjId="4" PropNum="200" Enable="1"></Link>
                                        <Link ObjId="1" PropNum="1" Enable="1"></Link>
                                </Event>
                        </Object>
                        <Object uuid="{6186094D-50F4-455B-A54A-5B71C65B97C8}" Id="4">
                                <Prop>
                                        <TCount>1</TCount>
                                        <Timer1>1</Timer1>
                                </Prop>
                                <Event PropNum="100">
                                        <Link ObjId="2" PropNum="4"></Link>
                                </Event>
                        </Object>
                </Special>
        </Objects>
</Project>
```

Предположим, что дальнейшее развитие программы требует отображения только текущего времени. Для этой цели дополняем компонент, который будет преобразовывать информацию о дате и времени в требуемый формат отображения. Графическое представление модифицированной структуры программы «Пример-1»:

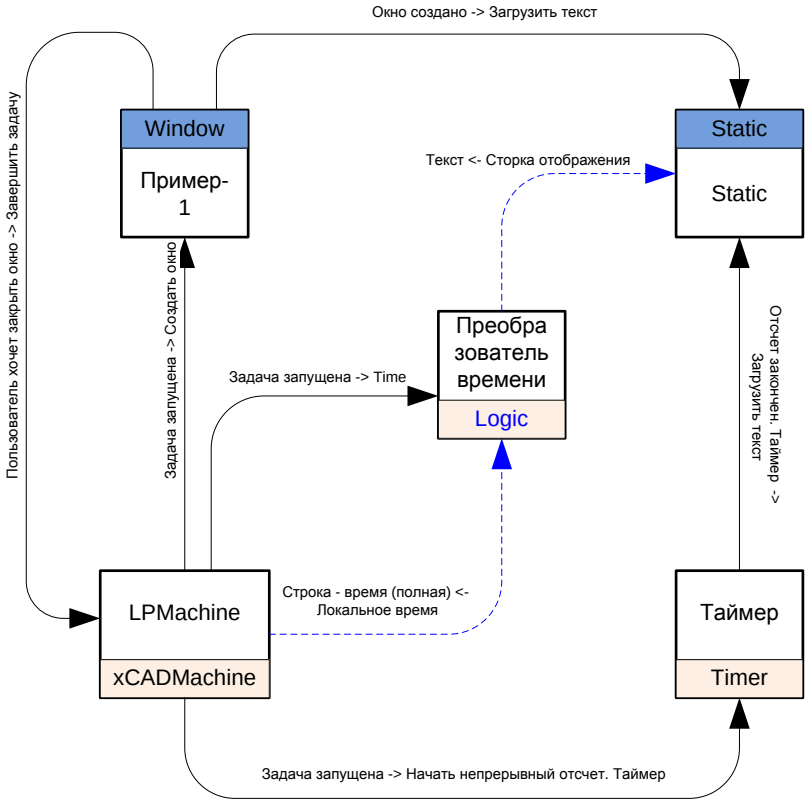


Рисунок 8 Схема программы "Пример-1" (модификация)

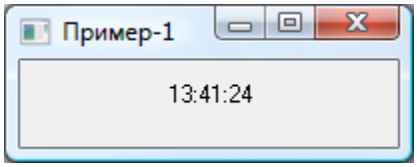


Рисунок 9 Представление программы "Пример-1" пользователю (модификация)

Полученный пример можно разместить на web-странице в виде java-апплета. Подробности о том, как это сделать даны в главе: «*Организация работы LP-машины под Java, через интернет браузер*».

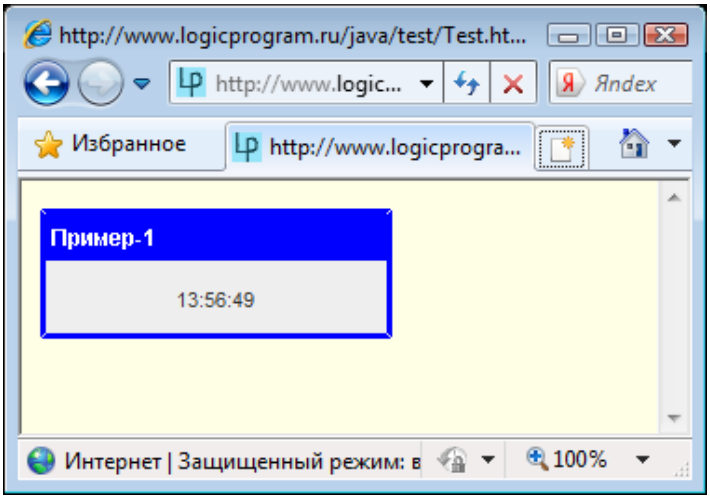


Рисунок 10 Пример работы программы в Internet браузере (под управлением Java машины)

Назначение компонентов, правила их настройки, особенности применения и т.п., описаны в главе: «*Справочник по компонентам платформы LogicProgram*».

Правила проектирования на языке LogicProgram

Основные правила проектирования в LogicProgram

- Добавляйте компонент в проект только пред тем как собираетесь его использовать;
- Устанавливаете отношения между компонентами в рамках процесса, обеспечение которого вы программируете;
- Используйте один и тоже компонент для нескольких процессов. При этом не забывайте по-разному обрабатывать событие компонента, с учетом активного процесса;
- Используйте паки для сборки специальных компонентов обеспечивающих один процесс или тип действий;
- Переименовывайте компоненты в имена обозначающие назначение компонента в программе;
- Помните, что скорость работы компонентов может изменяться от различных условий, поэтому структура процесса не должна базироваться на ожидаемой последовательности исполнения, а на факте подтверждения изменения состояния компонента;
- Для упрощения обработки емких процессов используйте специализированный объект «логический анализатор», позволяющий формировать новый компонент системы;
- Не забываете, что параллельное исполнение нескольких процессов производиться в рамках единой очереди.

Отладка и тестирование

Среда разработки программ LP-Studio предоставляет средства для проведения процедур отладки и тестирования проектируемых программ на предмет выявления ошибок. Разработка программ на различных языках программирования сопряжено с выявлением нескольких типов ошибок – технических и логических. Технические ошибки возникают вследствие нарушения синтаксиса языка программирования, типизации данных и т.п. Логические ошибок это ошибки, при которых алгоритм поведения программы неверный или создан на основе неправильных представлений о действительности.

LP-программы могут иметь один тип ошибок - *логические*. Технические ошибки исключаются за счет применения выверенных компонентов и правил организации сети решения, в которой исключаются неверно заданный синтаксис записи строк кода.

Выявление логических ошибок в проектируемых программах основано на применении специальных приёмов, позволяющих найти причину неверной работы программы. Методика приёмов направлена на уменьшение времени отладки за счет проверку условий работы компонентов, а также правильности исходных связей и их последовательности.

Методика тестирования основана на нескольких механизмах:

- *Очередь сообщений* – предоставляет информацию о фактической очереди изменения состояния компонентов;
- *Зоны влияния* – предоставление информации о группе событий подчиненных компонентов;
- *Трассировка – журнал* – предоставление полной информации о ходе работы программы с предоставлением информации о характере выполняемой связи и об информационных потоках между компонентами в виде списка;
- *Трассировка – Таймлайн* – предоставление информации о характере выполняемой связи и об информационных потоках между компонентами в виде сетевого графика;
- *Выделение компонента* – механизм выделения всех строк в журнале событий с именем компонента.

К средствам отладки программы также относиться механизм формирования графического представления сети связи компонентов в виде графических примитивов.

Отдельно представлена отладка компонентов разработанных средствами автоматизированного проектирования в компоненте «Логический анализатор». Данный компонент предоставляет собственные средств отладки. Методика тестирования работы компонента «Логический анализатор» основана на механизмах:

- *Контроль состояния элементов компонента* – предоставление информации о текущем состоянии элемента, его частей (содержимое переменных, массивов, состояние переключателей и т.п.);
- *Список событий* – фактический ход исполнения алгоритма логического анализатора;
- *Выделение событий в списке по элементу* – механизм выделения в журнале событий всех строк связанных с компонентом;
- *Выделение связей* – механизм выделения в журнале событий всех строк по связям;
- *Точки остановки* – механизм обеспечивающий паузу исполнения событий компонента в заданной связи.

Появление технических ошибок в рамках работы LP-программ может возникнуть по причине сбоя работы компонента. Исправление данной ситуации находиться в ведении программиста, разрабатывающего компонент к платформе LogicProgram.

Компоненты

Компонент платформы LogicProgram является наименьшей величиной в LP-программе. Использование компонента позволяет программисту оперировать логикой работы проектируемой программы через формирование сети отношений между элементами системы. При этом каждый экземпляр компонента в приложении может иметь собственную конфигурацию работы. Все компоненты относятся к одному из двух классов: *интерфейсные* и *специальные*.

Следует помнить, что LP-программа может быть создана и без интерфейсных компонентов, например как сервис: «Web сервер, предоставляющий HTML файлы».

Наличие специальных компонентов в проекте является обязательным, так как именно работа компонентов данного класса обеспечивает исполнение логических операций, расчетов и управления. Один из специальных компонентов – **LPMachine** является первым компонентом, который должен быть включен в проект, так как именно наличие данного объекта обеспечивает функционирование программы под управлением выбранной операционной среды.

Интерфейсные компоненты

Наличие интерфейсных компонентов не является обязательным для разрабатываемой программы.

Организация интерфейсов взаимодействия человека с программой оформляется через компонент «Окно». Все интерфейсные компоненты могут быть использованы только в рамках компонента «окно». Структура окна с компонентами в нем представляется в виде собственной группы в дереве проекта.

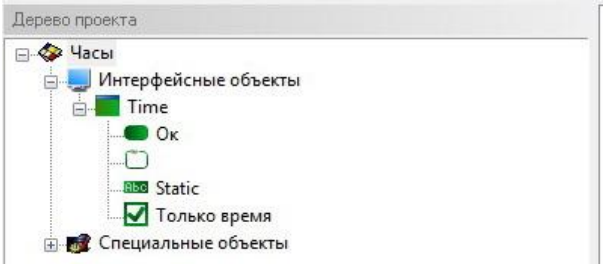
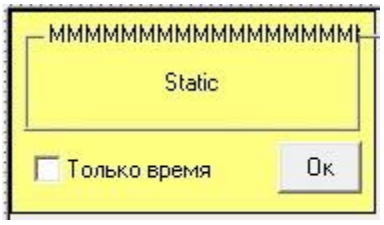


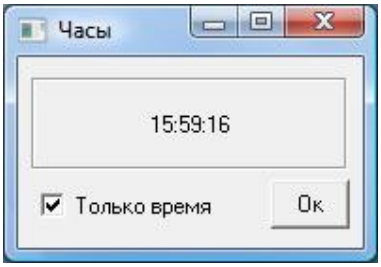
Рисунок 11 Пример дерева проекта

Представление компонентов в объекте «окно» отражается в закладке LP-Studio «Редактор форм».

Пример организации компонентов в объекте "Окно"



Пример предоставления «окна» пользователю программы



Для организации различных схем взаимодействия окон в рамках работы программы, компонент окно имеет свойства позволяющие формировать следующие схемы:

- *Всплывающие окна*– вспомогательный режим работы окна, предназначен для установки отношений с родительским окном, работает вне области окна родителя.
- *Перекрывающие окна* – основной режим предоставления окна.
- *Дочернее окна* – вспомогательный режим работы окна, предназначен для установки отношений с родительским окном, работает в области окна родителя.

Специальные компоненты

Специальные компоненты обеспечивают решение по управлению файлами, расчетами, взаимодействием с оборудованием, организации взаимодействия через web-технологии. Наличие специальных компонентов в программе обязательно.

Наличие компонента LPMachine в проект **ОБЯЗАТЕЛЬНО.** Данный компонент предназначен для организации взаимодействия LP-программы с LP-машиной. Наличие объекта в проекте должно быть в единственном экземпляре.

В дереве проекта специальные компоненты могут группироваться. Для этой цели предоставляется механизм создания группы и перенос требуемых объектов в группу. В техническом плане наличие групп не предоставляет, каких либо дополнительных средств управления, и служит исключительно для повышения удобства восприятия человеком структуры проекта.

Взаимодействие с компонентом

Каждый компонент платформы обладает интерфейсами взаимодействия с ним. Наличие интерфейсов позволяет организовать требуемое управление компонентом, получение сообщений об изменении его состояния, обеспечить обмен данными с ним. В зависимости от назначения компонента количество предоставляемых интерфейсов может быть от одного до четырех. Полный список интерфейсов конкретных компонентов платформы LP и их назначение см. «*Спецификация компонент*».

Тип интерфейса определят его назначение, а также характер связей, который может быть установлен с ним.

Тип интерфейса	Назначение интерфейса	Интерфейс, с которым допускается устанавливать связь
Событие	Список событий сигнализирующих о характере изменений в состоянии компонента. (*Т.е. реакция на команды*)	Команда
Команда	Список команд позволяющих изменить состояние компонента.	Событие
Входные данные	Список приемников данных требуемых для работы компонента. (*Тип данных*	Выходные данные

	поставляемых значений определяется структурой компонента)	
Выходные данные	Список информационных источников компонента. (*Тип данных определяется структурой компонента*)	Входные данные

Интерфейс является базовым средством организации взаимоотношений между компонентами. Вне интерфейсных связей не может быть установлено взаимодействие объектов в системе. Характеристики интерфейсов представлены в виде набора всех возможных способов взаимодействия с компонентом. Назначение характеристик представлено в описании свойств компонентов. См. «*Справочник по компонентам платформы LogicProgram*». Однако общий принцип работы интерфейса соответствует его типу.

Интерфейсы компоненты

Общее назначение и механизм работы интерфейса по типу связей определяет их работу в общем механизме функционирования программы. Для каждого компонента платформы назначение типа интерфейса неизменно, и обозначает единый принцип представления в системе.

Интерфейс «Событие»

Событие – основной способ получения информации об изменении состояния компонента. Каждый компонент, выполняющий какую либо работу, сообщает системе о данном событии. Так если пользователь нажимает на интерфейсный компонент, например «кнопка», то в общей очереди событий возникает информация о данном процессе. Если на данное событие были связи с другими компонентами (команды), как команды к ним, то начинается выполнении цепочки связей. Событие является исполняющим механизмом работы системы, именно ее наличие активирует все установленные связи, которые в свою очередь производят активизацию собственных связей, и вся цепь событий и команд исполняется до последнего события в системе, для которого нет связей с другими компонентами. Исполнение цепи связей по исполнению определенного алгоритма называется «*процессом*». Отражение события в системе возникает в момент, когда компонент фактически изменил свое состояние.

Базовое свойство интерфейса – «событие», является активизацией команд компонентов системы, т.е. возможностью продолжения работы компонентов программы.

Интерфейс «Команды»

Команда – механизм, позволяющий изменить состояние компонента. Команда может быть активирована только со стороны какого либо события. Последовательность команд создает очередь исполнения, поэтому следует учитывать, что активизация команды не означает ее мгновенное исполнение, так как очередь в системе может уже содержать команды на исполнение. При этом время исполнения команды также зависит от ее последовательности в списке события. Например, при следующей структуре:

- *C: (Browser) Страница загружена→(Редактируемый текст) Загрузить текст*
- *C: (Browser) Страница загружена→(Логический аккумулятор) Установить состояние TRUE*

Очередь исполнения будет выглядеть:

1. Редактируемый текст:Загрузить текс
2. Логический аккумулятор:Установить состояние TRUE

Очередь событий:

3. Редактируемый текст: Успешная загрузка данных
4. Логический аккумулятор: Установлено состояние TRUE

Следует помнить, что последовательность очереди событий зависит от факта исполнения компонентом команды, и может не совпадать с очередью команд. Однако большинство команд исполняются в момент их активации.

Рассмотрим пример процесса, очередь событий исполнения в котором не имеет гарантированной последовательности. В окне программы расположено 3-и интерфейсных элемента «Browser». При открытии окна, каждому компоненту «Browser» задается команда загрузки URL страницы как команда на событие подготовки компонента к работе (Browser:Пустая страница загружена) Ответная реакция образует очередь на загрузку URL, где последовательность ответов не всегда будет соответствовать последовательности команд.

Так в примере очередь команд:

1. Загрузить URL – Яндекс;
2. Загрузить URL – Google;
3. Загрузить URL – Yahoo!.

Очередь событий в системе:

1. Яндекс – загружен;
2. Yahoo! – загружен;
3. Google – загружен.

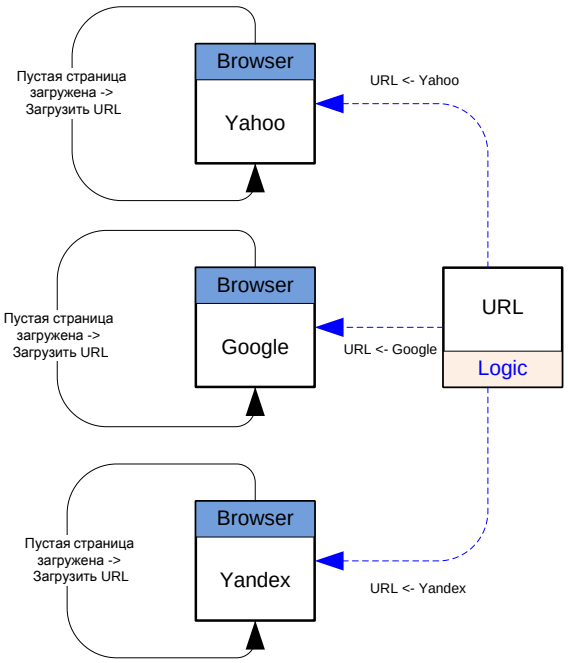


Рисунок 12 Пример организации очереди загрузки URL в компонентах Browser

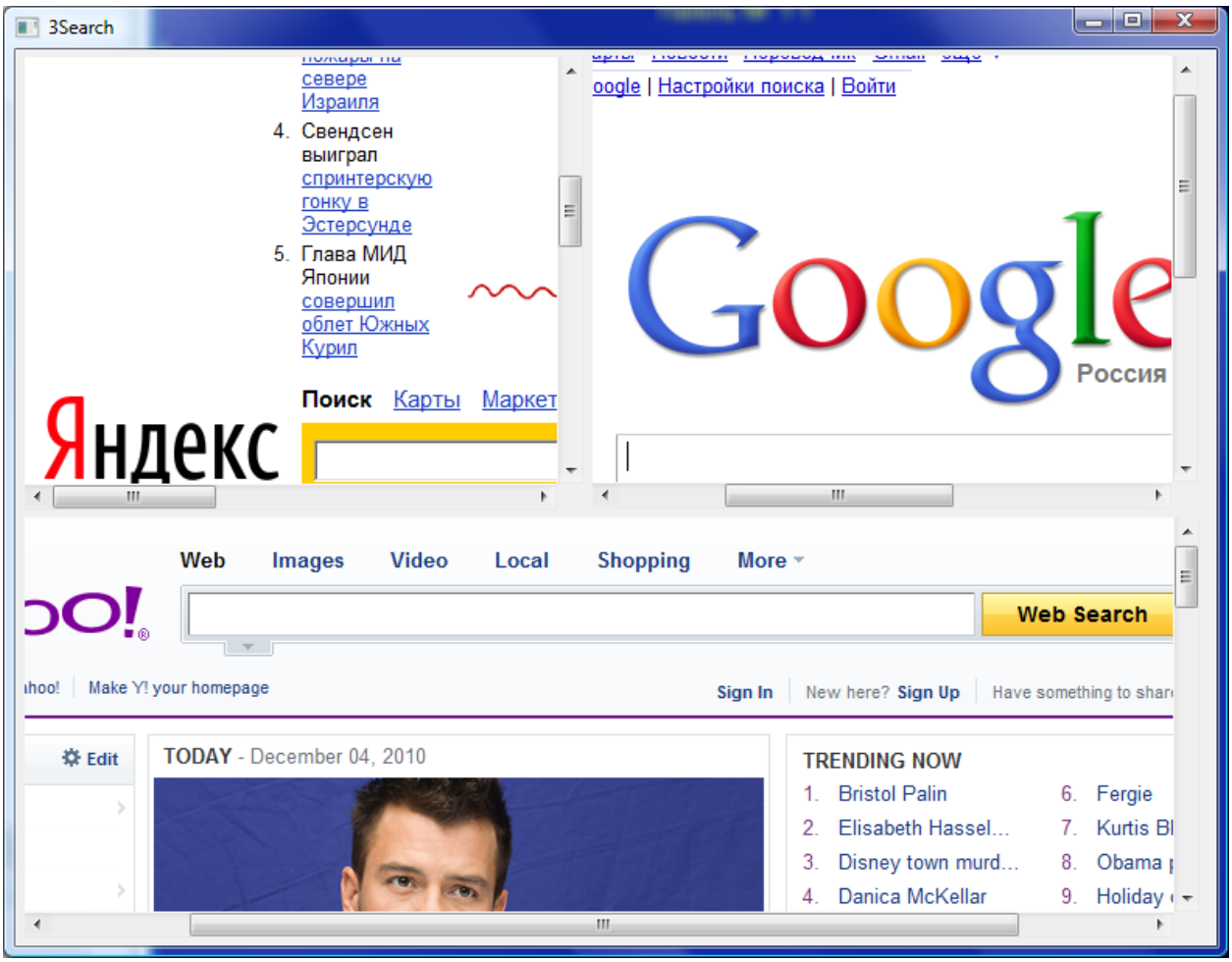


Рисунок 13 Пример работы программы

При проектировании процессов следует придерживаться следующих правил:

- Если у компонента есть ответная реакция на команду, то продолжение алгоритма работы следует строить от ответного события;
- Если на команду имеются события, как на успешное, так и на ошибочное исполнение, то следует учитывать это в проектируемом алгоритме;
- Если на команду в компоненте нет ответного события, то исполнение команды происходит гарантировано, в момент её активизации;
- Если компонент используется для разных процессов, то следует организовывать «ветвление» алгоритма на событие исполнения команды, с тем что бы исключить внепроцессорное срабатывание цепочек.

Базовое свойство интерфейса – «команда», является возможностью изменения состояния компонента, т.е. активизацией его работы.

Интерфейс «Входные данные»

Входные данные – механизм, обеспечивающий передачу данных в компонент. Наличие данного механизма обеспечивает организацию потоков данных между компонентами. При этом связь по данным между компонентами является самостоятельной цепочкой отношений независящей от цепочки отношений по управлению состоянием компонентов.

Интерфейс взаимодействия – «входные данные» может получать значения только от одного источника. Смена источников данных для одного приемника осуществляется с помощью специальных компонентов. Например, через специализированный объект «Маршрутизатор потоков данных -2». Данный специализированный объект создан средствами проектирования компонентов платформы LogicProgram. И имеет следующую структуру:

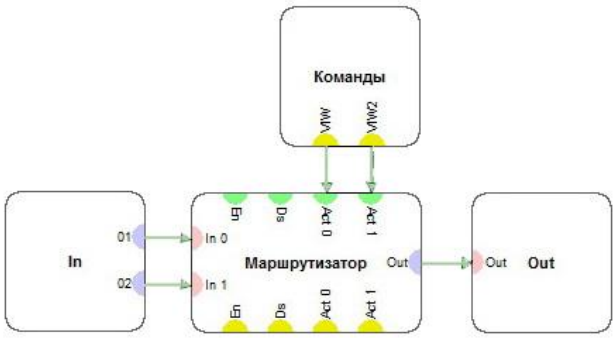


Рисунок 14 Пример организации компонента "Маршрутизатор..." (в графическом представлении среды проектирования компонентов платформы LP)

Компонент обеспечивает прием данных от двух источников и организацию передачи значений по заданному маршруту в другой компонент. Наличие подобных средств обеспечивает выполнения требования на «прием данных только от одного источника», с необходимостью получения данных от N-источников.

При работе с поступающей информацией следует помнить, что момент подачи команды на загрузку данных, и время фактического получение значения может иметь продолжительный разрыв. Например, при получении значений с устройства, предоставляющего данные с большой скоростью изменения. При поступлении команды на загрузку информации, значения в источнике могут быть иными, чем в точке активизации команды загрузки значений. Решение подобных «задач» достигается применением специальных методик:

- появление информации расценивается системой как событие;
- поддержка множественных входящих связей.

Подробно см. «*События по Входным данным*».

Базовое свойство интерфейса – «Входные данные», является возможностью получения компонентом набора значений необходимых для его работы от других объектов программы.

Интерфейс «Выходные данные»

Выходные данные – механизм, обеспечивающий передачу данных из компонента. Наличие данного механизма обеспечивает организацию потоков информации между компонентами программы. Характеристика интерфейса взаимодействия – «выходные данные» может предоставлять значения одновременно более чем для одного получателя. Наличие данного механизма позволяет организовывать требуемые структуры потоков данных в системе без применения дополнительных организационных средств.

Например, один источник данных, компонент «Таймер» может предоставлять текущее значение таймера сразу нескольким компонентам, с каждым из которых установлена связь по данным с этим объектом.

События по Входным данным

Информационные потоки, обеспечивающие алгоритмы работы программы могут также являться и управляющими элементами системы, как и связи управления. Использование механизмов управления информацией в системе позволяет минимизировать количество дополнительных компонентов в программе, упростить проектирование связей, упростить всё решение в целом.

Общий арсенал механизмов управления информационными потоками в LP-программе:

- Множественные входящие связи, т.е. когда с интерфейсом «Входные данные» установлено более одной связи;
- Использование факта поступления информации, как события, позволяющего формировать управляющую цепочку работы.

Множественные входящие связи

Согласно правилу интерфейс «Входные данные» может получать значения только от одного источника, в механизме событий по Входным данным имеет расширение, согласно которому, наличие нескольких входящих связей может обеспечить прием данных от N-источников.

Это достигается механизмом «*рассылки*» (принудительной передачи) значений от источника данных. При использовании данного подхода, возможно организовать получение значений, без использования дополнительных средств маршрутизации потоков данных.

Например, в программе представлено два интерфейсных компонента «Редактируемый текст» (Текст1/Текст2), а также компонент «Не редактируемый текст» (Отображение). Наличие 3-х кнопок обеспечивают передачу значений:

- «Загрузить текст-1» – загружает значение из компонента «Текст-1» в объект «Отображение»;
- «Загрузить текст-2» – загружает значение из компонента «Текст-2» в объект «Отображение»;
- «Загрузить» - загружает значение из компонента «Текст-1» в объект «Отображение»;

Компонент Отображение имеет связи:

1. *Д: (Edit) Текст1→(Static) Отображение*
2. *Д: (Edit) Текст2→(Static) Отображение*

Т.е компонент «Отображение» имеет 2-е входящие связи на один приемник.

Пояснение работы:

- нажатие кнопки «Загрузить текст-1» активирует команду компонента «Редактируемы текст-1»:Разослать текст, т.е. компонент начинает передавать данные из источника «Выходные данные» для всех компонентов имеющих связь с ним.
- Компонент «Отображение» имеет несколько связей по данным, в том числе и с этим объектом, вследствие чего он получает значения из компонента «Редактируемый текст-1».
- Тоже происходит и при нажатии кнопки «Загрузить текст-2», только для компонента «Не редактируемый текст-2», в этом случает компонент «Отображение» получает значения из другого компонента, при этом нет необходимости в организации маршрутизации данных из 2-х источников по средствам дополнительного компонента, обеспечивающего маршрутизацию данных.

Однако работа кнопки «Загрузить» будет происходить строго по правилу: «один приемник - один источник». В этом случае объект «Отображение» будет загружать только из источника «Текст-1» (при условии, что эта связь в списке связей приемника является первой.)

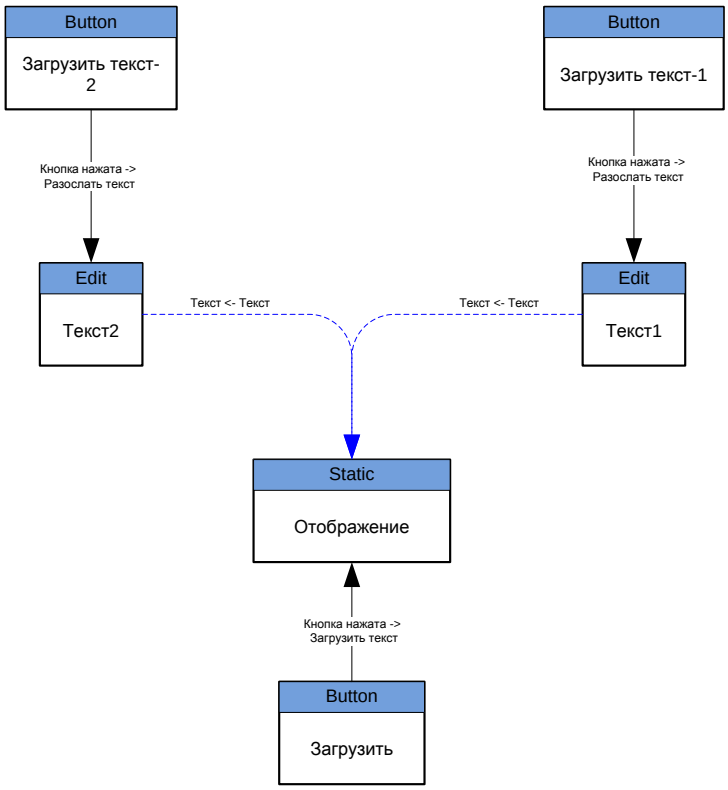


Рисунок 15 Пример организации схемы «N-источников - один приемник»

Использование механизма «*рассылки*» значений компонентом источником, обеспечивает передачу в компоненты приемники информации, согласно установленным связям, без каких либо дополнительных механизмов маршрутизации.

Поступление информации как события

Одним из уникальных механизмов платформы LogicProgram является возможность организации связей по данным между компонентами, с получением событий позволяющих поддержать требуемое управление поведением программы.

Данный механизм позволяет получать событие о том, что в компонент, от источника поступили данные и при необходимости продолжить требуемую цепочку связей управления. Применение этого способа обеспечивает решение задач синхронизации момента получения данных с точкой их обработки. Таким образом, факт рассылки данных от источника может быть переведен в событие системы.

Стандартные компоненты платформы не имеют механизмов трансформации факта поступления информации в компонент с предоставлением события об этом. Все события об изменении данных в стандартных компонентах являются подтверждением исполнения команды загрузки данных.

Организация механизма поступления информации как события возможна через проектирование требуемой структуры в компоненте «Логический анализатор».

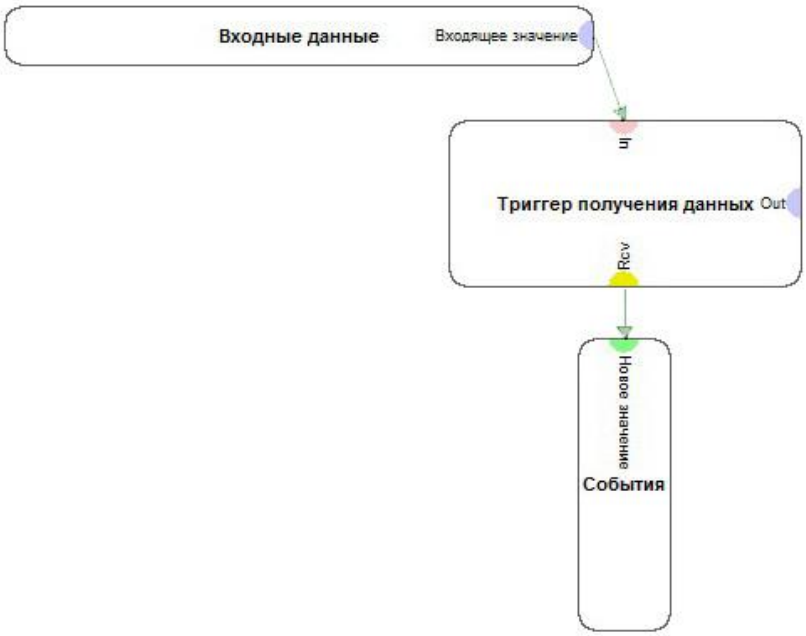


Рисунок 16 Пример организации трансформации факта поступления данных в событие в компоненте «Логический анализатор»

Работа механизма, в структуре проектирования логического анализатора, основана на следующем алгоритме:

- В момент получения данных в объекте «Входные данные» в точку приема «Входящие значения» происходит передача управления на объект «Триггер получения данных»;
- Объект «Триггер получения данных» активирует событие «Rev», которое передает управление на объект «События»;
- Объект «Событие» выдает в систему событие «Новое значение».

Следует помнить, что представленный пример проектируется средством автоматизированной разработки логического анализатора. См. «Подкласс языка LogicProgram». Использование компонента в программе будет выглядеть следующим образом:

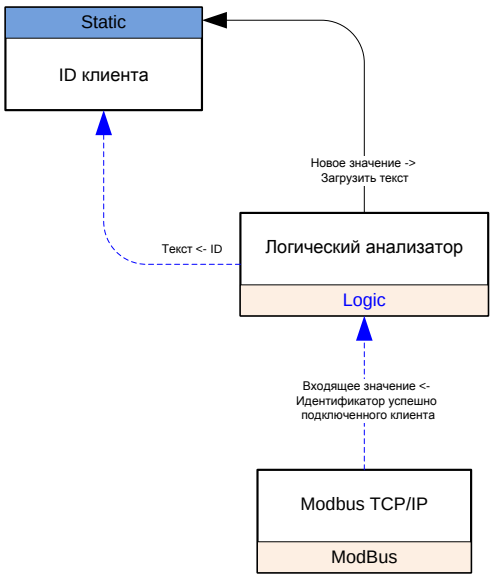


Рисунок 17 Пример работы компонента данные→Событие

В данном примере, в момент получения информации от компонента «Modbus TCP/IP», объект «Логический анализатор» формирует событие на загрузку значений в интерфейсный

компонент «ID клиента», который в свою очередь запрашивает значение из логического анализатора, так как в объекте «Modbus TCP/IP» может быть уже иное значение.

Правила проектирования собственных компонентов средствами объекта «Логический анализатор» см. «*Создание нового компонента*». Информация о работе компонентов платформы описана в разделе «*Справочник по компонентам платформы LogicProgram*».

Очередь исполнения в LP-программе

Очередь исполнения обеспечивает работоспособность программы, вне зависимости от её фактического состояния, например программа в момент появления нового события уже может совершать какое либо действие. Наличие механизма очереди обеспечивает запуск команд согласно их расположения в точке запуска. Команды могут исполняться относительно друг друга несколькими способами:

- *Исполнение по порядку*: команды запускаются на исполнение в том порядке, в котором они расположены в очереди и завершаются так же по порядку. То есть команды выполняются синхронно.
- *Непоследовательное исполнение*: команды отправляются на исполнение по порядку, но не ждут завершения предыдущей команды перед началом исполнения. То есть команды выполняются асинхронно.

С одним событием может быть связано несколько команд, образующих очередь. Такая последовательность исполняются как есть, если все команды синхронные, то и события от них появятся в той же последовательности. Если среди них были асинхронные команды, то ответы от них появятся по факту исполнения и могут быть «внедрёны» в последовательность событий от других синхронных команд.

Отправка и получение данных происходит синхронно, и не отражается в очереди исполнения программы.

Большинство команд компонентов платформы LogicProgram являются синхронными, более точная информация представлена в «*Справочник по компонентам платформы LogicProgram*».

Очередь исполнения в компоненте «Логический анализатор»

Компонент «Логический анализатор» представляет собственную программную структуру, также построенную на механизме очереди. Следует помнить, что передача управления данному компоненту останавливает обработку общей очереди программы до момента завершения его работы. Что следует учитывать при создании программ обработки средствами компонента «Логический анализатор.

Механизм обработки очереди исполнения в логическом анализаторе отличается от механизма обработки очереди программы. Принцип обработки очереди в компоненте основан на стековом механизме.

Стек вызовов хранит информацию для возврата управления из подпрограмм (процедур) в программу (или подпрограмму). Объем, стека имеет ограничение на глубину вызовов. В системе обработки программ созданных в компоненте «Логический анализатор» размер стека - 10.

Общий принцип обработки стека вызовов обеспечивает последовательное исполнение связей от первого события к первой команде, далее (если иметься связь) обрабатывается событие от первой команды к следующей команде и т.д., до момента отсутствия связей по событию. Далее если у первого события была очередь команд, то следующая команда обрабатывается также по всей цепочке связей.

Основные требование проектирования структуры программы в «Логическом анализаторе», при котором понимание очереди исполнения становиться особенно важно:

- некоторые субкомпоненты обнуляет стек (т.е. их применение должно быть последним в структуре алгоритма компонента);
- некоторые конструкции циклической обработки увеличивать стек. Для этой цели имеются специализированные субкомпоненты. Подробности см. «*Очередь данных*».

Дополнительно

Тип данных

При работе с информацией, имеющей разный тип данных, платформа LogicProgram проводит автоматическую типизацию значений по принципу: «привидение значений к типу первого значения в операции». Так если при попытке использования математической операции, над значениями содержащими символы, результат будет строковым, а математическая операция выдаст ошибку обработки. При этом операция со значениями, представленных строками, но содержащих только цифры, будет произведена успешно, так как будет произведена автоматическая типизация символов в цифровое значение.

Помимо автоматической типизации данных, может быть использована явная типизация. Для этой цели используется правила описания переменных содержащих значения в компоненте «Логический анализатор». Правила явной типизации см. «*Правила определения типа данных*».

Математические операции

Организация математических расчетов оформляется в рамках проектирования компонента «Логический анализатор». Среда автоматизированного проектирования алгоритма работы компонента, предоставляет следующие математические операции:

- сложение;
- вычитание;
- умножение;
- деление.

Тип данных и результат расчетов зависит от характера значений используемых в операции. Точность получаемого значения определяется точностью первого значения.

Например:

Первое значение	Операция	Второе значение	Результат
INT(2);	деление	DOUBLE(7);	INT(2);
DOUBLE(2);	деление	DOUBLE(7);	0.285714
2	деление	7	0.285714
LONG(2);	деление	DOUBLE(7);	0.285714

В рамках обработки математических операций поддерживается максимальная точность расчетов. Т.е. передача полученных значений из компонента в компонент не приводит к снижению точности значений. Однако отображение значений ограничивает точность строки символов до 6 знаков после запятой.

Строковые операции

Строковые операции в платформе LP представлены механизмом типизации данных и специализированным механизмом логического анализатора – «регулярные выражения».

Приведения значений к строковому формату возможно через описание переменной в синтаксисе присвоение соответствующего типа данных. Тип данных:

- строка;
- дата-время;
- дата;
- время.

Также строковые переменные могут содержать символы выравнивания, и переносов. Подробности см. «*Правила определения типа данных*».

Регулярные выражения – механизм позволяющий проводить действия над строками, например, выделять часть строки. Подробности работы см. «*Краткая справка по регулярным выражениям*».

Библиотеки компонентов

Базовые компоненты языка

Платформа LP представляет программисту набор компонентов позволяющих формировать структуру программы. При этом имеется возможность расширения новыми компонентами, которые могут быть созданы как с использованием языков высокого уровня, так и с использованием встроенной системы автоматизированного проектирования компонентов.

Расширение компонентов языка

API к «платформе LP»

Платформа LP предоставляет возможность расширения компонентов платформы за счет создания собственных объектов.

Подкласс языка LogicProgram

«Логический анализатор» - это система автоматизированного проектирования новых компонентов. Проектирование осуществляется в специализированной графической среде представляющей единый механизм формирования требуемого многообразия вариантов новых компонентов.

Основа для проектирования представлена в виде «субкомпонентов», выполняющих различные действия. Все субкомпоненты разделены на классы:

- взаимодействие;
- логические операции;
- переменные и константы;
- управление;
- операции.

Подробности программирования логического анализатора см. «*Создание нового компонента*».

API к «Логическому анализатору»

Платформа LP предоставляет возможность расширения субкомпонентов автоматизированной системы проектирования логического анализатора, за счет создания собственных объектов.

Форматы хранения

В качестве формата хранения LP-программы используется расширенный язык разметки XML — расширяемый язык разметки. Структура разметки определена языком программирования платформы LogicProgram.

Проект

Проект LP-программы хранит информацию о компонентах их настройках, связях и комментарии программиста. Формат хранения – открытый, использование описания структуры языка LP позволяет создавать собственные средства представления кода программы. Подробности см. «Формат файла проекта».

Скрипт

Скрипт – псевдо компилированный файл проекта LP-программы, является «исполняемым» файлом в среде LP-машин. Большинство компонентов имеют представление на всех поддерживаемых операционных средах. Кроссплатформенность LP-программ обеспечивается на уровне единого формата записи проекта и на уровне выполнения за счет применения для каждой операционной среды собственной машины обеспечения. См. «*Среда исполнения*».

LP-Studio

Программирование в редакторе LP-Studio

Организация нового проекта LP-программы начинается с задания свойств «проекта». Так обозначение «тип проекта» будет указывать, какие компоненты из общей библиотеки подходят для использования в проекте. Доступ к режиму настроек свойств «проекта», возможно, осуществить через выбор верхнего узла компонентов в дереве проекта и активизацией закладки «*Основной режим*».

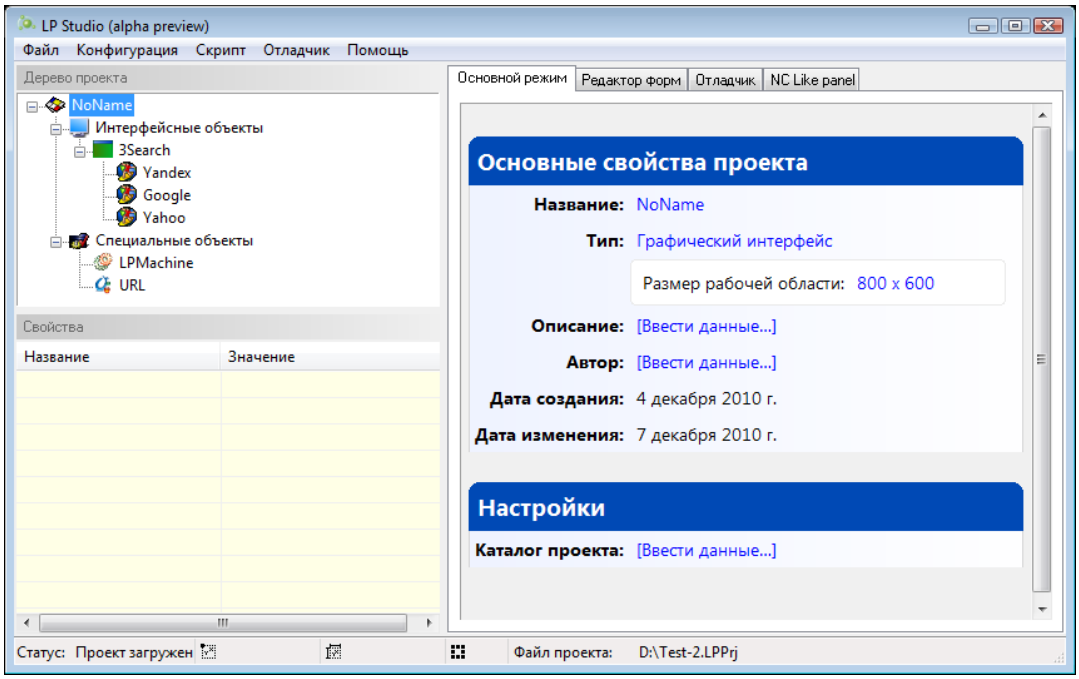


Рисунок 18 Пример выбора свойств проекта

Структура свойств «проекта»:

- *Название* – краткое описание назначения проекта
- *Тип* – характер разрабатываемого проекта. Следует помнить, что тип проекта предопределяет набор доступных компонентов платформы. Типы проекта:
  - *Графический интерфейс* – означает, что проект может использоваться под управлением всех поддерживаемых платформой операционных сред;
  - *Графический интерфейс (Windows)* – означает, что проект предназначен для работы только под управлением ОС-Windows;
  - *Графический интерфейс (x11)* – означает, что проект предназначен для работы только под управлением *nux;
  - *Консоль* – означает, что проект будет работать в консольном режиме;
  - *Портативное устройство* – означает, что проект будет работать на портативном устройстве.
  - *Размер рабочий области* – разрешение, в рамках которого планируется работа программы. Указывается в пикселях.
- *Описание* – предназначено для ввода информации о характере проекта
- *Автор* – предназначено для ввода данных об авторе проекта
- *Дата создания* – автоматическое заполнение, дата создания проекта
- *Дата изменения* – автоматическое заполнение, дата последней модификации проекта.

Область доступа к файлам

Платформа LogicProgram устанавливает определенное ограничение на доступ к файловой структуре.

Корневым каталогом считается каталог, в котором была запущена LP-программа. Осуществить доступ к файлам, выше данного каталога будет невозможно.

Для удобства ведения нескольких проектов в рамках LP-Studio, имеется возможность разграничить структуру файлов LP-программ. Данный механизм представлен в закладке «Настройки» основных свойств проекта.

Путь к проектам определяется настройками общего пути доступа к каталогам проекта, плюс директория указанная в качестве директории конкретного проекта. Для ввода персональной директории проекта задается только её название.

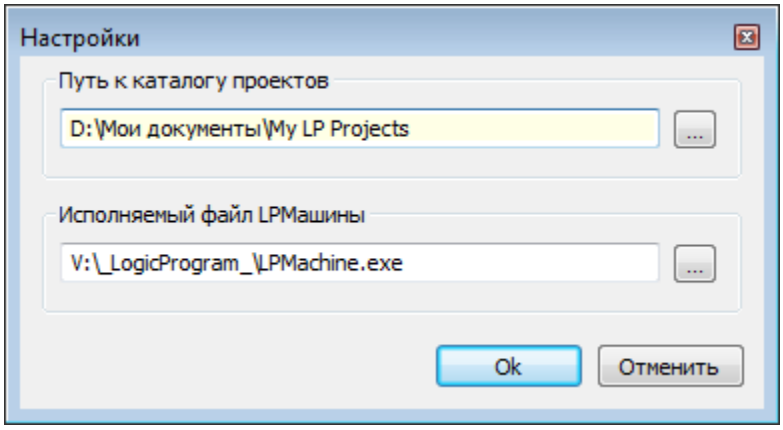


Рисунок 19 Режим ввода общего пути к каталогам LP-Studio

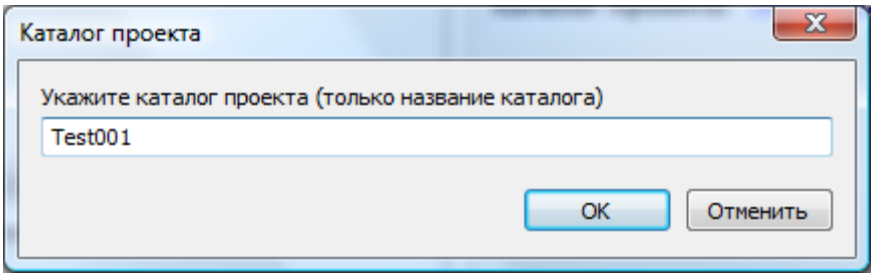


Рисунок 20 Режим ввода названия каталога проекта

Общие принципы формирования проекта

Общий принцип построения структуры проекта базируется на формировании дерева проекта и связей между его объектами.

Структура каждого проекта состоит из корня проекта и 2-х узлов: класс интерфейсных компонентов, и класс специализированных компонентов.

Добавление необходимого экземпляра компонента в проект осуществляется выбором узла класса компонента, активизацией команды «Добавить объект…».

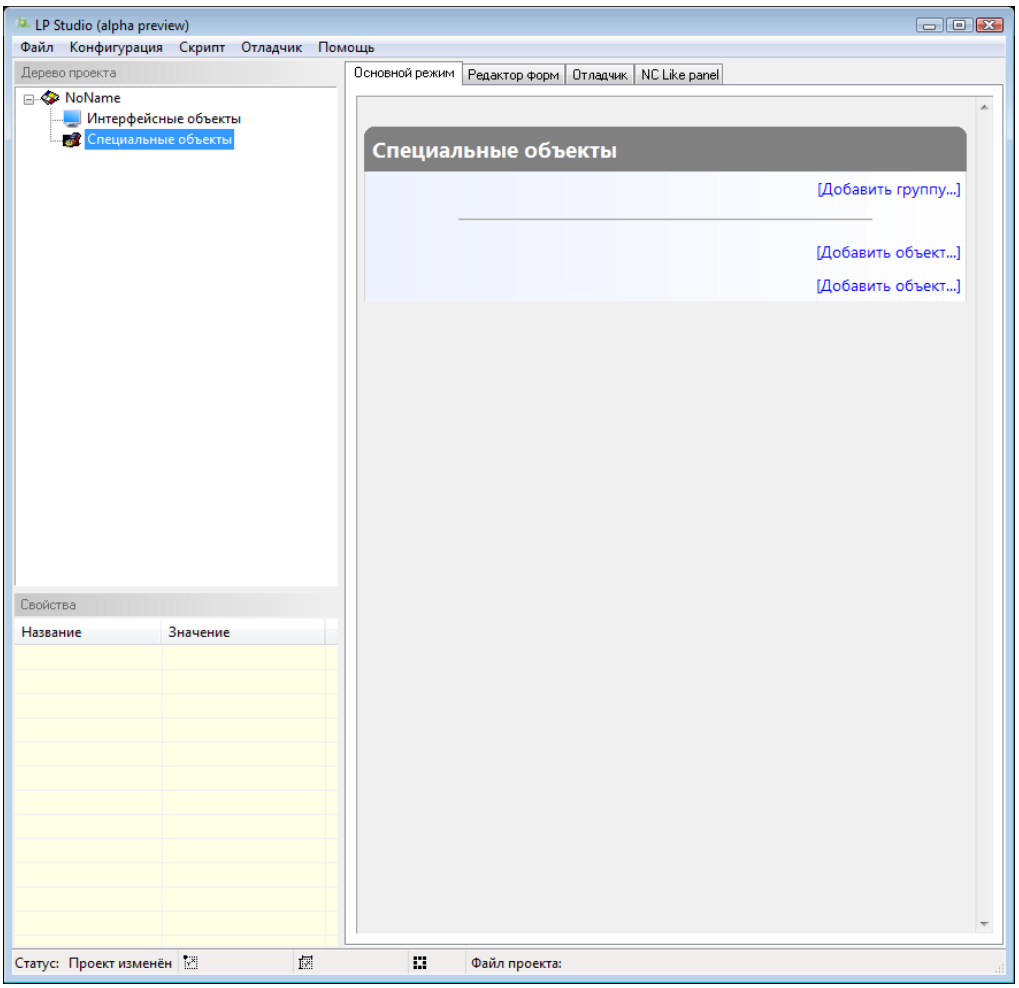


Рисунок 21 Пример добавления специализированного объекта в проект

Подтверждение выбора требуемого объекта в списке выбора, завершает включение в проект нового объекта.

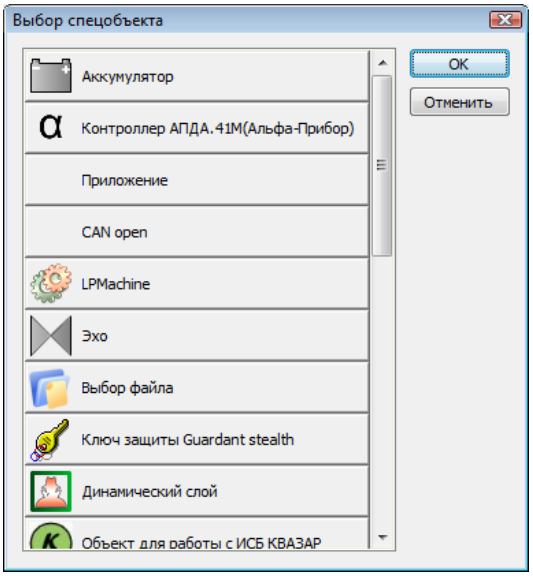


Рисунок 22 Пример списка базовых компонентов платформы LogicProgram

После включения в проект нового компонента, он отражается в дереве проекта. Далее при необходимости, возможно, произвести конфигурирование выбранного экземпляра компонента проекта.

Внедрение визуальных компонентов производиться через процедуру графического проектирования геометрии и области вывода компонента. Следует помнить, что все интерфейсные компоненты могут быть добавлены в проект только в рамках базового компонента – «*окно*». Количество окон в проекте неограниченно.

Внимание! Начальным компонентом в проекте должен быть – **LPMachine**, объект, отвечающий за организацию взаимодействия LP-программы с LP-машиной. Данный компонент должен быть представлен в единственном экземпляре.

Для удобства сборки компонентов в дереве проекта имеется возможность группы (только для специализированных компонентов). Данные сборки не несут алгоритмической нагрузки и предназначены только для удобства управления экземплярами компонентов в проекте.

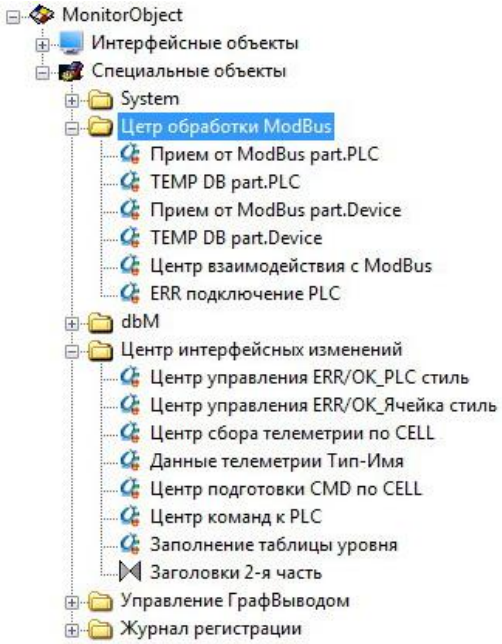


Рисунок 23 Пример организации групп для специальных компонентов

Дерево проекта

Дерево проектов отражает все компоненты программы. Структура дерева определяет разбиение объектов по их классам: интерфейсные, специализированные. Структура интерфейсных компонентов группируется по объекту – «окно». Специализированные объекты могут быть сгруппированы по любому принципу. При этом допускаются вложенные группы.

Область дерева проекта имеет две части:

- Непосредственно дерево проекта
- Свойства компонента выбранного в дереве проекта

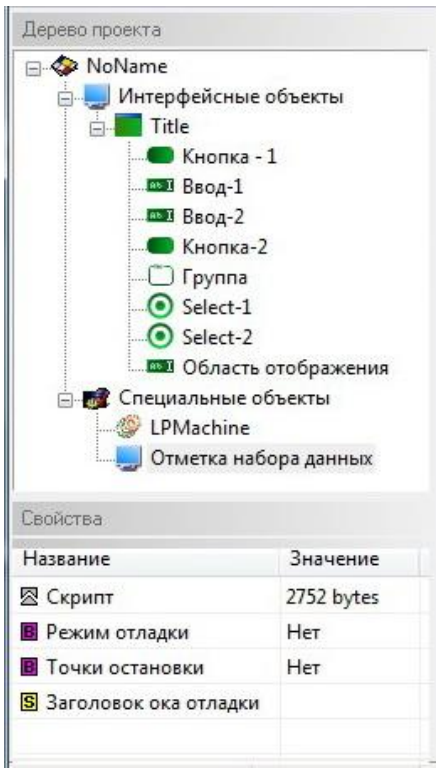


Рисунок 24 Пример отображения области "дерево проекта"

Модификация свойств компонента

Компоненты платформы LP могут иметь структуру свойств, настройка которых позволяет указывать требуемый режим работы конкретного экземпляра компонента в программе. Доступ к настройкам предоставляется в нижней части области отображения дерева проекта, для выбранного объекта.

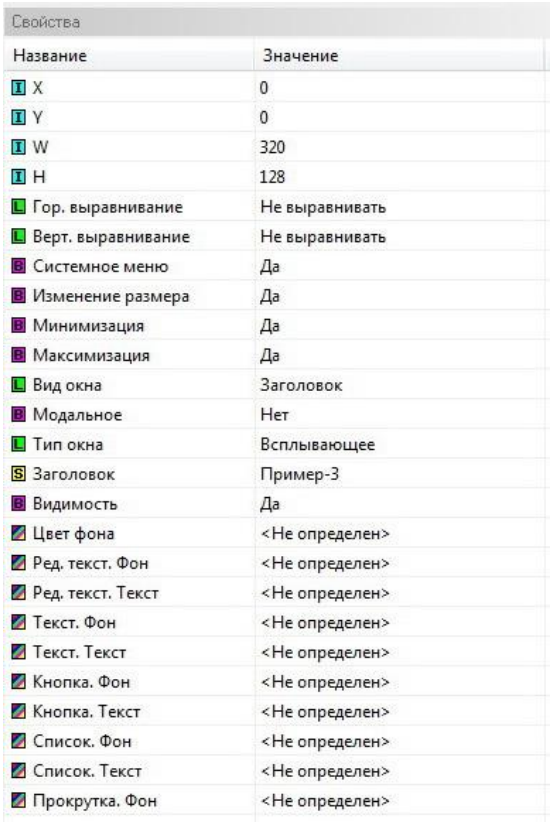


Рисунок 25 Пример отображения свойств компонента "Окно"

При включении нового экземпляра компонента в проект, его свойства имеют настройку «*по умолчанию*». Свойства объекта не могут меняться в процессе работы программы. При настройке свойств объекта, могут быть представлены новые разделы свойств, или некоторые свойства могут быть скрыты. Структура и назначение свойств, представлено в описании компонентов.

Режимы LP-Studio

Основной режим

«*Основной режим*» предоставляет базовый механизм программирования отношений между компонентами проекта. Режим позволяет осуществлять настройку проекта, добавлять новые элементы в проекте, вносить модификацию в последовательность связей, а также работу с ними: удаление, запрещение/разрешение на исполнение.

Процесс проектирования алгоритма работы компонентов производиться через организацию связей между объектами проекта. Формирование связей обеспечивают построение требуемых цепочек отношений. Например: Событие одного элемента может быть связано с командой другого элемента. В рамках работы программы при возникновении этого события произойдет активизация команды во втором компоненте, что приведет его в иное состояние. При этом изменение состояния второго компонента может сформировать собственные события. Если на эти события также есть связи с другими компонентами, то цепочка работы будет продолжена, вплоть до последнего компонента и события в нем на которое нет установленной связи.

На одно событие может быть установлена одна и более команд, последовательность которых формирует очередь их исполнения. Логическая цепочка «событий→команд» формирует процесс. Средства для просмотра и управления «процессами» не представляются. Понятие процесса используется для ограничения уровня обзора событий в системе обеспечивающих требуемый алгоритм работы.

Понятие процесса

Понятие процесса представлено в документации для пояснения методики программирования в LogicProgram. Назначение процесса – проектирование требуемого алгоритма поведения программы. При этом, основное свойство платформы – обеспечение простоты и ясности программы, при увеличении её сложности. Процесс – форма мышления при создании текущей итерации программы.

Пример работы нескольких процессов с одним набором объектов системы:

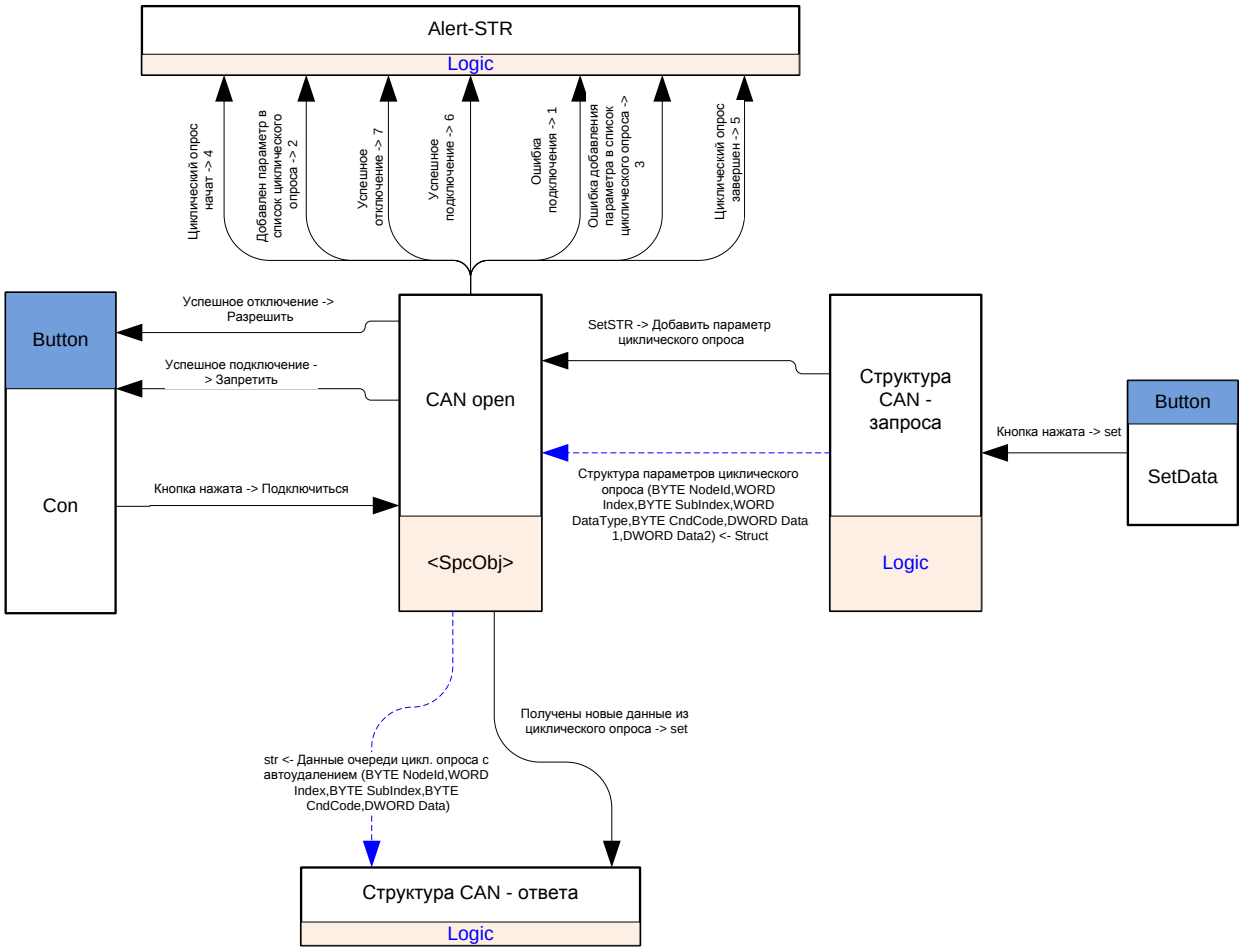


Рисунок 26 Пример пересечения процессов

Структура компонентов в примере обеспечивает исполнения нескольких процессов, каждый из которых направлен на решение определенного алгоритма.

Так, в представленном примере, реализованы алгоритмы, отвечающие за взаимодействие с оборудованием по протоколу CAN open. Один из алгоритмов (алгоритм-1) формирует структуру запроса к оборудованию, т.е. определенный процесс. Для обозначения его частей используется зеленый цвет и широкие линии:

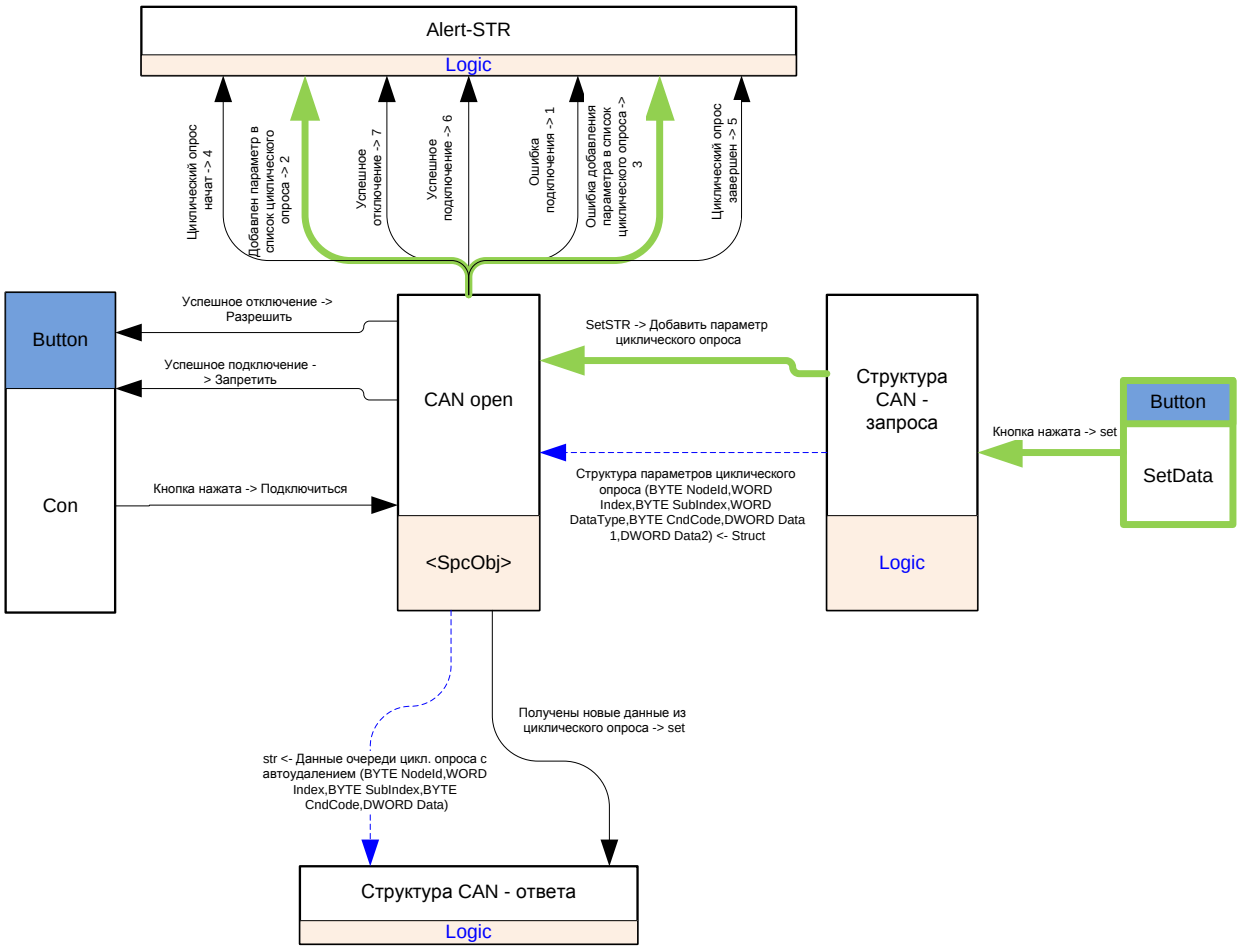


Рисунок 27 Пример связей процесса обеспечивающих алгоритм -1

Процесс (алгоритм-2), обеспечивающий получение информации от оборудования по протоколу CAN open реализован следующим образом (Выделен фиолетовым цветом и широкими линиями):

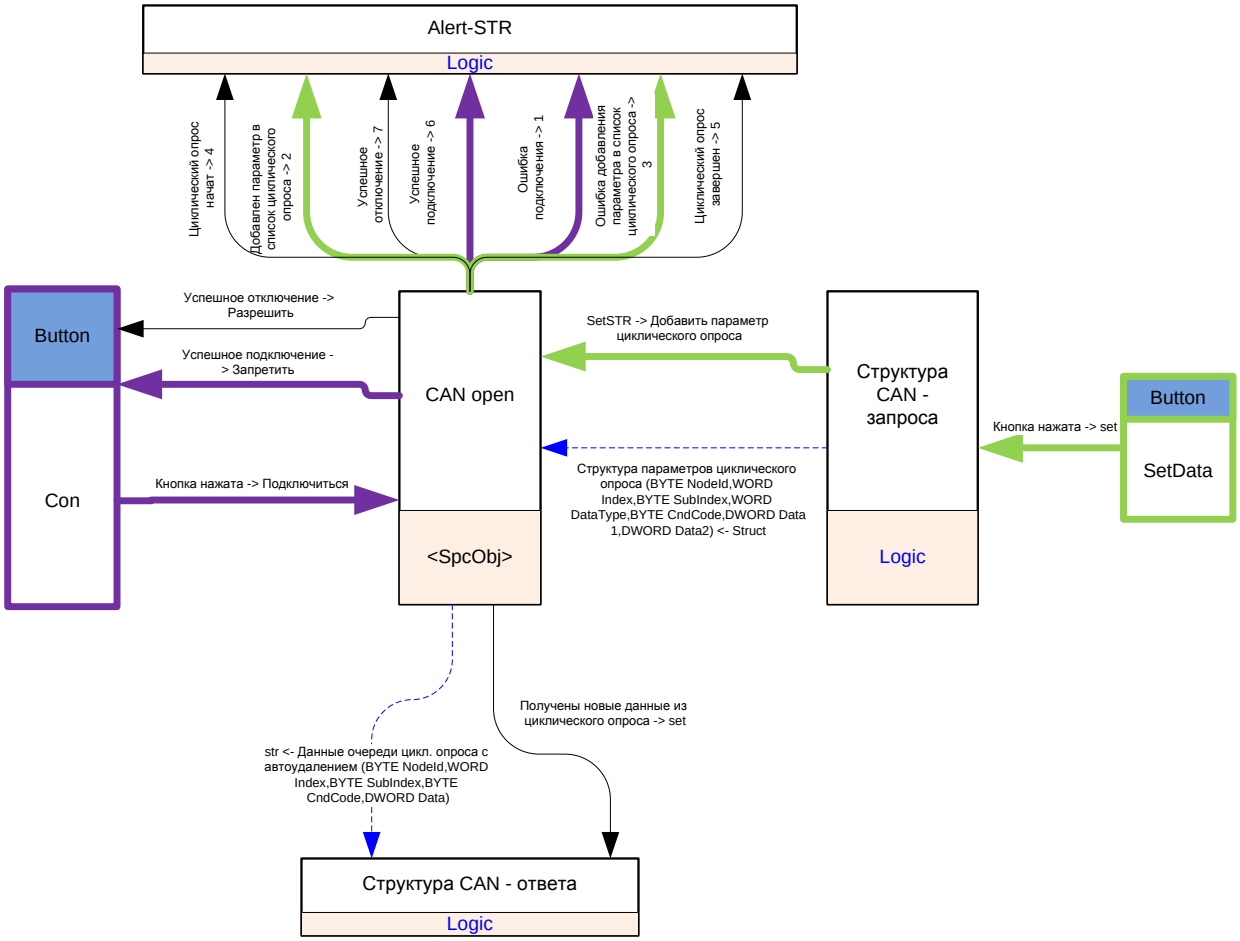


Рисунок 28 Пример связей процесса обеспечивающих алгоритм-2

Следует помнить, что понятие процесса используется только для обозначения цепочки связей в проекте.

Особое внимание следует уделять пересечению процессов, т.е. при использовании одних и тех же объектов, и событий от них, с тем, чтобы исключить «не ожидаемое» срабатывание компонентов (получение событий) вне рамок проектируемого процесса.

Пример. Программа должна обеспечить передачу введенной пользователем информации в единую область уведомления. Передача данных осуществляется нажатием соответствующих кнопок в окне программы. После приема данных необходимо выделить в интерфейсе программы, какой набор значений был передан в область уведомления.

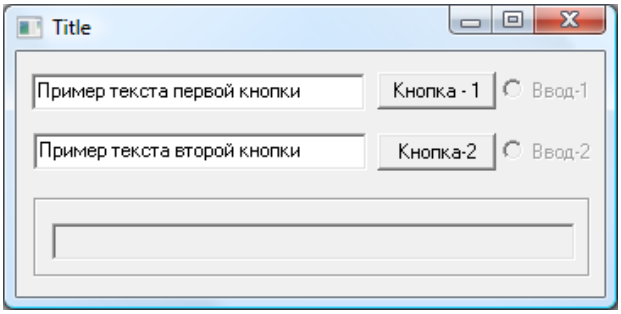


Рисунок 29 Пример программы

Алгоритм решения:

Д:(Ввод-1)Текст→(Область отображения)Текст
Д:(Ввод-2)Текст→(Область отображения)Текст
С:(Кнопка-1)Кнопка нажата→(Ввод-1)Разослать текст
С:(Кнопка-2)Кнопка нажата→(Ввод-2)Разослать текст
С:(Область отображения)Успешный приём данных→(Select-1) Отметить
С:(Область отображения)Успешный приём данных→(Select-1) Отметить

Структура алгоритма работы программы (процессы выделены собственными цветами):

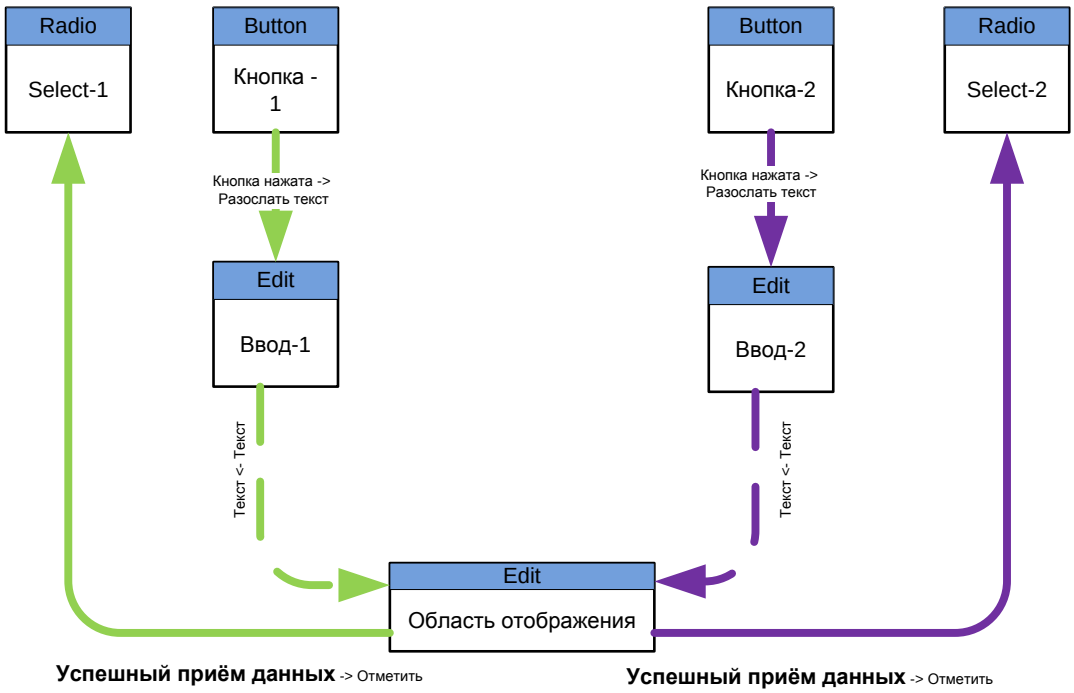
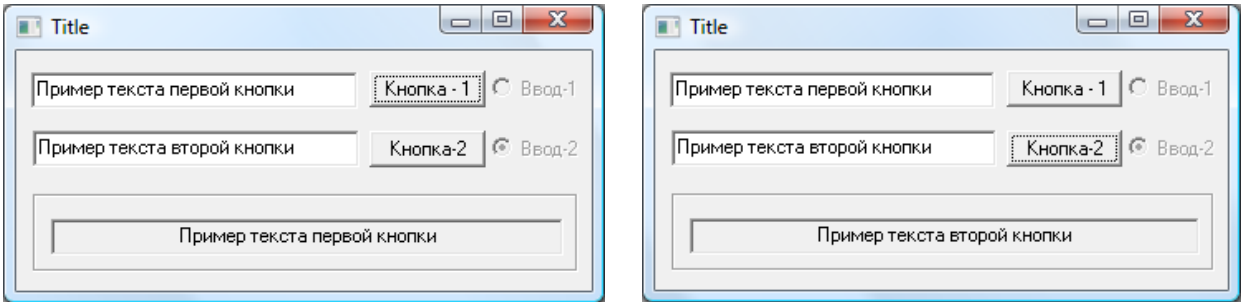


Рисунок 30 Пример пересечения процессов

В данном примере использование одного и того же события «Успешный приём данных» в компоненте «Область отображения» для выбора используемого набора значений не приведет к ожидаемому результату. Так как последовательность обработки команд по событию «Успешный приём данных» будет следующей:

- Отметить выбор в компоненте «Select-1»
- Отметить выбор в компоненте «Select-2»

Так как эти команды идут подряд, а оба компонента «Select-1/2» входят в одну группу, то после установки выбора во втором компоненте, отметка в первом исчезнет. Т.е не смотря на то, какой набор данных был предан, отметка всегда будет на компоненте «Select-2»



Пример работы алгоритма программы

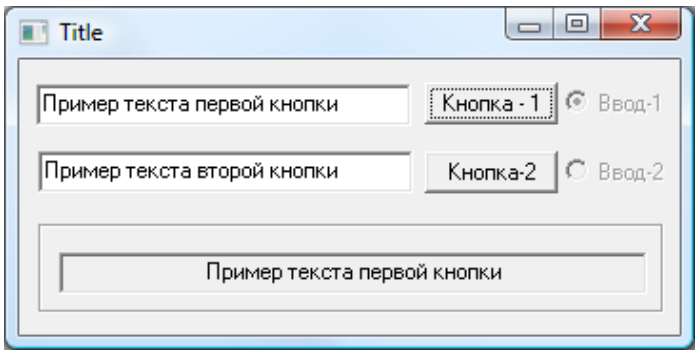


Рисунок 31 Пример работы измененного алгоритма

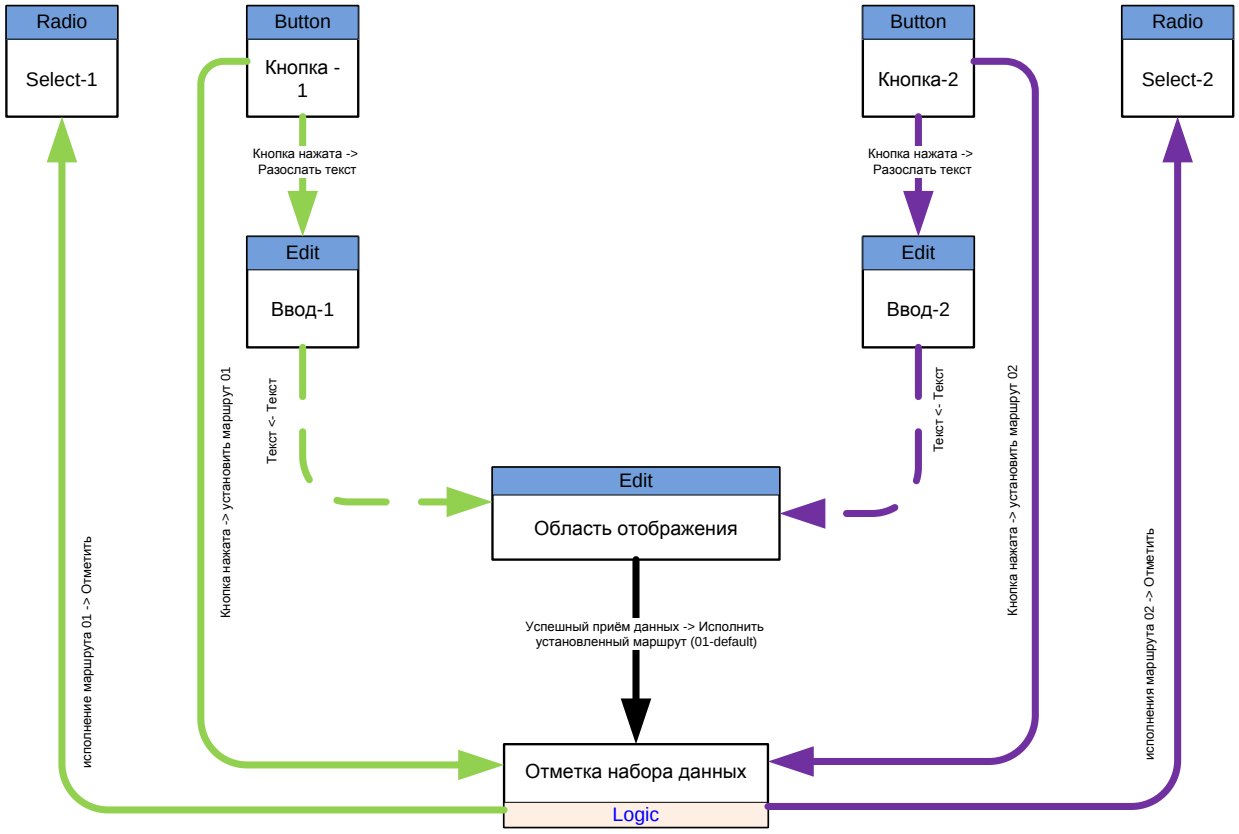


Рисунок 32 Пример решения проблемы пересечения процессов

Решение проблемы пересечения процессов, в данном примере, решено за счет использования дополнительного компонента, работа которого и обеспечивает корректный выбор объекта для указания набора данных в области отображения. Алгоритм работы компонента «Отметка набора данных» строиться на указании данному объекту, какое событие он должен послать в систему при обращении к нему. Активизация компонента производиться единой командой: «Исполнить установленный маршрут (01-default)», в ответ может появиться одно из двух событий:

- исполнение маршрута 01
- исполнение маршрута 02

Указание требуемого маршрута производиться при нажатии кнопки-1 или кнопки-2, таким образом, обеспечивается нужное поведение объекта «Отметка набора данных».

Работа в «Основном режиме» обеспечивает программиста информацией по всем интерфейсам объекта одновременно. В рамках единой страницы компонента для каждого интерфейса представлены собственные секции, в которых отражены все характеристики интерфейса и установленные связи.

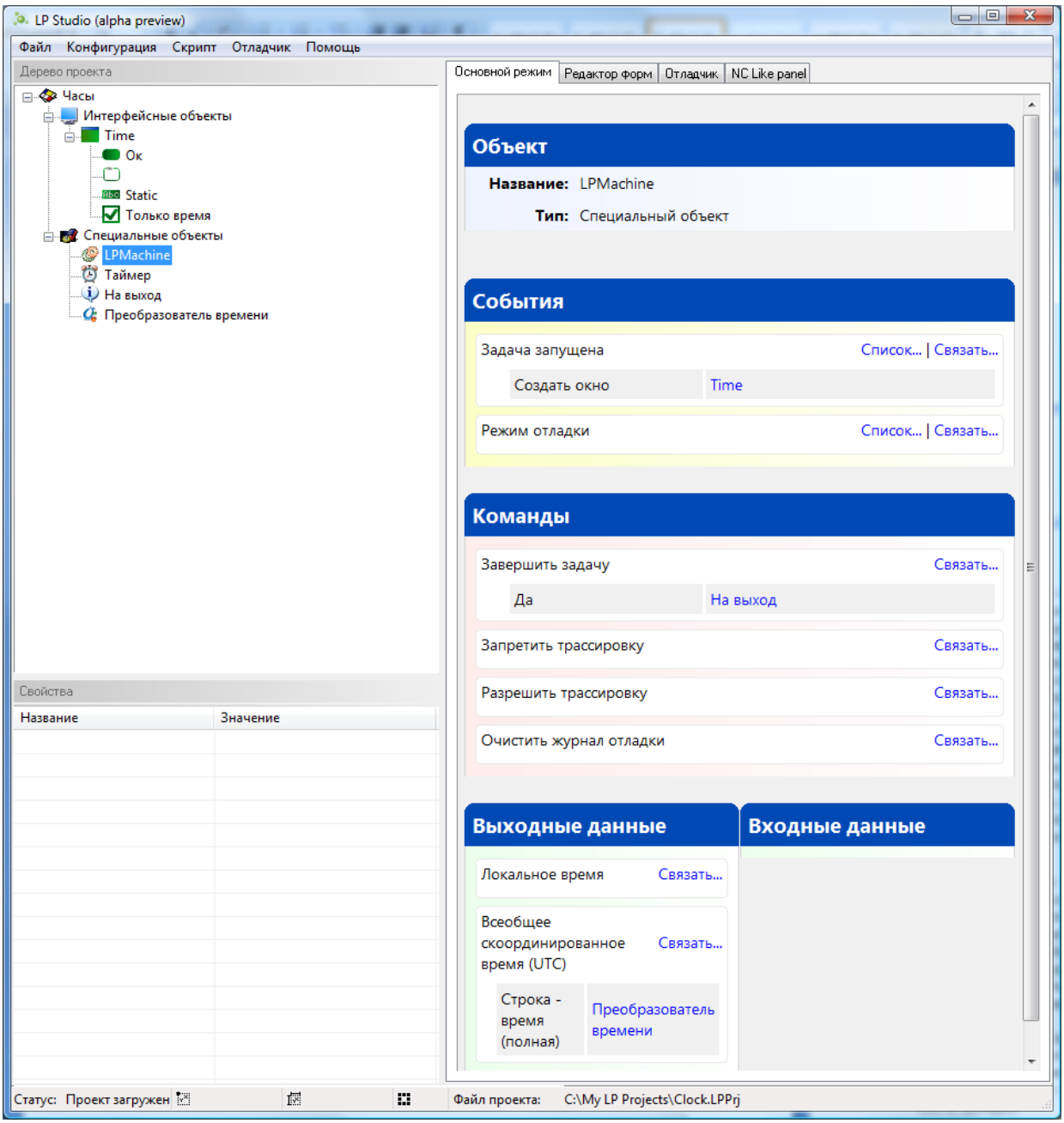


Рисунок 33 Пример программирования в "Основном режиме"

Наполнение закладок по интерфейсам зависит от типа компонента. Каждый интерфейс отображает полный список событий/команд/данных объекта. Характеристики интерфейсов, задействованные в связях, отображают последовательность связей в виде названия компонента и используемой характеристики.

Режим установки связи происходит через активизацию команды «Связать…» в нужной характеристике компонента. В ответ предоставляется окно для выбора нужного компонента и характеристики в нём. При этом, платформа предоставляет для выбора только допустимые для связи характеристики. См. «*Взаимодействие с компонентом*»

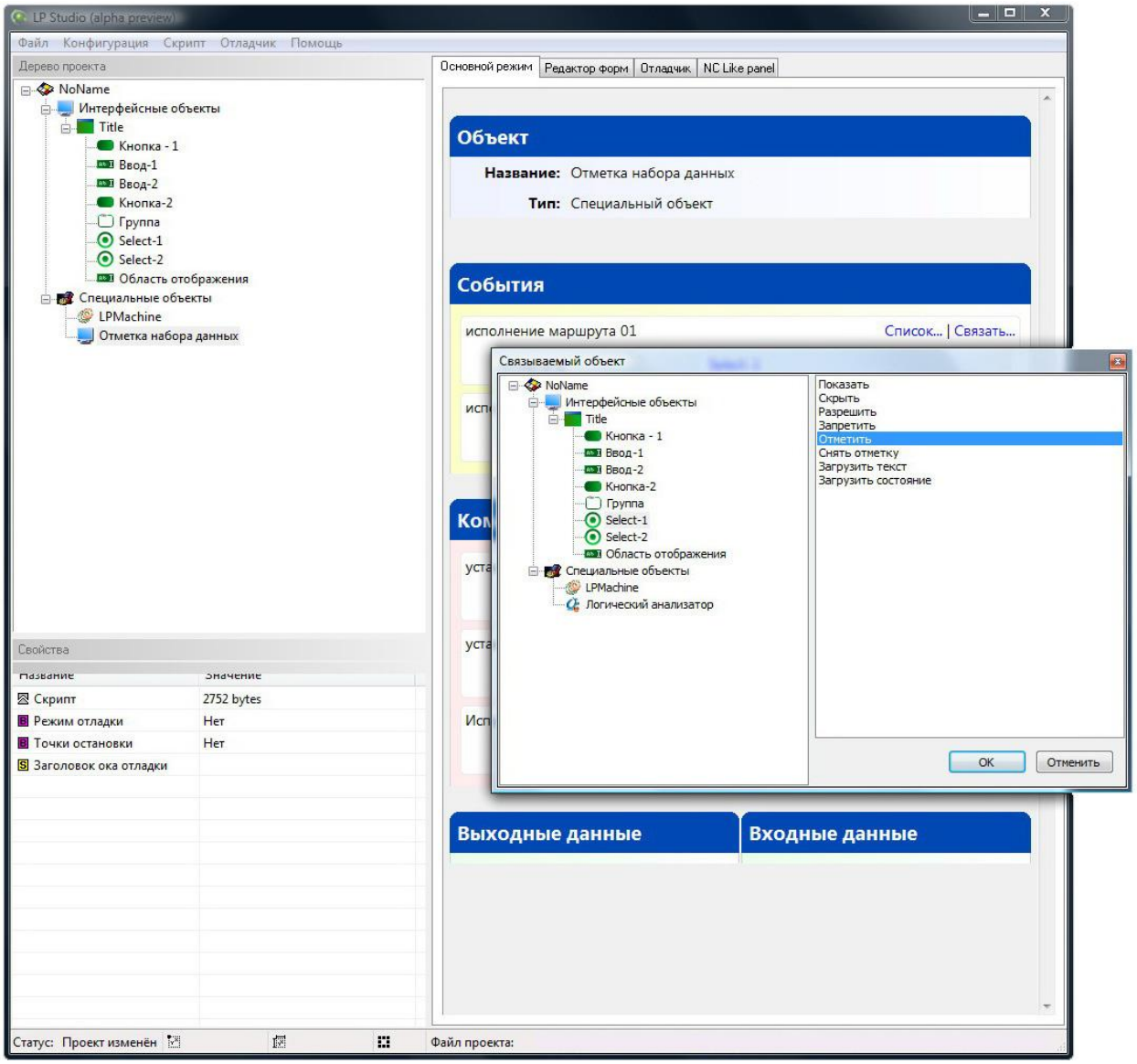


Рисунок 34 Пример проектирования связи компонентов в "Основном режиме"

Управление связями

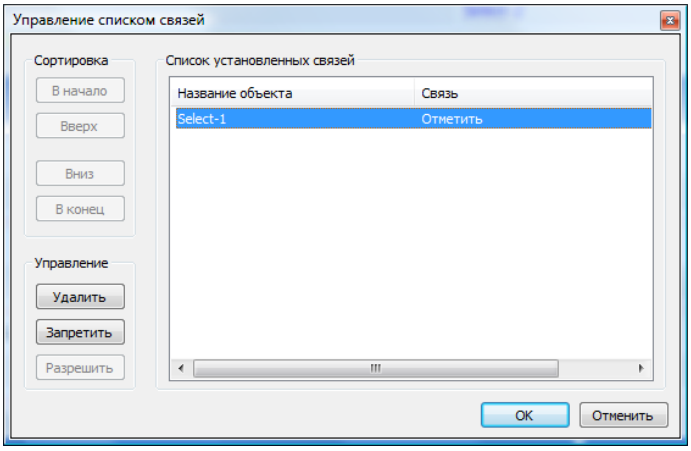


Рисунок 35 Пример диалогового окна управления списком связей

Редактор форм

«Редактор форм» предназначен для формирования структуры визуальных компонентов. Основные возможности – формирование оконного интерфейса, размещение дополнительных визуальных компонентов в нём. Каждое окно представлено узлом в дереве проектов, а дополнительные объекты, размещенные в нем, элементами данного узла. Все интерфейсные компоненты кроме объекта – окно, могут быть представлены только в рамках компонента окно. Разместить компонент «окно» в рамках другого компонента «окно» – не допускается. Однако имеется возможность организации подчиненности одного компонента окно, другому. Это достигается через организацию соответствующих отношений и конфигурирование необходимого объекта «окно».

Формирование окна и элементов в редакторе форм производиться через выбор требуемого объекта в панели компонентов.

Панель компонентов вызывается через выбор соответствующей иконки, расположенной в верхнем левом углу закладки «Редактор форм». Сама педель может быть размещена в произвольном месте закладки.

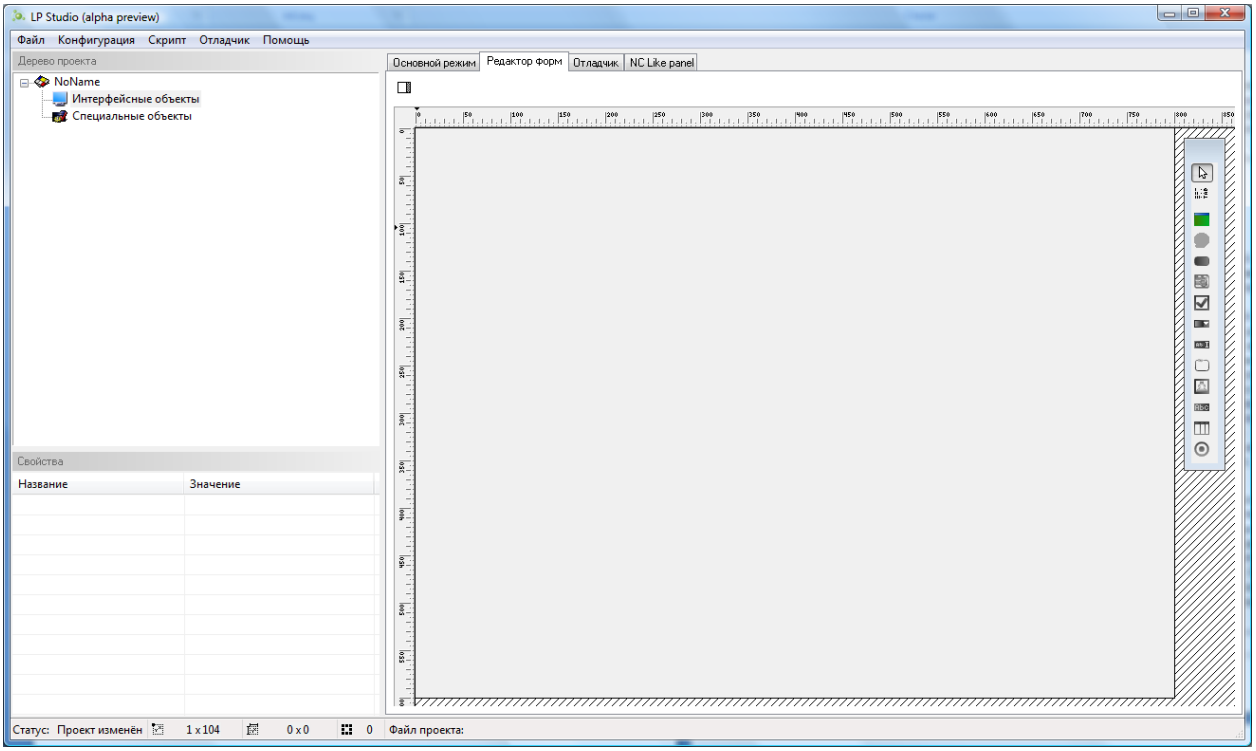


Рисунок 36 Пример формирование нового компонента Окно

Внимание! В версии LP-Studio (alpha preview) доступ к панели инструментов возможен через кнопку «панель», находящуюся в левом углу области редактора форм, даже тогда, когда кнопка не отображается.

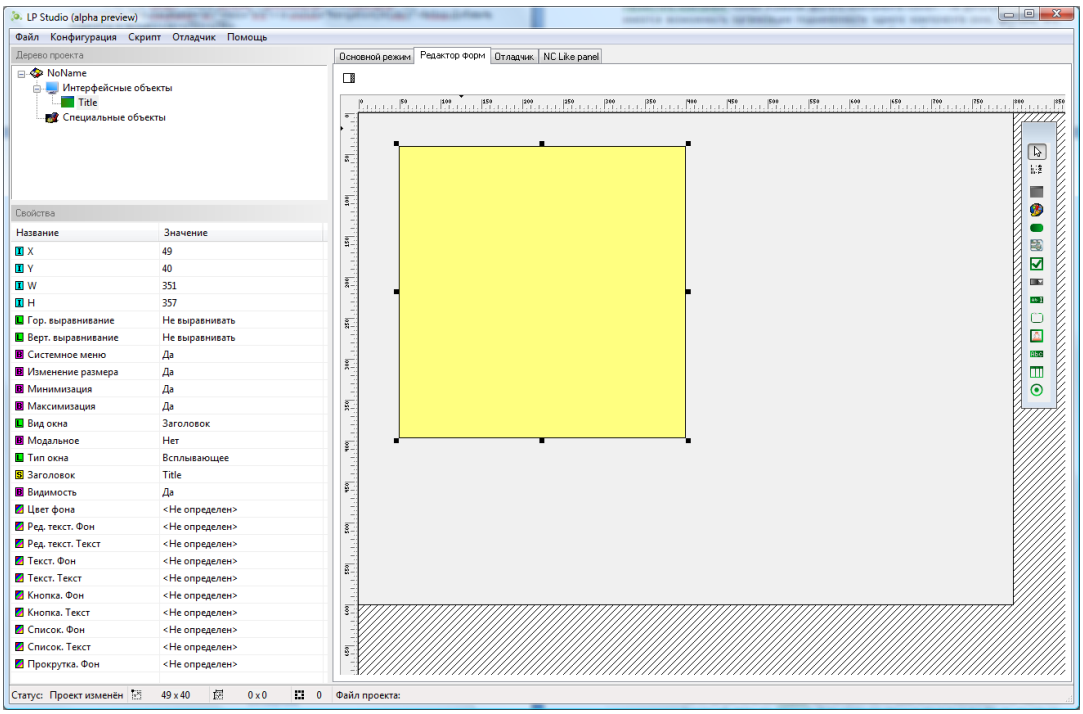


Рисунок 37 Пример определения области и размещения объекта – окно

Режим «Редактор форм» предоставляет средства по созданию, модификации размеров визуальных объектов, заданию точки их размещения и т.п. Следует помнить, что окно будет выводиться на экране монитора с учетом заданной точки вывода. Точка вывода задается координатами X,Y для верхнего левого угла объекта окна.

Настройка элементов позволяет указывать метод выравнивания, смещения и т.п. для элементов окна при изменении его размеров.

Отладка

Данный режим активируется при запуске LP-программы с указанием «Создать и запустить в отладчике».

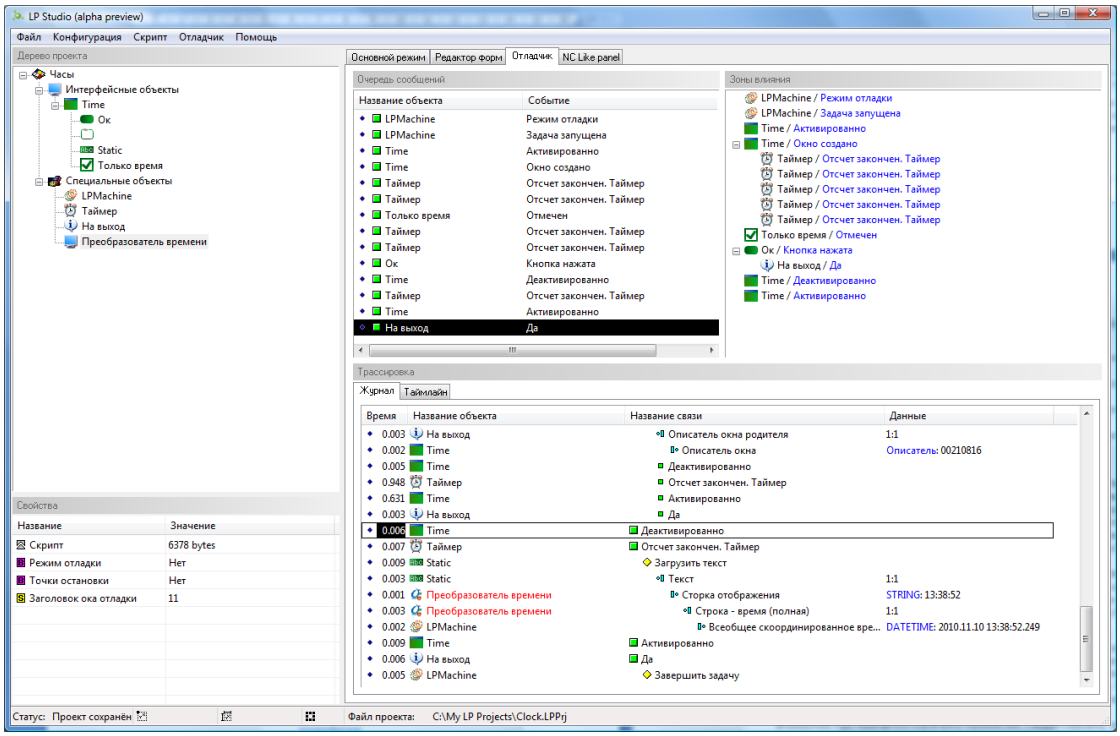


Рисунок 38 Пример работы режима "Отладчик"

Область отладчика разделена на группы:

- *Очередь сообщений* – область отражает фактическую очередь событий в системе. Следует помнить, что фактическая очередь событий может зависеть не только от последовательности действий оператора, последовательности активности внешних источников, но и от хода работы операционной системы. При разной технической конфигурации, загруженности ОС, одна и также обработка может требовать разный объем времени, что в свою очередь может приводить к разным временным характеристикам появления событий от объекта. Выбор (двойное нажатие левой кнопкой мыши) на строке в журнале «очередь событий» позволяет выбрать соответствующую запись в трассировке.
- *Зона влияний* – древовидное отображение очереди событий, имеет ограничение – отображается только один уровень отношений, при этом, фактически, может быть более глубокий уровень отношений между компонентами. Выбор (одинарное нажатие левой кнопкой мыши) на строке в журнале «зона влияния» позволяет выбрать соответствующую запись в трассировке.
- *Трассировка* – журнал, отражающий весь набор событий, реакций, передачу данных между компонентами. Журнал имеет 2^e формы представления:
    - *Журнал* – предоставление хода работы программы в виде списка. Отражает время появления строки регистрации, название объекта производящего работу, название связи, данные (*участвующие в связи*). Выбор (*двойное нажатие левой кнопкой мыши*) на строке в журнале позволяет выбрать соответствующий объект в дереве проекта. При этом выбранный в дереве проекта компонент подсвечивается красным названием в журнале.
    - *Таймлайн* – временное представление хода работы программы. Режим поддерживает 2-окна отображения одного журнала. Структура представления: по оси X – временная последовательность работы объекта, по оси Y – последовательность появления объекта в ходе работы LP-программы. При этом объект представлен только один раз.

Быстрое переключения между режимами осуществляется двойным щелчком левой кнопки мыши на строке журнала.

NC like panel

Альтернативный механизм программирования, основанный на парадигме работы с свойствами компонентов (2 одинаковые панели, между которыми происходят операции), где команды выполняются только «горячими клавишами»

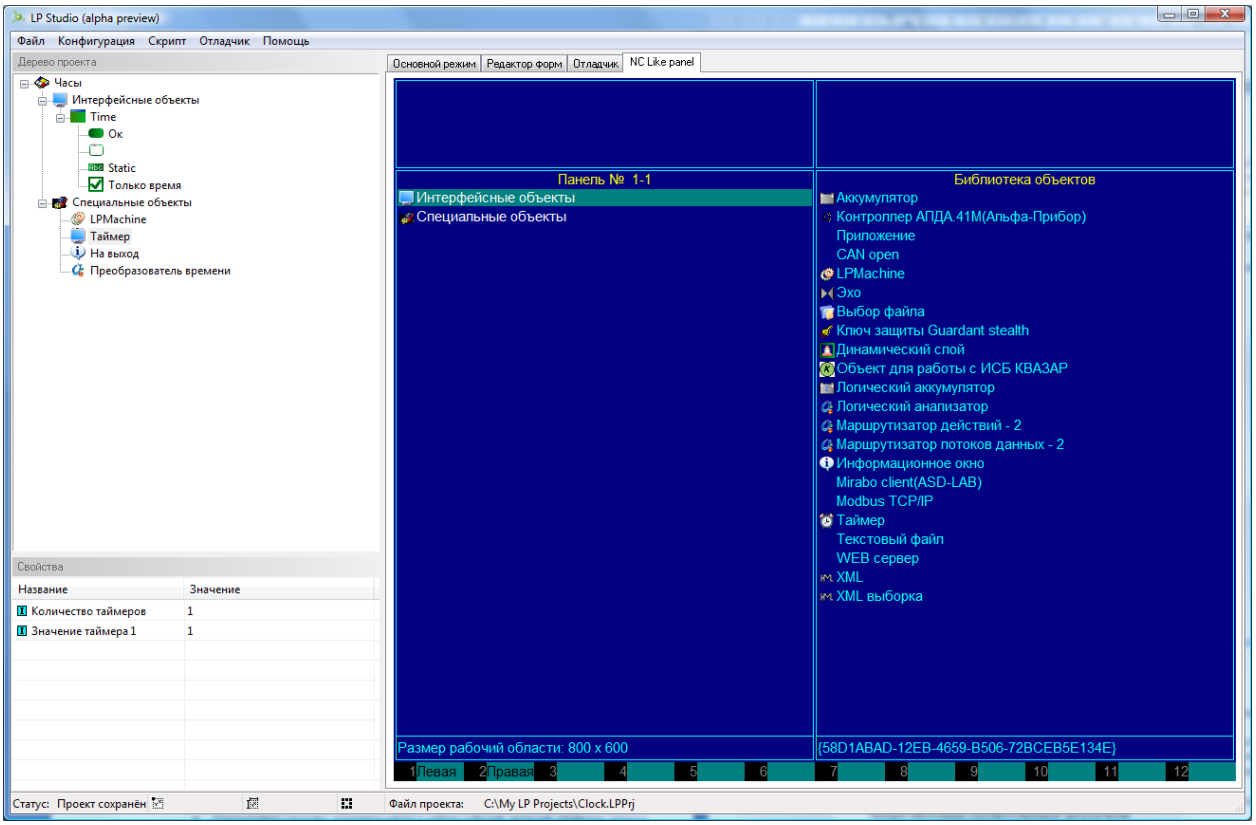
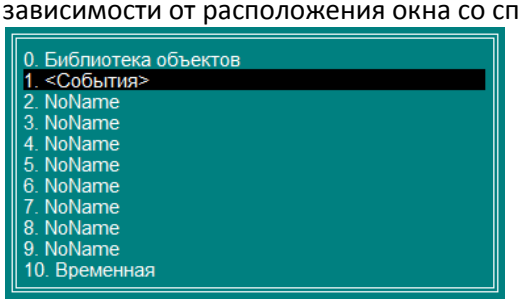


Рисунок 39 Пример работы режима "NC like panel"

Базовые режимы системы:

- Переход на библиотеку компонентов или на одну из 10 закладок - Alt+F1 / Alt+F2 (в зависимости от расположения окна со списком)



- Добавление специализированного компонента в проект – выбор закладки «Библиотека» - - Alt+F1 / Alt+F2 далее копирование компонента в дерево проекта – F5
- Удаление компонента – Del
- Переименование компонента – F2
- Создание группы (для специализированных компонентов) – F7
- Формирование отношений – F5
- Копирование компонента в группу – F6

Для работы с компонентами во «временной» закладке

- Экспорт компонентов во временной закладке во внешний файл *.LPPart – F11
- Убрать компоненты (не удаление) из временной закладки– F12

- Создать файл с графическими примитивами и связями между ними (в формате Microsoft Visio) – Alt+F11

Общий принцип программирования механизмом NC основан выборе свойств 2-х объектов и «*копировании*» необходимой связи. Например:

Панель №1	Панель №2
Интерфейсный объект **Time** **Ok** **<События>** **Кнопка нажата**	**Специальные объекты** **На выход** **<Команды>**
…	… **Создать информационное окно**

Находясь на строке «Создать информационное окно» нажмите клавишу F5. Это приведет к созданию связи: «*Кнопка нажата->Создать информационное окно*».

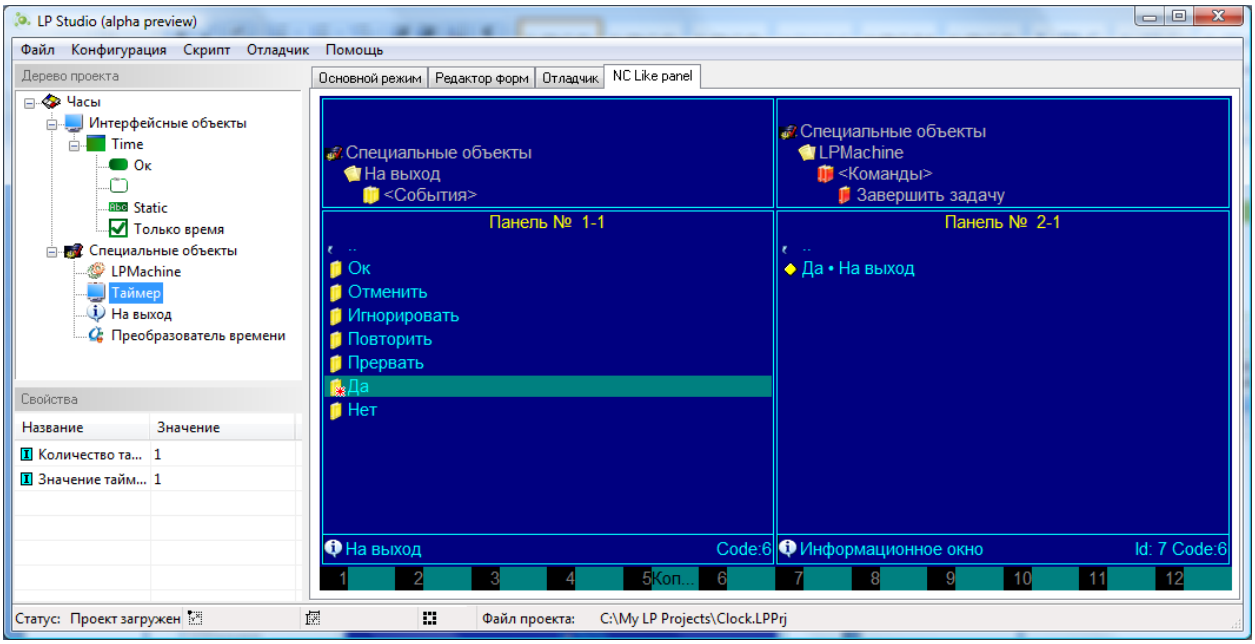


Рисунок 40 Пример организации связи между компонентами

При наличии дополнительного узла в рамках специальных объектов, имеется возможность переноса компонентов в требуемый узел, по клавише F6.

Дополнительная информация по выбранным компонентам дерева проекта отражает в нижней части панелей: идентификатор и (или) код команды, события.

В режиме «NC» проведения синхронизации позиционирования выбранного компонента в режиме «NC» и дерева проекта необходимо произвести нажатие клавиш: Shift+Enter

Каждая панель поддерживает до 8 закладок на элементы дерева проекта. При этом каждая закладка может отображать библиотеку специальных объектов, а также поддерживает единую временную папку. Временная папка позволяет размещать ссылку на необходимые компоненты в рамках единой, виртуальной группы. Что позволяет упрощать доступ к требуемым объектам в сеансе работы над проектом.

Внимание! Удаление ссылки в закладке «Временная» приведет к удалению экземпляра компонента в дереве проекта. Для извлечения ссылки из временной папки используйте клавишу – F12.

Дополнительное свойство временной папки

- экспорт содержимого в файл частичной записи проекта. Расширение файла - *.LPPart В файле размещаются все компоненты находящиеся во временной группе, с учетом их настроек и связей. В дальнейшем данный файл может быть внедрен в любой проект необходимое число раз, при этом сохранится всz группа объектов и их внутренние связи. Данный механизм позволяет многократно использовать частичные решения в новых проектах. Коллекцию примеров вы может найти на сайте http://www.logicprogram.ru
- формирование файла в формате Microsoft Visio.

Режим программирования «NC» ориентирован на работу исключительно с клавиатурой. Основное назначение режима «NC like panel» - поддержка скоростного программирования на платформе LogicProgram.

Меню LP – studio

Файл

Разделы меню:

- *Новый* – создание нового проекта (файл с расширением *.LPPrj)
- *Открыть* – загрузка проекта в LP-Studio
- *Импортировать…* – внедрение в дерево проекта группы компонентов сохраненных в экспортном файле. Формат файла *.LPPart Компоненты размещаются в специализированном разделе (узле) дерева проекта. Используя режим «NC like panel» компоненты могут быть перемещены в необходимые точки дерева проекта.
- *Сохранить* – сохранение текущего проекта
- *Сохранить как…* – сохранение текущего проекта под новым именем
- *Выход* – окончание работы в LP-Studio

Конфигурация

Разделы меню:

- *Настройки…* – конфигурирование настроек работы LP-Studio

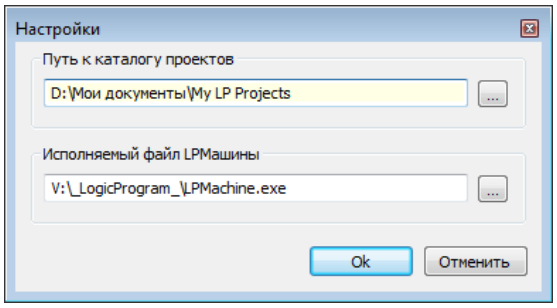


Рисунок 41 Пример диалога настроек работы LP-Studio

Свойства настроек:

- *Путь к каталогу проектов* – путь размещения проектов. При этом если у проектов заданы собственные директории, то они будут располагаться внутри заданного каталога проектов.
- *Исполняемый файл LP-машины* – путь к расположению исполняемого файла LP-машины.

Скрипт

Разделы меню:

- *Создать* –создание исполняемого файла-скрипта проекта (расширение *.LPScp)
- *Создать и запустить* – создание и одновременный запуск файла-скрипта

- *Создать и запустить в отладчике* – создание и одновременного запуска файла-скрипта в режиме отладки. Активным режимом работы LP-Studio становиться - «Отладка»
- *Прервать* – команда, позволяющая прервать работу файла-скрипта.
  Внимание! Данная команда не только останавливает работу скрипта проекта, но и завершает сессию работы LP-машины. Наличие доступа к данной команде указывает, что сессия LP-машины не была завершена. Одна из причин, закрытие окна взаимодействия с пользователем. Без передачи команды специальному компоненту – LP-Машина: завершить работу.
- *Пауза* – команда, позволяющая приостановить работу скрипта проекта. Не влияет на работу внешних объектов, что в свою очередь может повлиять на частичное продолжение работы скрипта

Отладчик

Разделы меню:

- *Очистить* – команда очистки содержимого журнала работы скрипта проекта
- *Удалить пустые записи* – команда, производящая удаление записей в журнале трассировки которые не участвовали в цепочках обработки
- *Разрешить трассировку* – команда, указывающая, что запуск проекта в отладчике будет формировать записи в журнале трассировки
- *Абсолютное время* – флаг переключения отражения типа времени в строке журнала трассировки.
  Допустимые значения:
  - *Абсолютное время* – время, прошедшее между предыдущей и текущей записью. Отражает фактически затраченное время на работу компонента
  - *Фактическое время* – время, отражающее фактическое завершение работы компонента
- *Управление трассировкой* – флаг разрешения/запрета управления журналом трассировки с помощью команд компонента LP-Машина.

Помощь

Сведения о проекте, авторские права и т.п.

Справочник «горячие клавиши» LP-Studio

LP-Studio

Действие	Клавиша
Файл.Сохранить	**Ctrl+S**
Выход	**Alt+F4**
Скрипт.Создать	**Ctrl+M**
Скрипт.Создать и запустить	**Ctrl+R**
Скрипт.Создать и запустить в отладчике	**Ctrl+D**
Скрипт.Прервать	**Ctrl+T**
Скрипт.Пуза	**Ctrl+P**

Дерево проектов

Действие	Клавиша
Переименовать компонент	**F2**
Удалить компонент	**Del**

NC like Panel

Действие	Клавиша
Переименовать компонент	**F2**
Удалить компонент	**Del**

Создание нового компонента

Специализированный компонент «Логический анализатор» предназначенный для создания новых компонентов платформы предоставляет дополнительную графическую среду автоматизированного проектирования. Доступ к данной среде проектирования осуществляется через выбор значения – размер скрипта в конфигурировании экземпляра компонента в дереве проекта.

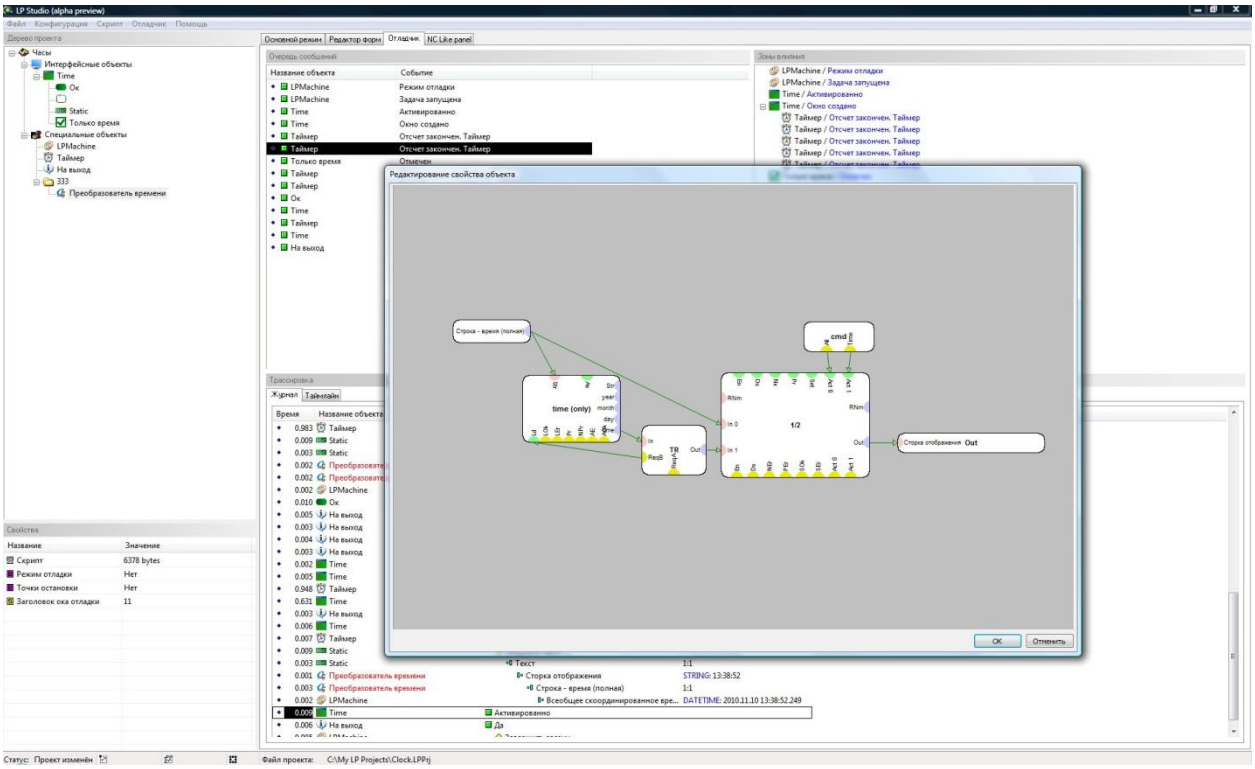


Рисунок 42 Пример вызова диалога проектирования нового компонента

Среда проектирования

Среда проектирования представляет окно проекта компонента, в рамках которого располагаются субкомпоненты и отражаются установленные связи. Доступ к компонентам осуществляется через контекстное меню, вызываемого щелчком правой кнопки мышки в рамках окна проектирования.

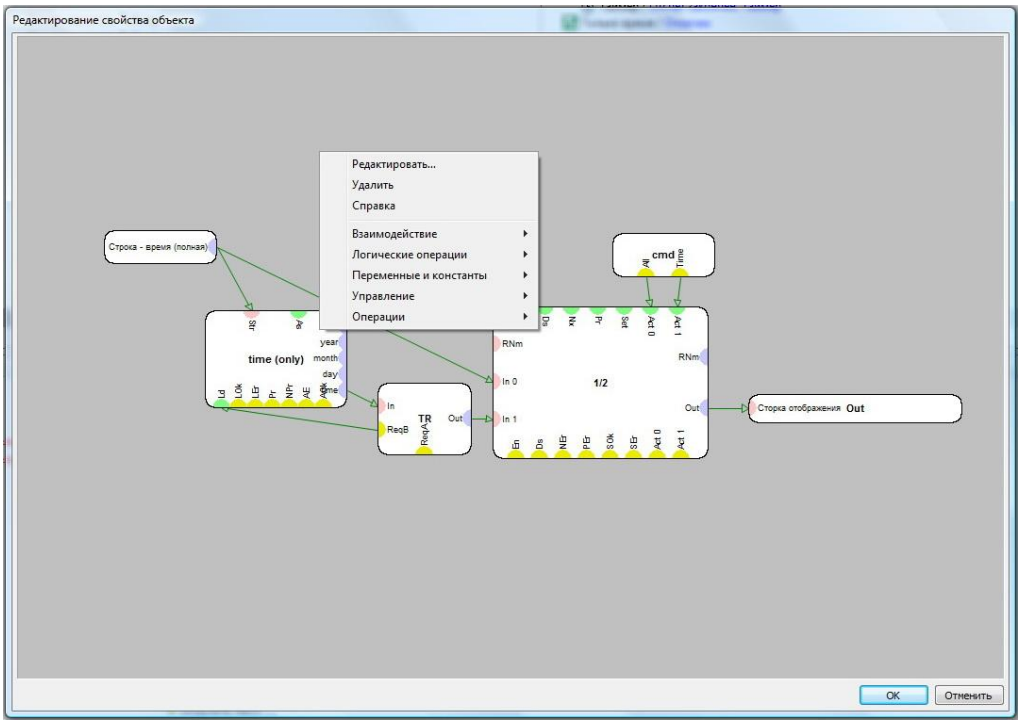


Рисунок 43 Пример вызова меню свойств системы проектирования

Общие понятия

Основой проектирования нового компонента являются субкомпоненты, представленные в разных классах. В зависимости от типа субкомпонента он может обладать разным набором свойств. Некоторые компоненты позволяют формировать новую структуру свойств, однако общая структура свойств субкомпонента неизменна.

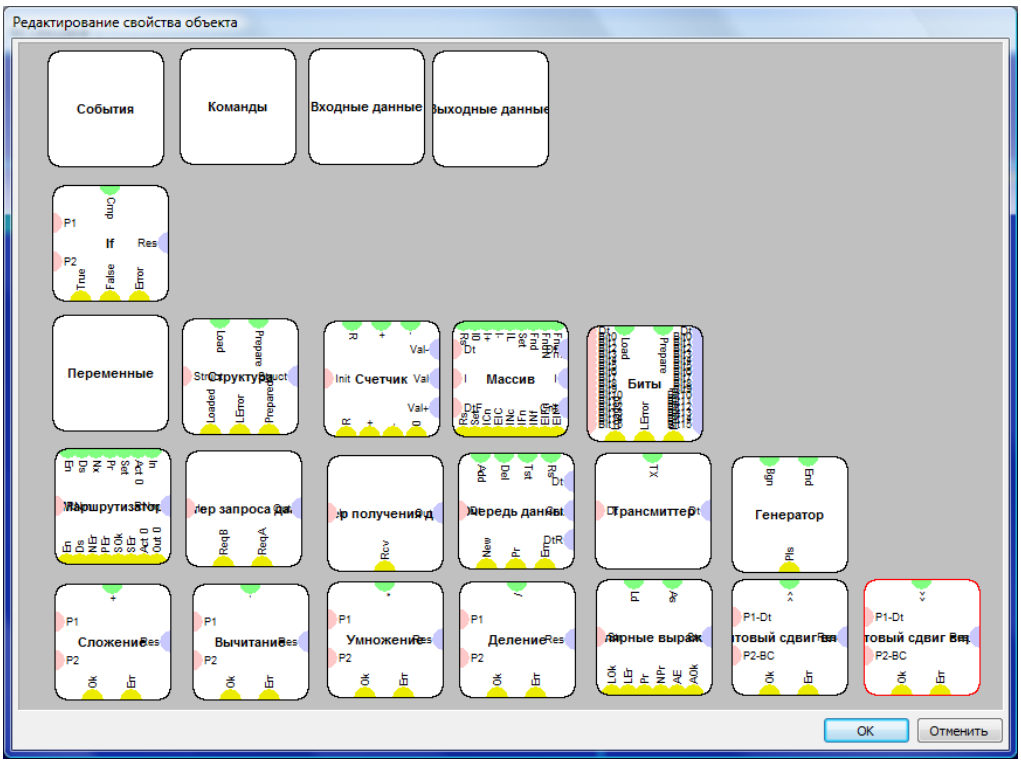


Рисунок 44 Пример отражения субкомпонентов

Горячие клавиши управления окном «Редактирования свойств объекта»:

- Закрытие окна – ESC (**Внимание!** Без сохранения последних изменений.) Действие равноценное действию кнопки – «Отменить».

Общие принципы управления окном «Редактирование свойств объекта»:

- Изменение размера окна. При повторном открытии окна будет использовать последний заданный размер;
- Вывод окна осуществляется в месте последнего расположения;
- Выбор объекта – одинарное нажатие левой кнопкой мыши;
    - Действия с выбранным объектом:
        - Изменение размера – «захватите» манипулятором мышь за край области субкомпонента и потяните в строну;
        - Вызов «контекстного меню» в области выбранного объекта – предоставляет доступ к элементам меню:
            - *Редактировать* – вызов окна «свойства объекта»;
            - *Удалить* – удаление объекта из проекта компонента;
            - *Справка* – вызов справочной информации по типу экземпляра субкомпонента.
- Вызов окна «Свойства объекта» - двойное нажатие левой кнопкой мыши на объекте;
- Выбор несколько объектов – установите курсор мыши вне области объектов, нажмите и удерживаете левую кнопку манипулятора, произведите выбор группы объектов
    - Выбранную группу можно перемещать в области проекта компонента.

Редактирование свойств выбранного субкомпонента производиться в рамках окна «Свойства объекта». Вызов окна – двойное нажатие левой клавишей мыши.

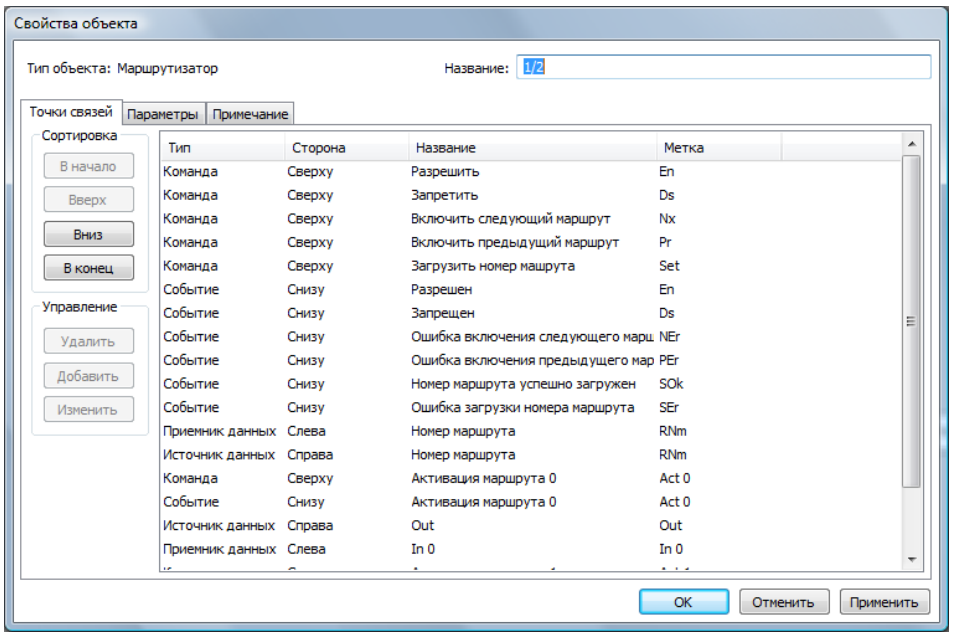


Рисунок 45 Пример диалога настройки свойств субкомпонента

Структура свойств компонента:

- *Точки связей* – закладка предназначена для управления точками обеспечивающими взаимодействие других субкомпонентов с редактируемым объектом.
  Группа «*Сортировка*» - позволяет перемещать выбранную строку в закладке.
  Группа «*Управление*» - позволяет модифицировать структуру строк в закладке.
  Вызов «контекстного меню» на строке закладки, позволяет менять сторону ориентации представления строки в графическом отображении объекта.
- *Параметры* – закладка предназначена для конфигурирования свойств субкомпонента, установки режима его работы и т.п.
- *Примечание* – закладка предоставляющая область ввода произвольного текста, назначение которого, хранение дополнительной, поясняющей информации для программиста.
- *Название* – отражение названия субкомпонента, по умолчанию каждому экземпляру субкомпонента присваивается системное наименование. Для удобства работы данное название можно модифицировать.

Горячие клавиши управления окном «Свойства объекта»:

- Закрытие окна – ESC (**Внимание!** Без сохранения последних изменений.) Действие равноценное действию кнопки – «Отменить».

Общие принципы управления окном «Свойства объекта»:

- Изменение размера окна не допускается;
- Вывод всегда производиться на главный монитор.

Методика представления субкомпонентов разработки

Все субкомпоненты графической среды автоматизированного проектирования представлены в виде квадрата, каждая сторона которого отражает один из четырех интерфейсов взаимодействия с субкомпонентом. Часть объектов имеет возможность изменения набора элементов в интерфейсах взаимодействия. При этом каждый тип интерфейса имеет собственный цвет. Программист может менять строну расположения элемента интерфейса.

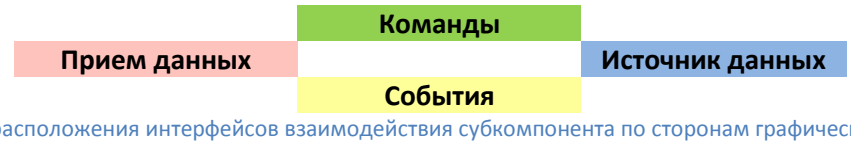


Схема расположения интерфейсов взаимодействия субкомпонента по сторонам графического представления

В каждом типе интерфейса может быть различное количество элементов. Однако каждый из них имеет соответствующую своему типу – цветовую идентификацию. При изменении стороны расположения, цветовая идентификация не изменяется.

Структура субкомпонентов

Субкомпоненты графической автоматизированной среды проектирования разделены на следующие классы:

- **Взаимодействие** – группа субкомпонентов отвечающих за организацию взаимодействия разрабатываемого компонента с LP-программой. Именно наличие данных элементов позволяет организовывать необходимый набор команд и связей по данным с новым объектом программы.
- **Логические операции** – субкомпоненты, предоставляющие операции сравнения с предоставлением соответствующих результатов.
- **Переменные и константы** – субкомпоненты предоставляющие инструменты по организации работы с переменными, массивами и т.п.
- **Управление** – субкомпоненты предоставляющие инструменты по организации хода исполнения проектируемой логики компонента.
- **Операции** – субкомпоненты, предоставляющие инструменты для проведения математических операций, обработки текстовых значений и т.п.

Правила проектирования отношений в Логическом анализаторе

Проектирование структуры компонента в «Логическом анализаторе» основано на действии установки связи между двумя субкомпонентами. Связь от одного элемента объекта может идти к одному и более элементам других компонентов.

Предположим, что необходимо создать компонент, который будет предоставлять следующий сервис:

- Нажатие кнопки пользователем должно обеспечить сохранение информации о дате и времени данного события.

Формируем базовую структуру нового компонента:

Название субкомпонента	тип элемента	название элемента
Событие	событие	нажатие кнопки
Выходные данные	источник данных	время
Входные данные	преемник данных	Текущее время
Переменные	источник данных	Time
	преемник данных	Time
	команда	Time

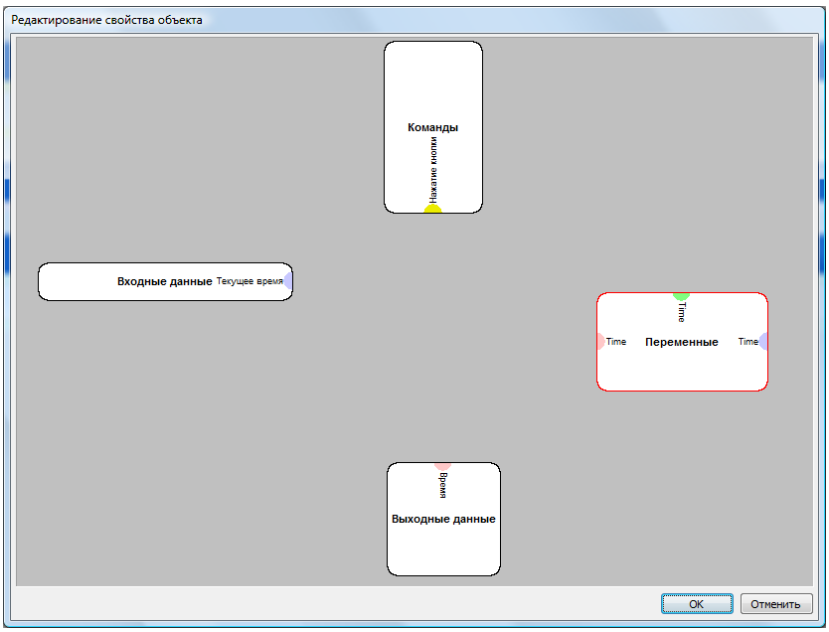


Рисунок 46 Пример формирования связей, часть 01

Формируем связи, от субкомпонента «Команды» к субкомпоненту «Переменные»: «*С:Нажатие кнопки →Time*». Формирование связи производится выбором точки одного объекта и протягиванием линии связи к другому компоненту. Система проектирования производит контроль допустимости установки отношений между типами элементов субкомпонентов, с тем, чтобы исключить связи типа «Событие:Команда».

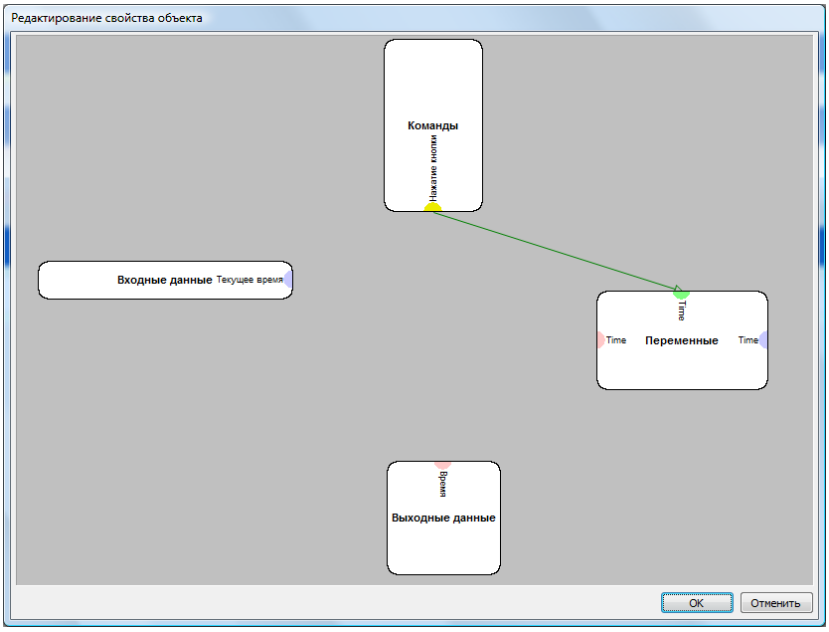


Рисунок 47 Пример формирования связей, часть 02

Внешнее представление данного компонента выглядит следующим образом:

Тип интерфейса	Элемент
События	
Команды	Нажатие кнопки
Выходные данные	Время
Входные данные	Текущее время

Полная структура компонента выполняет работу по следующему алгоритму:

Нажатие кнопки в окне интерфейса пользователя, активирует компонент командой «Нажатие кнопки», что в свою очередь приводит к сохранению текущего времени в переменной компонента. При запросе у компонента сохраненного значения, это значение предоставляется из области хранения заданной переменной Time.

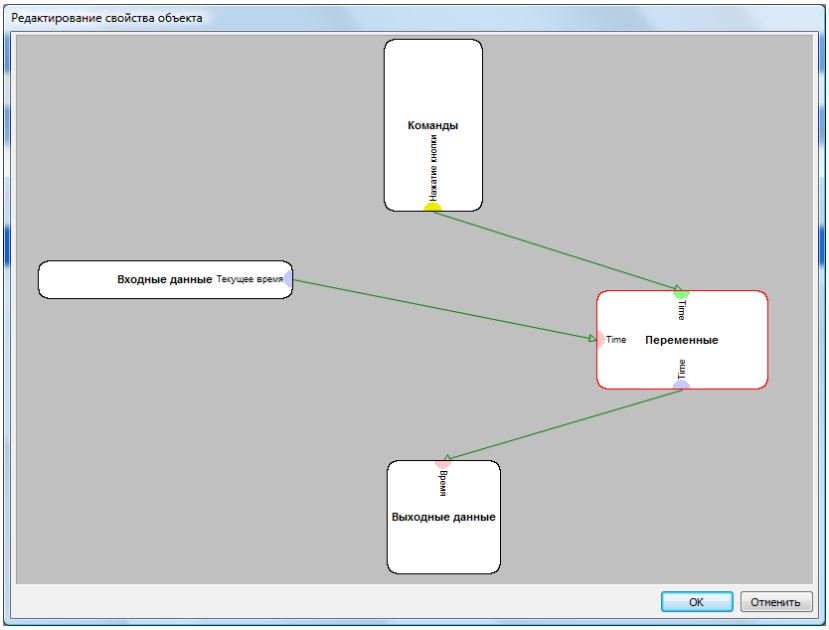


Рисунок 48 Пример формирования связей, часть 03

При этом элемент субкомпонента «Переменные»- это выходные данные:Time

Если необходимо после сохранения переменной, получить событие от компонента:«ОК», что будет означать что работа компонента завершена, то следует доработать проект как это показано ниже:

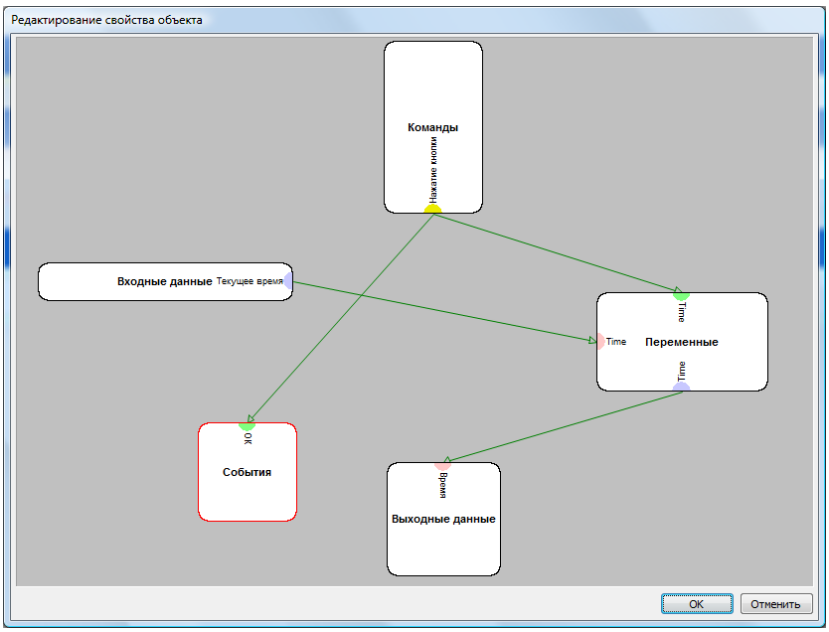


Рисунок 49 Пример формирования связей, часть 04

Внимание! Последовательность связей от элемента субкомпонента, определяет очередь исполнения.

В данном примере сначала отработает цепочка:

1. *С:Нажатие кнопки->Сохраннее значение в переменной*
2. *С:Нажатие кнопки->отправка события ОК*

Общие правила управления связями

Доступные действия с установленной связью:

- *Удаление* – выберите стрелку связи, произведите удаление через контекстное меню;
- *Установка точки остановки* – двойным нажатием на выбранной стрелке вызовите диалоговое окно, в котором введите название точки остановки. Связь, имеющая точку остановки, имеет большую стрелку. Точка остановки позволяет определить место прерывания работы компонента, до момента продолжения работы в процессе отладки. Повторный выбор стрелки, сбрасывает «точку остановки».
- *Перенос связи* – если связь необходимо перенести на другой элемент, то выберите вершину стрелки связи и, удерживая её, перенесите на другой элемент.

  Внимание! Перенос связи изменяет последовательность связей в точке объединения.

Принцип формирования логики работы компонента соответствует общей парадигме разработки структуры LP-программы.

Отладка работы компонента «Логический анализатор»

LP-Studio предоставляет механизм отладки логики компонента «Логический анализатор». Для активизации отладки необходимо установить флаг отладки компонента: «Режим отладки», а также флаг обработки «Точек остановки».

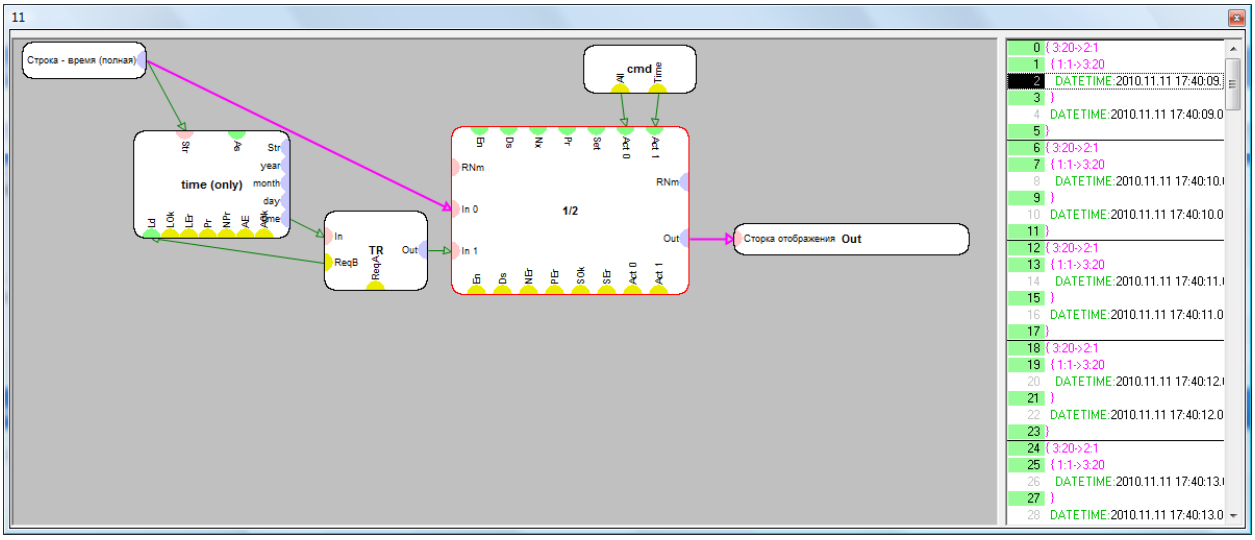


Рисунок 50 Пример работы отладки проекта компонента

Окно отладки работы компонента Логический анализатор представлено в виде:

- *Область проекта* – отражает графическое представление компонента, предоставляет возможность выбора субкомпонентов, связей, активизации окна «Инспектор» субкомпонента, отражающего внутреннее состояние объекта;
- *Область журнала* – отражает события, произошедшие в рамках компонента.

Окно отладчика позволяет выбирать объекты контроля и получить выбор событий объекта в области журнала регистрации событий субкомпонентов. Выбор компонентов позволяет получить доступ к специализированному диалогу «Инспектор» текущего состояния субкомпонента. Данный режим предоставляется не для всех объектов автоматизированной среды проектирования.

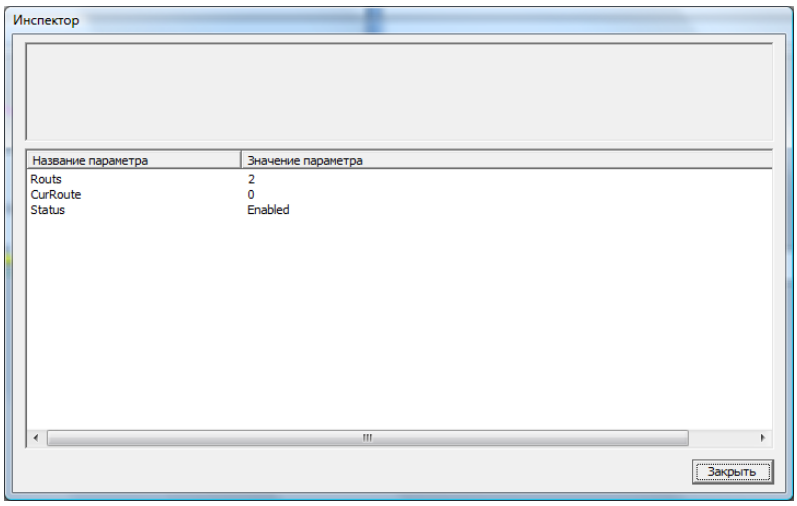


Рисунок 51 Пример диалога "Инспектора"

Режим отладки разрабатываемых компонентов представляет необходимые средства для проведения анализа хода работ, фактического потока данных и его обработки, проведения анализа последовательности исполнения цепочек связей, содержимого переменных, массивов, состояния маршрутизаторов и т.п.

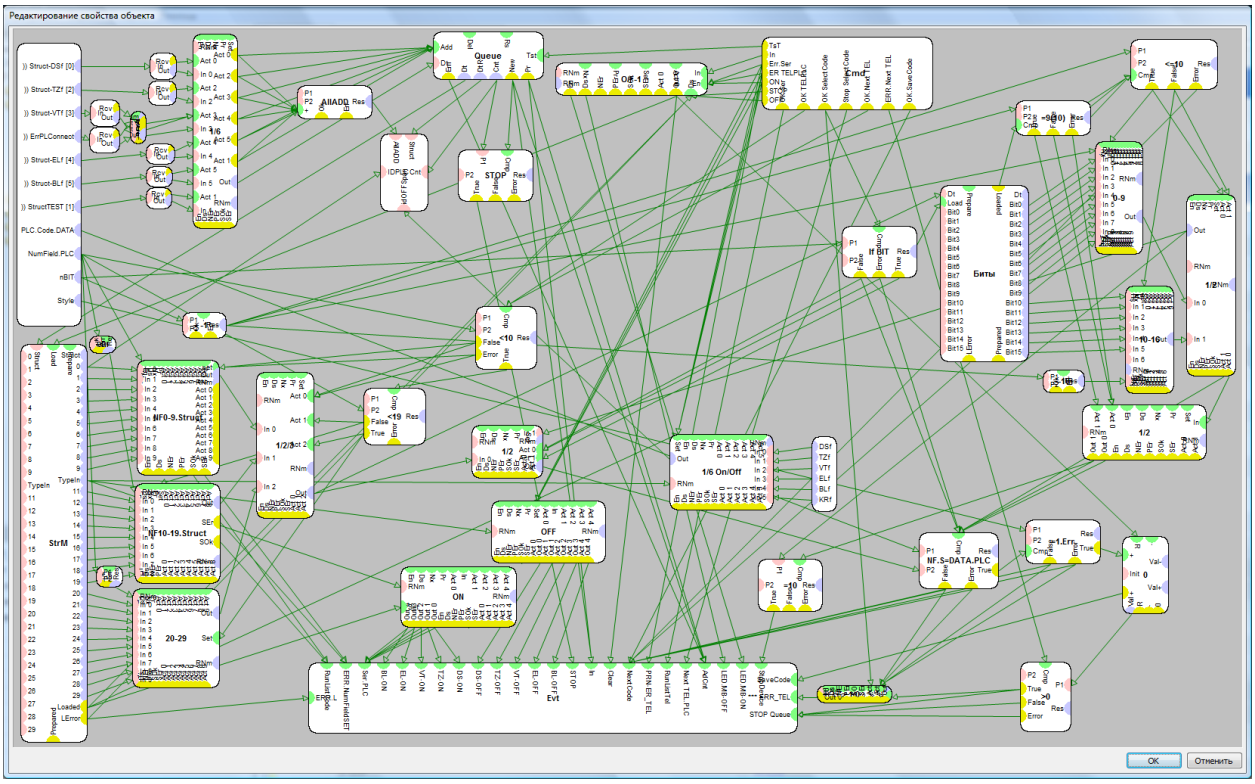


Рисунок 52 пример структуры компонентов

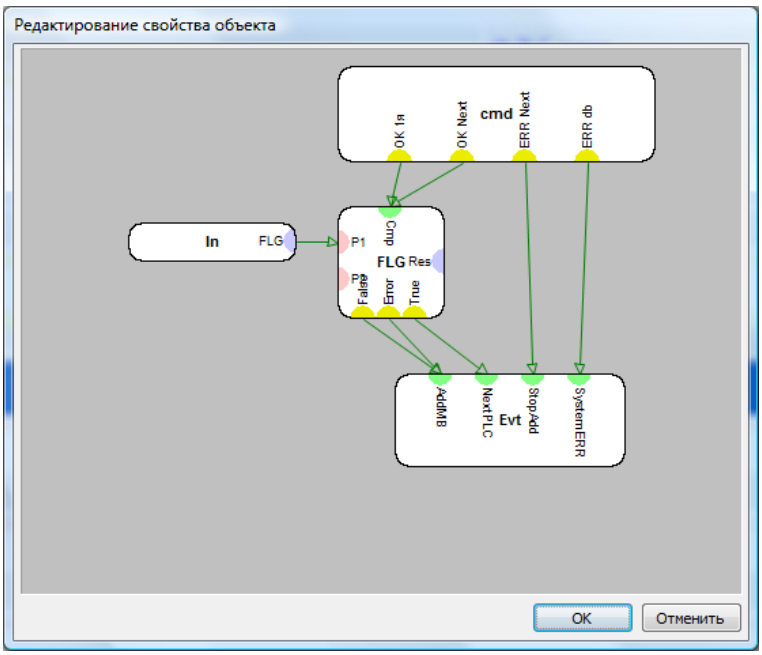


Рисунок 53 пример структуры компонентов

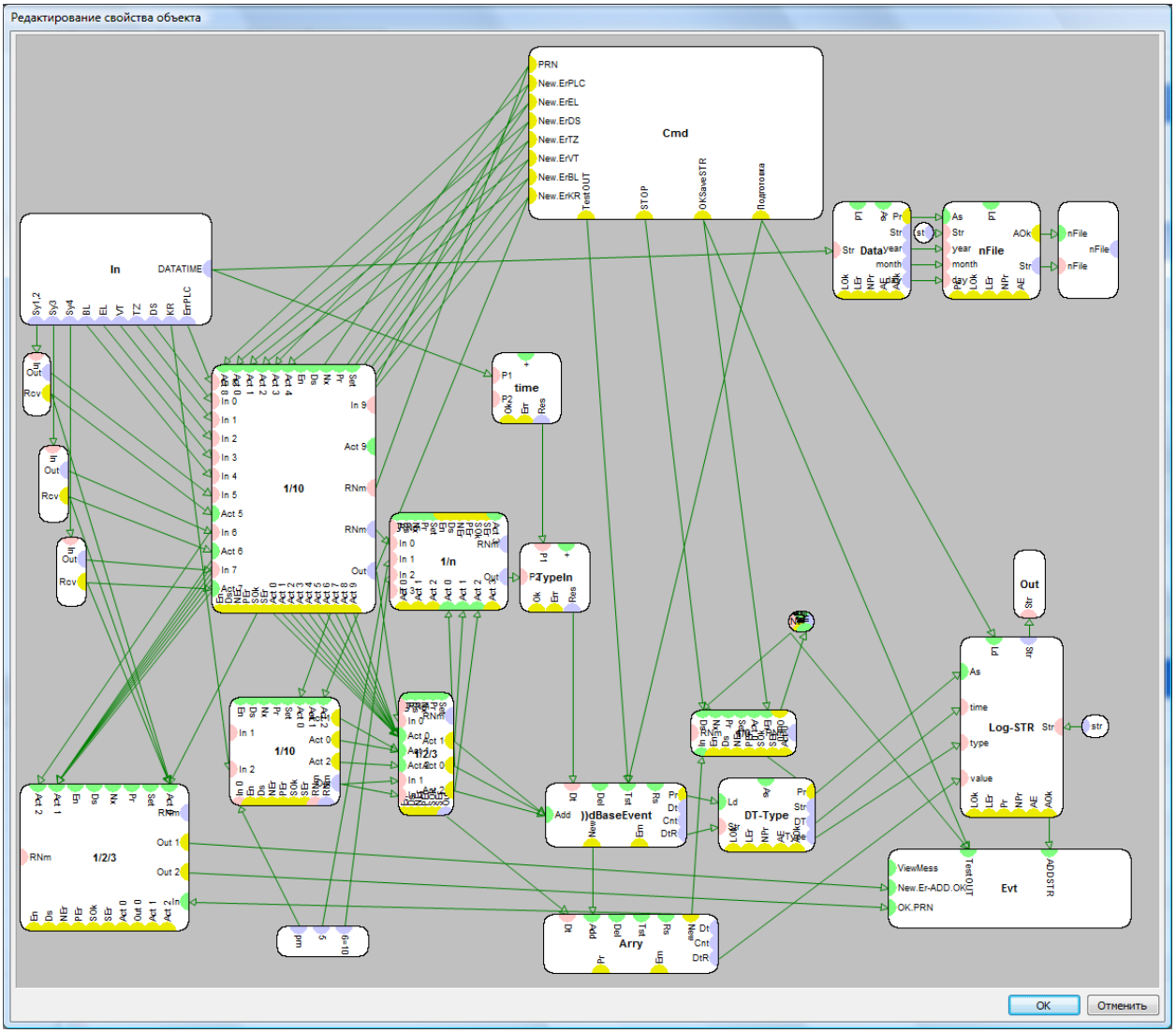


Рисунок 54 пример структуры компонентов

Механизм расширения субкомпонентов

Платформа LogicProgram представляет API, позволяющее пополнять структуру «Логического анализатора» новыми субкомпонентами. За получением технических подробностей по application programming interface к «Логическому анализатору» – обращайтесь на сайт платформы – http://www.logicprogram.ru

Справочник по компонентам платформы LogicProgram

Интерфейсные компоненты

Все визуальны компоненты оформляются в рамках режима «Редактор форм» программы LP-Studio, путем выбора необходимого объекта, задание ему необходимых размеров и расположения на экране, окне.

Все интерфейсные объекты могут быть распложены только в рамках окна отображения, само окно может быть представлено единожды для конкретного экземпляра компонента – окно. При этом компонент окно отражается в дереве проекта как узел, в рамках которого отражаются дополнительные интерфейсные компоненты. Количество экземпляров окон в проекте неограниченно.

Окно

Предназначено для предоставления окна взаимодействия с пользователем.

Browser

Компонент предназначен для организации интерфейса взаимодействия через HTML страницы. Поддерживается передача ссылки на страницу шаблон, формирование и отображение динамических страниц, с учетом каскадной таблицы (CSS), исполнения JavaScript.

Кнопка

Компонент предназначен для организации элемента управления работой программы.

Таблица

Компонент предназначен для организации списочного вывода данных.

Флаговая кнопка (Checkbox)

Компонент предназначен для организации графического пользовательского интерфейса, который разрешает пользователю производить выбор из предопределенного набора значений. При этом выбор может быть осуществлен как для всех элементов так и не для одного из них.

Выпадающий список

Компонент предназначен для организации графического пользовательского интерфейса, предназначенного для размещения в нём нескольких элементов выбора.

Редактируемый текст

Компонент предназначен для организации ввода данных пользователем.

Группа

Компонент предназначен для графического оформления элементов окна.

Изображение

Компонент предназначен для организации вывода графических файлов. Компонент поддерживает вывод по слоям. Также возможно получить координаты X,Y при нажатии кнопок манипулятора мышь.

Не редактируемый текст (Static)

Компонент предназначен для отображения текстовой информации. Компонент позволяет отображать как статическую, так и динамическую информацию.

Список

Компонент предназначен для организации списочного вывода информации в заданной области. Список может иметь от одной до нескольких колонок заданной ширины (в пикселях).

Радио кнопка (Radio)

Компонент предназначен для организации графического пользовательского интерфейса, который разрешает пользователю производить выбор из предопределенного набора значений. При этом может быть выбрано только одно значение из группы. Номер группы задается как число.

Внимание! Для создания независимых групп выбора в рамках одного окна, необходимо в конфигурации экземпляра компонента – радио кнопка, задать отличные друг от друга номера

групп. Совпадение номеров групп у компонентов расположенных в разных окнах не влияет на их совместную работу.

Специальные компоненты

Специализированные компоненты обеспечивают необходимой набор средств по обработке информации в проекте. При этом в рамках работы специализированных компонентов могут быть созданы специализированные, визуальные пользовательские диалоги позволяющие оператору определить ход работы компонента. Специализированные компоненты могут осуществлять свою работу и без взаимодействия с пользователем.

Аккумулятор

Компонент предназначен для организации процесса «счетчик», при этом диапазон работы может быть как в плюс, так и в минус. Имеется возможность задавать начальное значение. **Внимание!** Компонент работает только с целочисленными значениями.

Контроллер АПДА.41М(Альфа-Прибор)

Компонент предназначен для организации взаимодействия с контроллером АПДА.41М фирмы «Альфа-Прибор».

Приложение

Позволяет организовать запуск LP-программ, с учетом корня работы программы запускающей другое приложение. Также можно запустить и программу операционной среды (*.exe), путь доступа определяется правилами доступа к программе операционной среды, с учетом текущего рабочего каталога LP-программы.

CAN open

Компонент предназначен для организации работы по протоколу CAN с учетом описания того, каким образом CAL использовать определенные функции. CAN/Open раскрывается в двух документах: первый, базовый, DS301 описывает общую структуру единицы оборудования, а второй, дополнительный, DS40x описывает определенный тип функциональных возможностей (например, для удаленного ввода/вывода, управления движением и т.д.). Совместно эти документы дают представление о том, каким образом CAL должен использоваться для реализации конкретного набора функций.

LPMachine

Компонент предназначен для организации взаимодействия LP-программы с LP-машиной. Данный компонент должен быть включен в проект **ОБЯЗАТЕЛЬНО**. При этом его наличие в проекте должно быть в единственном экземпляре.

Первое событие в LP-программе формируется именно этим компонентом: «Задача запущена», при этом команда компонента: «Завершить задачу» приведет к выгрузке экземпляра LP-машины из памяти, и как следствие полную остановку работы LP-программы.

Внимание! Если команда «Завершить задачу» не была активирована, то несмотря на отсутствие активных окон взаимодействия с пользователем, LP-программа будет в памяти компьютера и продолжит свою работу по последнему состоянию.

Эхо

Компонент предназначен для удобства организации общей логики работы проектируемой программы. Использование данного компонента позволяет оформлять активизацию набора команд для разных точек проекта. Вместо того чтобы поддерживать одинаковой набор действий для каждой из таких точек проекта.

Выбор файла

Компонент предназначен для организации специализированного диалога с пользователем – выбор файла.

Ключ защиты Guardant stealth

Компонент предназначен для организации взаимодействия с электронным ключом защиты Guardant stealth. Использование данного ключа позволяет организовывать необходимый уровень защищенности работы LP-программы, шифрование данных и т.п.

Динамический слой

Компонент предназначен для расширения функциональных возможностей визуального компонента «Изображение». Основное назначение – предоставление возможностей загрузки в визуальный компонент «Изображение» динамических файловых картинок, управления отображением: изменения координат, удаления изображения, слоя и т.п. Идентификация компонента «Динамический слой» по отношению к «Изображению», осуществляется через структуру данных компонента – «Описатель визуального объекта» и соответствие коду динамического слоя.

Объект для работы с ИСБ КВАЗАР

Компонент предназначен для работы с контроллером «Квазар» производства ОАО «Приборный «Завод Тензор». Поддерживается серия контроллеров – Квазар:4/8/16

Логический аккумулятор

Компонент предназначен для организации хранения состояния: TRUE/FALSE

Логический анализатор

Компонент предназначен для организации работы нового компонента, созданного средствами графического автоматизированного проектирования. Среда разработки вызывается путем активизации значения «Скрипт» в свойствах экземпляра компонента в дереве проекта. При наличии скрипта компонента, в ячейке – свойства будет отображен размер скрипта в байтах.

Наличие данного компонента значительно расширяет возможности языка программирования LogicProgram, за счет формирования специализированных (не штатных) компонентов платформы LP.

Маршрутизатор действий – 2

Компонент предназначен для управления разным набором действия из компонента (маршрут) при обращении к нему. Данный компонент создан средствами компонента «Логический анализатор» и может быть модифицирован пользователем.

Маршрутизатор потоков данных – 2

Компонент предназначен для управления разным набором данных из компонента (маршрут) при обращении к нему. Данный компонент создан средствами компонента «Логический анализатор» и может быть модифицирован пользователем.

Информационное окно

Компонент предназначен для организации информационного окна, структура которого задается конфигурацией свойств экземпляра компонента в дереве проекта. Для прекращения доступа к окну программы, в случае пока информационное окно активно, необходимо указать «описатель окна родителя».

Modbus TCP/IP

Компонент предназначен для организации взаимодействия по протоколу Modbus TCP/IP. Modbus - коммуникационный протокол, основанный на клиент-серверной архитектуре. Разработан фирмой Modicon для использования в контроллерах с программируемой логикой (PLC). Стал стандартом в промышленности и широко применяется для организации связи промышленного электронного оборудования. Компонент работает только в сетях TCP/IP. Задание структуры пакета взаимодействия оформляется через активизацию диалога в свойствах экземпляра компонента: значение структуры данных. Пример структуры:

```
<StructSet>
    <Struct Name="Дренаж" Code="3">
        <Address Reg="5391" Func="3" Len="1"></Address>
    </Struct>
</StructSet>
```

Таймер

Компонент предназначен для организации режима отсчета по заданному временному интервалу. Наименьшее значение 01 секунда. Поддерживается несколько таймеров в одном экземпляре компонента. Отсчет может быть организован как непрерывно, так и единожды.

Текстовый файл

Компонент предназначен для организации работы с простым текстовым файлом. Основное назначение упрощенный процесс записи в файл. Также обеспечивается чтение файла. Разделение по строкам основано на знаке перевод строки или достижении конца файла. Имеется возможность задавать позицию в файле, относительно которой будут производиться операции. По умолчанию позиция в файле: «конец файла».

WEB сервер

Компонент предназначен для организации на базе LP-программы службы web-сервера. В рамках конфигурирования экземпляра компонента в дереве проекта, возможно, задавать порт обращения, а также название корневого каталога для web-сервера.

Внимание! Следует помнить, что корневой каталог может быть расположен только в области видимости LP-программы.

Организация получения данных от клиента и предоставление данных клиенту через HTML страницы осуществляется через каналы данных компонента: входные/выходные данные.

Структура web-сервера, расположенного в заданной директории состоит из необходимого набора файлов и каталогов. Страницы сервера имеют разрешение *.lptpl – оформленные в формате HTML. Для организации взаимодействия с пользователем имеются специализированные механизмы:

Действия пользователя на страницы обрабатываются через стандартный механизм навигации, например: window.location="index.lptpl "

Для передачи на сервер пакета значений можно использовать механизм: ViewINF.lptpl?tpinf=1&idinf="+Part. Где Part - текстовое значение, не более 256 символов. Обработка на стороне сервера будет происходить согласно структуре выходных данных, по имени передаваемого параметра. В представленном выше примере: *tpinf* и *idinf*

Организация сессии для каждого активного пользователя обеспечивается компонентом, и со стороны программиста не требуются каких либо организационных действий.

Структура формирования запросов параметров и файлов из страницы шаблонов (*.lptpl):

Правила оформления	Назначение
{параметр}	Запрос значений (8 символов, латиница, только нижний регистр, без специальных символов и пробелов)
{+параметр}	Повторяющийся запрос значений, до момента прекращения передачи данных по ним
{:параметр}	Запрос имени файла
{*параметр}	Повторяющийся запрос имени файла
{имя файла.inc}	Указание о внедрении заданного файла, с расширением INC (обязательное расширение), содержимое как есть
{*имя файла.inc}	Повторяющийся запрос заданного файла, до момента прекращения передачи данных по параметрам

Дополнительный формат – файлы с расширением в формате INC предназначены для динамического внедрения шаблона (.inc) в страницу (*.lptpl) необходимое количество раз.

Страницы *.inc оформляются языком разметки HTML, но не содержат тегов страницы, и т.п., а отражают только необходимый фрагмент вставки. Например, страница **A.lptpl** имеет следующую структуру:

```
<TABLE width="100%" border="0" cellspacing="5px">
        <TR><TD valign="middle" colspan=2 onmouseover="this.className='Vs';"
onmouseout="this.className='Vr';" class="VrS"><a onclick="Navigation(26{idp})"> Добавить
индикатор в процесс</a></TD></TR>
        <TR><td colspan=2 class="TF">Задействованные индикаторы</td></TR>
       {*editinfprocess.inc}
 </TABLE>
```

Вставка - {*editinfprocess.inc} указывает, что необходимо загрузить шаблон editinfprocess.inc столько раз, сколько отдает Web-сервер. После этого скомпонованный текст страницы загружается в A.lptpl, а та предоставляется пользователю в браузере.

Структура страницы editinfprocess.inc:

```
<TR><TD>{infnameinprocess}</TD><TD class="c" width="20%"
onmouseover="this.className='hon';" onmouseout="this.className='hoff';">{inipdel}</TD></TR>
```

XML

Компонент предназначен для организации работы с файлом в формате XML. Конфигурирование экземпляра компонента в дереве проекта позволяет описывать набор используемых атрибутов, а также групп поиска по заданным узлам.

В свойствах – путь доступа к файлам, при создании файла, наличие директорий должно быть

Сохранение узла, он становиться текущим, если сделать создание узла, то он будет подчиненным первому, а не на одном уровне

Новый узел всегда в конце списка

XML выборка

Компонент предназначен для организации процесса поиска по файлу XML. В качестве условий поиска используются названия узлов, атрибутов и их значения. Возможно организовать несколько способов поиска: по текущему уровню узла, по всем вложенным узлам. Поиск ведется относительно текущего узла в файле XML. Результат выборки – одноуровневый список узлов удовлетворяющих условиям поиска.

Внимание! Компонент работает только на основании ссылки на компонент работы с XML файлом.

Справочник субкомпонентов

Взаимодействие

События

Список событий, которые будет предоставлять в LP-программу проектируемый компонент;

Команды

Список команд, которые будет предоставлять в LP-программу проектируемый компонент;

Входные данные

Список входных данных, которые может принять проектируемый компонент;

Выходные данные

Список выходных данных, которые может отдать проектируемый компонент.

Логические операции

IF

Процесс организации сравнения двух параметров. Один из параметров может быть задан как константа. Конфигурирование субкомпонента позволяет задавать метод сравнения. При сравнении текстового значения с неопределенным параметром, сравнение производиться по размеру текста передаваемого в качестве одного из параметров значения. При описании констант для проведения процесса сравнения можно использовать специализированная запись типа константы. Т.е. проведение явной типизации сравниваемых значений.

Внимание! Тип сравнения задается типом первого параметра. Константа может быть применена как в первой, так и во второй части сравнения, в зависимости от наличия входящей связи по данным. Если связь по данным применена к P2, то константа будет применена как P1.

Переменные и константы

Переменные

Контейнер для хранения заданного описания переменных. Переменные могут быть объявлены как «Константы» или как «Переменные». Типа данных в значении переменной определяется либо типом предаваемого значения для хранения, либо типом заданного значения. Типизация определяется по значению, либо по форме записи.

Внимание! При перезаписи значения в переменной, следует помнить: если новое значение null, то перезапись не осуществиться. В итоге в переменной будет содержаться предыдущее значение. Размер значения в переменной до 8Кб;

Структура

Инструмент, предоставляющий средства по работе с данными в виде заданной структуры (пакета данных). Элементы структуры могут быть объявлены как: BYTE, WORD, DWORD, Строка. Существует возможность как получать структуру и предоставлять доступ к ее элементам, так и формировать структуру как единой набор;

Счетчик

Субкомпонент, предоставляющий сервис – счетчик. Счетчик может быть проинициализирован начальным значением. Значение счетчика может быть в диапазоне «плюс» и «минус»;

Массив

Субкомпонент, предоставляющий сервис ведения массива данных. Размерность массива объявляется в конфигурировании объекта и не может быть динамически изменена. Набор команд объекта предоставляют сервис по перемещению, поиску по массиву, инициализации элементов массива и т.п. **Внимание!** Команда «Fnd» всегда осуществляет поиск с 0 элемента массива, если поиск был неудачным, то индекс там, где был до поиска. Команда «FndN» осуществляет поиск со следующей от текущей позиции массива, если поиск был неудачным, то текущая позиция в

массиве на последнем элементе; Команда «Fnd» не приводит к сообщениям об ошибке или успешном изменении индекса;

Биты

Биты – субкомпонент предназначен для осуществления работы с битовыми значениями. Конфигурирование объекта позволяет указать тип обрабатываемых данных – BYTE, WORD, DWORD.

Управление

Маршрутизатор

Субкомпонент предназначен для организации переключения схем отношений между объектами разрабатываемого компонента. Режимы работы объекта:

Команды – т.е. управление схемой ответной реакции на получаемые команды:

- 1 вход – N выходов;
- N входов – N выходов.

Данные схемы позволяют решить задачу, когда необходимо активировать различные действия для разных режимов работы логики проектируемого компонента.

Данные – т.е. управление потоками значений:

- 1 вход – N выходов;
- N входов – N выходов.

Данные схемы позволяют решить задачу, когда необходимо предоставлять различные данные при различных режимах работы проектируемого компонента.

Управление маршрутами может быть выполнено с помощью соответствующих команд субкомпонента, а также через установку значения – номера маршрута. Пример:

События:

- Act0...n - событие о том, что маршрут установлен;
- Out0...n - событие о том, что активирован маршрут;
- Act0...n - событие о том, что маршрут установлен;
- Out0...n - событие о том, что активирован маршрут.

Команды:

- In - активирует текущий маршрут;
- Set - назначить маршрут по номеру Rnm;
- Nx - следующий маршрут.

Триггер запроса данных

Субкомпонент, предоставляющий сервис получения события в рамках проектируемого компонента, как ответную реакцию при попытке получения данных между объектами компонента;

Триггер получения данных

Субкомпонент, предоставляющий сервис получения события в проектируемом компоненте, как ответную реакцию на процесс получения данных компонентом. Т.е. когда значение в контролируемом объекте изменилось;

Очередь данных

Субкомпонент, позволяющий организовывать управление очередью событий в рамках проектируемого компонента;

Трансмиттер

Субкомпонент обеспечивает поддержку сервиса принудительной передачи значений (по цепочке связи по данным) в режиме Multicast *(специальная форма широковещания, при которой копии пакетов направляются определённому подмножеству адресатов*). **Внимание!** Использование трансмиттера должно быть последним в цепочке логики, так как после посылки Multicast следующие связи не обрабатываются;

Генератор

Субкомпонент предоставляет сервис для организации циклов Следует помнить, что данный объект используется как начало требуемого цикла, после которого идут связи с требуемыми субкомпонентами. Цикл должен обязательно получать команду окончания работы, в противном случае, новая команда начала цикла не будет обрабатываться. Компонент не поддерживает количество циклов. Этот режим должен быть организован иными субкомпонентами до начала работы объекта – генератор.

Внимание! Пульс (событие - генератора) работает от пульса по цепочке, в рамках которой и нужно остановить цикл пульса. Команда окончания работы Генератора ДОЛЖНА быть подана до окончания цепочки событий от точки “Pls” – Генератора.

Операции

Сложение

Субкомпонент предназначен для проведения операций сложения. Применим как для текстовых значений, так и для математических значений. Тип значений определяется по их содержимому или по указаниям в области описания констант;

Вычитание

Субкомпонент предназначен для осуществления математической операции вычитания;

Умножение

Субкомпонент предназначен для осуществления математической операции умножения;

Регулярные выражения

Субкомпонент обеспечивает поддержку процедуры обработки текстовых значений по правилам «регулярных выражений».

Регулярные выражения - формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

Внимание! Перед сборкой строки в объект должен быть загружен шаблон (хотя бы один раз пред использованием), что приведет к возникновению события:

строка собрана

или

после сборки строка не соответствует шаблону

Если после загрузки шаблона, и в случае отсутствия данных при сборке строки, блоки остаются равные значениям в шаблоне, при этом ошибки не возникает, так как строка все еще соответствует шаблону;

Битовый сдвиг влево

Изменение позиций битов в слове на одну и ту же величину;

Битовый сдвиг вправо

Изменение позиций битов в слове на одну и ту же величину.

Биты содержатся группами по 8, 16, 32 или 64 битов в словах. Все сдвиги похожи друг на друга поведением средних битов, которые просто сдвигаются влево или вправо на определённую величину (2). Однако, поведение крайних битов, которые уходят из слова и которые появляются в слове, зависит от типа сдвига.

Организация работы LP-машины под Java, через интернет браузер

Организация процесса доступа к данным в приложениях, предоставляемых через интернет браузер, в режиме сохранения значений на стороне web-сервера, требует специализированного оформления.

Страница, оформляющая запуск LP-машины в Java исполнении должна содержать тег APPLET с параметрами, указывающими принцип организации передачи данных от LP-программы на сторону web-сервера.

```
<APPLET code="lpmachine/MyApplet.class" width=970 height=750 ARCHIVE="../LPMachine.jar" >
    <param name="ScriptFile" value="test/Part-02-12.LPScp">
    <param name="Mode" value="">
    <param name="PostFile" value="qqq.php">
    <param name="IniString" value="ждите загрузку программы...">
    <param name="BGColor" value="#FF00FF">
</APPLET>
```

- code="lpmachine/MyApplet.class"-неизменяемый, необходимый параметр(имя запускаемого апплета)
- width=* и height=* - необходимый параметр (значение в пикселях или в процентах)
- ARCHIVE="*" необходимый параметр путь до архива LPMachine.jar

Значения параметров:

- ScriptFile – необходимый параметр (путь до исполняемого файла *.LPScp), имя запускаемой LP-программы;
- Mode – необязательный параметр, при наличии значений создается окно консоли и запускается режимы: -d (*режим отладки*) -v (*verbose*) -r (*выводится репозиторий*);
- PostFile – необязательный параметр. Имя файла, обрабатывающего метод POST на серверной стороне;
  Передаваемые параметры методом POST:
  - filename - имя файла;
  - content - содержимое файла;
  - mode - режим работы с файлом ("w +" - запись в начало файла, "a +" - запись в конец файла , "d" - удаление файла).
- IniString – необязательный параметр. Строка, появляющаяся в апплете, пока грузится содержимое апплета;
- BGColor – необязательный параметр. Цвет фона апплета в синтаксисе html.

Пример оформления обработки получаемых данных на стороне web-сервера указанного в параметре – PostFile.

```
<?php
    $filename=$_POST['filename'];
    $content=$_POST['content'];
    $mode=$_POST['mode'];
    $content = stripslashes($content);
    $dst = $_SERVER['DOCUMENT_ROOT'].$filename;
    $mode_in="w+";
    if ($mode=="d")         unlink($dst);
    else    {
            if ($mode=="a")  $mode_in="a+";
            $handle=fopen($dst, $mode_in);
            fwrite($handle, $content);
            fclose($handle);
    }
?>
```

Системные требования

Для работы среды программирования LogicProgram

Требования для работы LP-Studio под управлением версий операционной системы Windows соответствуют системным требованиям соответствующих версий ОС.

LP-Studio ориентирована для работы под следующие версии ОС: Windows XP, Windows2003, Vista, Windows7.

Среда исполнения

Платформа LogicProgram представляет специализированные машины исполнения скриптов (LP-программ). Один и тот же скрипт может быть исполнен под разными операционными средами.

LP-машины платформы LogicProgram:

- Операционная система – Windows;
- Операционная система – Linux;
  - Fedora 9 - Ядро Linux 2.6.25-14-fc9.i686;
  - Fedora 11 - Ядро Linux 2.6.29.4-167-fc11.i686.PAE;
- Операционная система – QNX версия 6.3.2 и 6.4.1;
- Операционная система – Android версия 2.2 и выше
- Среда Java под управлением операционной системы – Windows;
- Среда Java под управлением интернет браузера.

Для работы LP-машин

Операционная система – Windows

Требования для работы LP-машины под управлением версий операционной системы Windows соответствуют системным требованиям соответствующих версий ОС.

LP-машины ориентированы для работы под следующие версии ОС: Windows XP, Windows2003, Vista, Windows7.

Операционная система – Linux

Для работы LP-машины в Linux, необходимо наличие библиотек:

- libQtCore.so.4.5
- libQtGUI.so.4.5
- libQtWebKit.so.4.5

LP-машина реализована для работ под следующими вариантами Linux систем:

- Fedora 9 - Ядро Linux 2.6.25-14-fc9.i686
- Fedora 11 - Ядро Linux 2.6.29.4-167-fc11.i686.PAE

Операционная система – Android

LP-машина реализована для работ с следующими версиями ОС Android:

- версия 2.2 и выше

Операционная система – QNX

Для работы LP-машины в QNX версий 6.3.2 и 6.4.1, необходимо:

1. Установить библиотеку libxml2 (последнюю версию библиотеки libxml2-2.7.2.tar можно получить по адресу: www.xmlsoft.org), далее:
   - поместить архив libxml2-2.7.2.tar.gz по пути: /usr/local
   - распаковать архив: tar -xvzf libxml2-2.7.2.tar.gz
   - сконфигурировать библиотеку libxml2: ./configure --prefix=/usr/local
   - скомпилировать библиотеку libxml2: make
   - инсталлировать библиотеку libxml2: make install

2. Прописать ассоциацию:
    - Photon File Manager->Edit->Associations…->Add
    - Pattern: *.LPScp
    - Open: /full/path to LPMachine
    - View: ped
    - Edit: ped
3. Запустить скрипт *.LPScp

Для запуска в терминале введи строку по образцу:

cd /full/path to LPMachine ./LPMachine /full/path to *.LPScp

Среда Java SE под управлением операционной системы – Windows

Для работы LP-машины в качестве Java-программы под управлением операционной системы Windows2003, Vista, Windows7 требуется предварительная инсталляция - JDK 1.6 фирмы Sun Microsystems на машине пользователя. *JDK* — бесплатно распространяемый Sun Microsystems комплект разработчика приложений на языке Java, включающий в себя стандартные библиотеки классов Java.

Среда Java EE под управлением интернет браузера

Для работы LP-машины под управлением браузера: Internet Explorer или Mozilla Firefox работающих под управлением операционной среды Windows2003, Vista, Windows7, необходимо наличие виртуальной машины - Java SE v1.6.0.20, поддерживающей исполнение Java-приложений, без компилятора и других средств разработки. Структура Java SE v1.6.0.20: виртуальная машина — Java Virtual Machine и библиотеки Java-классов. Сборка включает в себя:

- Java Development Kit (JDK) v1.6.0.20 x86
- Java SE Runtime Environment (JRE) v1.6.0.20 x86
- Java DB 10.5.3.0 x86

Приложения

Правила определения типа данных

Без знака

- **BYTE(Num);** - Тип byte может представлять целые числа в диапазоне от 0 до 255 включительно.
- **WORD(Num);** - Тип word может представлять от 0 до 32 767 включительно.
- **DWORD(Num);** - Тип dword может представлять без знаковое целое число от 0 до 2 147 483 647.

Со знаком

- **INT(Num);** - Тип int может представлять целые числа в диапазоне от -32 768 до 32 767 включительно.
- **LONG(Num);** - Тип long может представлять целые числа в диапазоне от -2 147 483 648 до 2 147 483 647 включительно.
- **DOUBLE(Num);** - Тип double может представлять числа не более 10+308 (положительные и отрицательные) с точностью до 15 знаков и не менее 10-323. Тип double также может представлять значение NaN (не число), положительную и отрицательную бесконечность, а также положительный и отрицательный нуль.

Текстовые

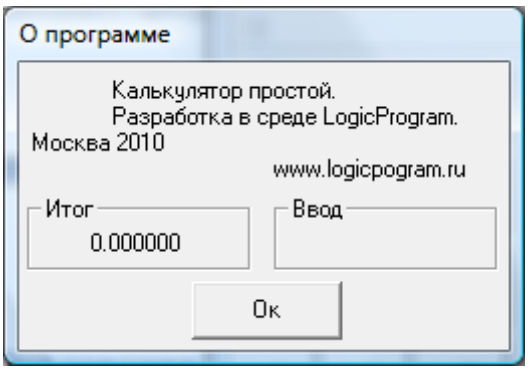
- **STRING("Str");** - Длина строки String может составлять от нуля до 8Кб.

Дата/Время

- **DATETIME("Str");** - Хранение даты и времени. Даты от 1.01.100 до 31.12.9999 Время от 00:00:00 до 23:59:59
- **DATE("Str");** - Хранение даты. Даты от 1.01.100 до 31.12.9999
- **TIME("Str");** - Хранение времени. Время от 00:00:00 до 23:59:59

Синтаксис описания значений для типа данных “String”

STRING(“…….”);
\t – отступ
\n – перевод строки
Пример: STRING("\tКалькулятор простой.\n\tРазработка в среде LogicProgram.\nМосква 2010\n\t\t\twww.logicpogram.ru");



Краткая справка по регулярным выражениям

Искомые выражения

Выражением может быть один символ или последовательность символов, заключенных в круглые или квадратные скобки.

Используя квадратные скобки, можно указать группу символов (это называют классом символов) для поиска.

Например, конструкция 'ма[ши]на' соответствует словам «машина» и «машинами», т.е. словам, начинающимся с «ма», за которым следуют «ш» или «и», и заканчивающимся на «на».

Возможно и обратное, то есть, можно указать символы, которых не должно содержаться в найденной подстроке. Так, '[^1-3]' находит все символы, кроме цифр от 1 до 3.

Если неизвестно, сколько именно знаков должна содержать искомая подстрока, можно использовать спецсимволы, квантификаторы (quantifiers). Например, можно написать "hel+o", что будет означать слово, начинающееся с "He", со следующими за ним одним или несколькими "l", и заканчивающееся на "o". Следует понять, что квантификатор относится к предшествующему выражению, а не отдельному символу.

Символ Описание

***** Соответствует отсутствию или нескольким вхождениям предшествующего выражения. Например, 'zo*' соответствует "z" и "zoo".

+ Соответствует 1 или более предшествующих выражений. Например, "zo+" соответствует "zo" and "zoo", но не "z".

? Соответствует 0 или 1 предшествующему выражению. Например, 'do(es)?' соответствует "do" в "do" or "does".

{n} n – неотрицательное целое. Соответствует точному количеству вхождений. Например, 'o{2}' не найдет "o" в "Bob",но найдет два "o"' в "food".

{n,} n – неотрицательное целое. Соответствует вхождению, повторенному не менее n раз. Например, 'o{2,}' не находит "o" в "Bob", зато находит все "o" в "foooood". 'o{1,}' эквивалентно 'o+'. 'o{0,}' эквивалентно 'o*'.

{n,m} m и **n** – неотрицательные целые числа, где n <= m. Соответствует минимум n и максимум m вхождениям. Например, 'o{1,3} находит три первые "o" в "fooooood". 'o{0,1}' эквивалентно 'o?'. Пробел между запятой и цифрами недопустим.

Концы и начала строк

Проверка начала или конца строки производится с помощью метасимволов ^ и $. Например, "^thing" соответствует строке, начинающейся с "thing". "thing$" соответствует строке, заканчивающейся на "thing". Эти символы работают только при включенной опции 's'. При выключенной опции 's' находятся только конец и начало текста. Но и в этом случае можно найти конец и начало строки, используя escape-последовательности \A и \Z. Все это относится только к Perl-совместимым реализациям. Остальные же будут искать только конец и начало текста. В .Net имеется еще и символ \z, точный конец строки.

Граница слова

Для задания границ слова используются метасимволы '\b' и '\B'.

Позиционирующие символы

Символ	Значение
^	Начало строки
$	Конец строки, или перед \n в конце строки(см. опцию m)

\A	Начало строки игнорирует опцию m)
\Z	Конец строки, или перед \n в конце строки (игнорирует опцию m)
\z	Точно конец строки (игнорирует опцию m)
\G	Начало текущего поиска (Часто это в одном символе за концом последнего поиска)
\b	На границе между \w (алфавитно-цифровыми) и \W (не алфавитно-цифровыми) символами. Возвращает true на первых и последних символах слов, разделенных пробелами
\B	Не на \b-границе
\w	Слово. То же, что и [a-zA-Z_0-9]
\W	Все, кроме слов. То же, что и [^a-zA-Z_0-9]
\s	Любое пустое место. То же, что и [\f\n\r\t\v]
\S	Любое непустое место. То же, что и [^ \f\n\r\t\v]
\d	Десятичная цифра. То же, что и [0-9]
\D	Не цифра. То же, что и [^0-9]

Примеры регулярных выражений

Получение из строки дата/время, выделенных параметров:

Строка	Выражение
2010.11.11 7:45:42.789	(?<year>.{4})\.(?<month>.{1,2})\.(?<day>.{1,2}) (?<time>.*)\.(?<pTime>.*)

Формирование строки по шаблону:

Строка	Выражение
наименование PLC:наименование устройства: характер проблемы:2	(?<IDPLC>.*):(?<TNAME>.*):(?<CDESK>.*):2

Формирование HTML строки по шаблону:

Строка	Выражение
<TABLE class="plcTel" id="nf1:2:0"><TR><td class="TelTit">TXT1</td><td> </td><td class="TelTit">TXT2</td></TR></TABLE><br style="font-size:2px;">	<TABLE class="plcTel" id="nf(?<PINf>.*)"><TR><td class="TelTit">(?<TXT1>.*)</td><td> </td><td class="TelTit">(?<TXT2>.*)</td></TR></TABLE><br style="font-size:2px;">

Формат файла проекта

Описание иерархии и возможных узлов

Object			Корневой узел. Задает базовые параметры объекта
	Draw		Узел описывает параметры и способы отрисовки объекта в редакторе
		Skin	В слычае использование механизма скинов описывает скин
			Top TopLeft TopRight Left Right BottomLeft BottomRight Bottom — Расположение основных частей скина

PropDesc			Узел описаний свойств объекта
	<propname>		Описание и характеристики свойства объекта с именем "propname". **ВНИМАНИЕ**! Название тэга не должно оканчиваться цифрой
		ListItem	В случае если тип данных List этими тегами перечисляются варианты
Prop			Значение свойств объекта по умолчанию
	<propname>		Значение свойства объекта с именем «propname»
EventsDesc			Узел описания событий объекта
	Event		Описание конкретного события
CommandsDesc			Узел описания команд объекта
	Command		Описание конкретной команды
DataOutsDesc			Узел описания источников данных
	DataOut		Описание конкретного источника
DataInsDesc			Узел описания приемников данных
	DataIn		Описание конкретного приемника

Атрибуты узла Object

Platform - Код (комбинация бит) платформы
1 - GUI Windows
2 - GUI *nix
4 - Console
8 - GUI Portable
Type - Определяет тип объекта
1 - визуальные объекты
2 - специальные
TopLevel - Признак объекта верхнего уровня (Объект который объединяет в себе все другие визуальные объекты)
0 - не является объектом верхнего уровня
1 - является объектов верхнего уровня
ShortName - Короткое название объекта
uuid - Уникальный идентификатор объектов данного типа
Dll - Название сервисной библиотеки для редактирования свойств объекта и получения списков событий, команд, источников и приемников данных

Атрибуты узла описания свойства объекта (PropDesc/propname)

RefreshSense - Перестройка индивидуальных списков событий, команд, источников и приемников данных
0 - не требуется (значение по умолчанию)
1 - требуется
Refresh - Способ обновления объекта после изменения свойства
0 - Обновление не требуется (значение по умолчанию)
1 - Перерисовать содержимое объекта
2 - Перерисовать содержимое родительского объекта
IsGroup - Признак группового параметра. Это свойство задает число повторений свойств группы. Тип этого узла должен быть Int или UInt
0 - Не является группой (значение по умолчанию)

1 - Является группой
ПРИМЕЧАНИЕ: Для создания уникальных имен тегов групповых параметров к их исходному имени добавляется номер. Например если ColName базовое имя то в свойствах будут имена ColName1,ColName2,ColName3 и т.д.
Min - Минимально возможное число повторений группы (работает только в сочетании с IsGroup=1)
Max - Максимально возможное число повторений группы (работает только в сочетании с IsGroup=1)
Type - Определяет тип данных свойства (String,Int,UInt,Color,Bool,List)

Атрбуты узла ListItem

Value - Значение присеваемое свойству
Name - Название свойства

Атрибуты узла EventsDesc/Event, CommandsDesc/Command, DataOutsDesc/DataOut, DataInsDesc/DataIn

PropNum - Номер события (1-65535)
Desc - Описание события

Глоссарий

RAD (быстрая разработка)

Методология быстрой разработки призывает программистов начинать работу не с детализированного набора требований, а с написания небольших функциональных частей, которые можно завершить в течение короткого времени (на языке данного подхода такие этапы называют «итерациями»). При этом тестированию отдельных модулей уделяется не меньшее внимание, как и реальному написанию кода. При завершении одной итерации разработчики принимаются за следующее требование, которое добавит новую функциональность к только что законченному модулю, и таким образом начинают новую итерацию. Применение быстрой разработки предоставляет возможность программистам создавать высококачественные решения с необходимой функциональностью при меньших затратах времени и ресурсов, чем в традиционных методах проектирования «ПО».

Определение LP-программы

LP-программа является законченным решением основанным на компонентах платформы LogicProgram, и оформленным в виде сети (матрицы) отношений между компонентами. Формат файла LP-программы: XML. При этом LP-программа может быть модулем для многократного использования в других проектах. Данный подход обеспечивает основное свойство сборочного программирования – конвейерную сборку программ.

Код LP-программы не зависит от операционной платформы функционирования. Однако следует помнить, что часть компонентов платформы LogicProgram ориентирована только на определенную операционную среду.

Java-апплет

Java-апплет — программа на Java в форме байт-кода. Java-апплеты выполняются в веб-браузере с использованием виртуальной Java машины (JVM), или автономно.

Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не могут быть предоставлены средствами HTML. Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться с помощью плагинов браузерами многих платформ, включая Microsoft Windows, UNIX, Apple Mac OS и GNU/Linux.

Платформа LogicProgram предоставляет java-апплет машины LP, в рамках которой исполняется скрипт платформы в неизменном виде. В тексте может использоваться понятие апплета непосредственно к программе на языке LogicProgram. Однако следует понимать, что фактически скрипт не имеет отношения к java технологии и может быт исполнен в виде java-апплета только под управлением LP-машины для Java.

Предметный указатель

Т

V

W

X

А

Б

В

Г

Д

З

И

К

М

О

П

Р

С

Т