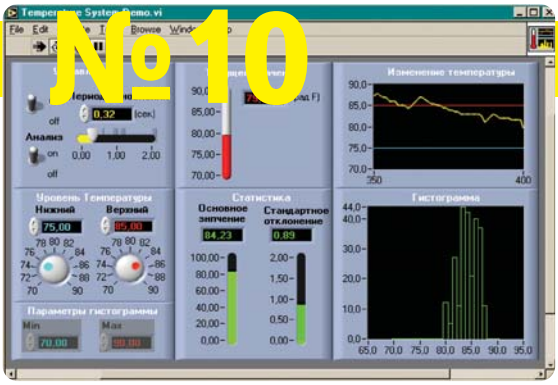
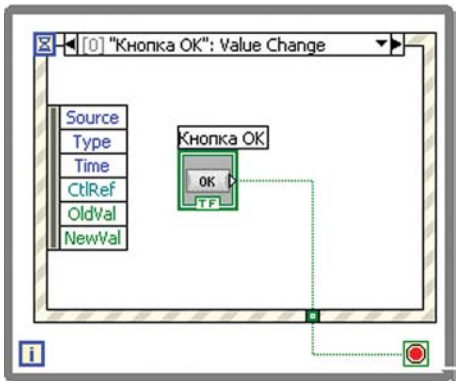
Уроки по LabVIEW



В LabVIEW программировать управление интерфейсом пользователя можно таким же образом, как пишется любая другая часть программы. А именно – использовать технику поллинга (постоянного опроса) элементов передней панели VI. Но в Вашем, уже должно быть самом любимом языке программирования G, существует возможность делать это более просто и эффективно, используя Event Structure. What is an Event Structure? Об этом и пойдет речь на этом уроке, но не только...

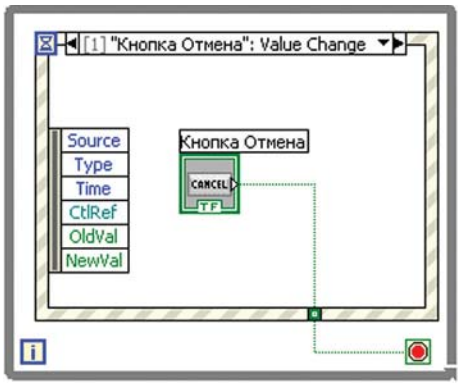
Event Structure - программная конструкция, описывающая событие, которое вызывает программное прерывание. Она позволяет применять программирование типа "событие-запуск" (Event-Driving). Такая модель успешно применяется и в других языках программирования - LabWindows/CVI и Visual Basic. Что же при этом происходит? Программа "спит" до тех пор, пока не произойдет что-нибудь интересное на передней панели вашего прибора. Реакция наступает быстрее, чем в случае постоянного опроса всех элементов панели. LabVIEW использует дополнительные возможности операционной системы для получения информации о действиях пользователя. Таким образом, операционная система дает возможность работать другим программам в то время, пока Ваша программа находится в ожидании. За какими элементами Вам необходим контроль Вы определяете самостоятельно при конфигурировании Event Structure.

Event Structure является такой программной "штучкой", которая лежит на пересечении функций "Wait on Occurrence" и "Case Structure". Имея сходство с "Case Structure", **Event Structure** содержит множество субдиаграмм, каждая из которых конфигурирует управление одним или несколькими событиями (например, нажатие кнопки мыши). Вы можете перетаскивать Event Structure по диаграмме, как это обычно делается в LabVIEW с любым другим объектом. Когда LabVIEW исполняет Event Structure, то код программы не исполняется. Примерно так, как при исполнении функции Wait, т.е. до тех пор, пока не наступит нужное событие. Когда событие наступает, Event Structure автоматически "просыпается" и выполняет соответствующую подпрограмму, управляя этим событием. Каждая поддиаграмма имеет коннектор **Event Data Node**, расположенный внутри на левой границе, и позволяет получить доступ к информации о наступившем событии. Этот коннектор похож на функцию **Unbundle by Name**.



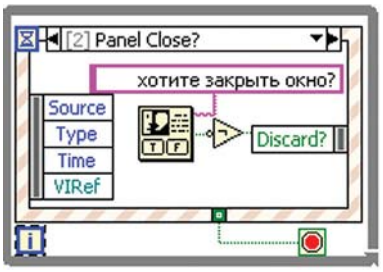
Рассмотрим простейший пример, в котором используется Event Structure для отслеживания нажатий клавиш "OK" и "Cancel", расположенных на передней панели, и выдачи диалогового окна при попытке закрыть панель. Заметим, что передняя панель обязательно должна быть активной.

Обратите внимание, что **Event Structure** помещены в цикл **While Loop**. Сделано это потому, что Event Structure исполняется только один раз, когда наступает событие, и если Вы хотите повторить отслеживание события, то следует повторить процесс обнаружения события. Всегда помните об этом! А если забудете, то получите проблему. В этом случае Event Structure выполняется один раз и не вызывается вновь. Это объясняется тем, что когда наступает событие и в Вашей программе содержатся другие Event Structure, управление которыми сконфигурировано, LabVIEW "защелкнет" переднюю панель до тех пор, пока другое событие не наступит. Это сделано для того, чтобы не потерять ни одного события и всегда выполнять прерывания в порядке, в котором они поступают. Вы можете обойти этот сценарий с защелкиванием передней панели, выключив опцию **Lock Panel** в окне **Edit Events Handled by This Case**, которое вызывается щелчком правой кнопки мыши по рамке структуры. Но это может привести к другим неожиданным результатам. Главное - поместить **Event Structure** в цикл **While Loop**.



Любые действия пользователя приводят к генерации событий, включая нажатие на клавишу мыши, клавиатуры, изменение какого-либо параметра, выделение пункта меню и закрытие окна. Вы выбираете, какое событие Вы хотите отслеживать, посредством использования диалогового окна конфигурации, которое предоставляет Вам возможность выбора событий, ассоциированных с каждой поддиаграммой. Эти события определены в двух классах **Notify Events** и **Filter Events**. **Notify Events** - это просто ссылка на свершившийся факт. Например, **Value Changed** (изменение состояния) для кнопок "OK" и "Cancel" относятся к **Notify Events**. С другой стороны, **Filter Events** позволяет Вам программно воздействовать на результат события, которое вызвало это событие. Вы можете выбрать одно из двух: сбросить прерывание или

модифицировать данные, вызвавшие прерывание, прежде чем LabVIEW закончит их обработку. Например, событие **Panel Closing** (Закрытие панели) относится к **Filter Events**. В данном случае Вы можете согласиться или нет с предложенным выбором. **Filter Events** находятся на правой границе поддиаграммы. Если пользователь нажмет **Really close window?** диалоговое окно вернет значение **FALSE** и терминал **Discard?** (Отменить?) перейдет в состояние **TRUE** . Таким образом, LabVIEW сбросит прерывание без закрытия окна. Все **Filter Events** имеют терминал **Discard**, а некоторые дополнительно имеют другие поля, которые Вы можете модифицировать как событие **Pass Through** (пропустить). Например, поскольку **Key Down** относится к **FIlter Events**, Вы можете транслировать маленькие буквы из **String Control** в большие и передать их в следующий **Case**. Если Вы ничего не подсоедините к терминалу **Filter Events**, то ничего и не произойдет, никаких модификаций.



Есть еще несколько соображений по поводу нашего простого примера, прежде чем мы с ним закончим. Заметьте, что Вы имеете доступ как к старым (**old Value**), так и к новым (**New value**) значениям контрольных элементов наших кнопок, которые ассоциированы с **Event**. В этом есть определенный смысл. Поскольку кнопки OK и Cancel могут "защелкиваться" (**Latching Booleans**), необходимо считывать их состояние для осуществления их сброса. Если Вы только управляете **Value Changed** и никогда не считываете терминалы **Old value** и **New Value**, кнопки останутся нажатыми навсегда. Это очень хорошая идея - разместить терминал управляющего элемента внутри **Event Case**, если регистрация его состояния больше нигде не понадобится. Уделите особое внимание этому правилу, если Вы используете кнопку **Stop** для завершения цикла. Если Вы будете читать ее снаружи (**Event Case**), LabVIEW может и будет читать терминал прежде, чем **Event Case** исполнится, и, в результате, программа перейдет в режим ожидания и Ваш **Loop** будет исполняться дольше, чем Вы ожидаете.

Также обратите внимание на маленькую белую точку на туннеле терминала, который выходит из **Event Structure**. Нововведение было сделано еще в LabVIEW 6.1. Это означает автоматическое использование значения (по умолчанию), если к терминалу в других **Case** ничего не подсоединено. Вы можете включить или отключить эту опцию, щелкнув правой кнопкой мыши на туннеле и выбрав пункт **Use Default if Unwired**.

И, наконец, что это за песочные часы в левом верхнем углу? Это терминал **Time Out**, к которому Вы можете подсоединить значение в миллисекундах. **Event Structure** "проснется" и выполнится в любом случае по истечению этого времени. Если к этому терминалу ничего не подсоединено, **Event Structure** будет ждать вечно своего события. Возможно, Вам понадобится использовать этот терминал, если нужно что-либо исполнять дополнительно.

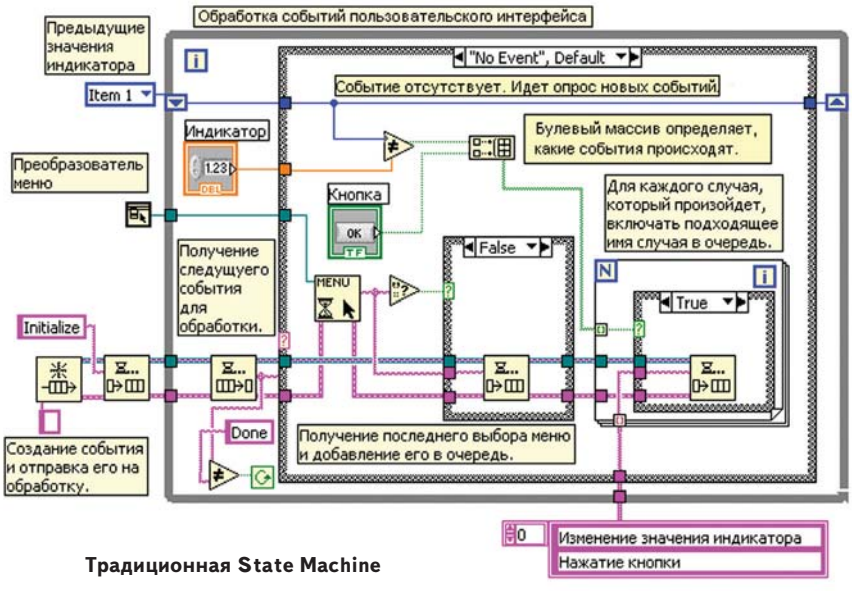
Вот что мы можем узнать из простого примера. Конечно, в реальной жизни программы оказываются более сложными, чем этот простой пример. Более "продвинутые" пользователи могут интуитивно прочувствовать, что использование **Event Structure** совместно с **State Machine** более интересно.

Машина Состояний. Если у Вас есть ранее написанные программы, в которых уже используется машина состояний для управления интерфейсом пользователя, то обычно удается довольно просто внедрить **Event Structure**. Есть смысл внедрять **Event Structure** в отдельные части машины состояний, которые опрашиваются для внесения изменений в интерфейс пользователя. В большинстве случаев Вы можете использовать **Event Structure** для расширения возможностей и упрощения Вашей машины состояний (а в наиболее простых случаях машина состояний может оказаться просто ненужной).

Традиционно, с минимальной скоростью или при отсутствии **Event Structure**, программа постоянно опрашивает состояние элементов передней панели и сравнивает их состояние с предыдущими в сдвиговых регистрах. Результаты этого сравнения показывают, какие кнопки были нажаты, или какие пункты меню были выбраны, и что после этого делать. К сожалению, этот подход имеет два ограничения:

- если происходит несколько изменений, то нет возможности отследить порядок поступления;
- если есть возможность изменения состояния контрольного элемента в новое состояние и обратно прежде чем программа отреагирует, то этот факт не будет зафиксирован.

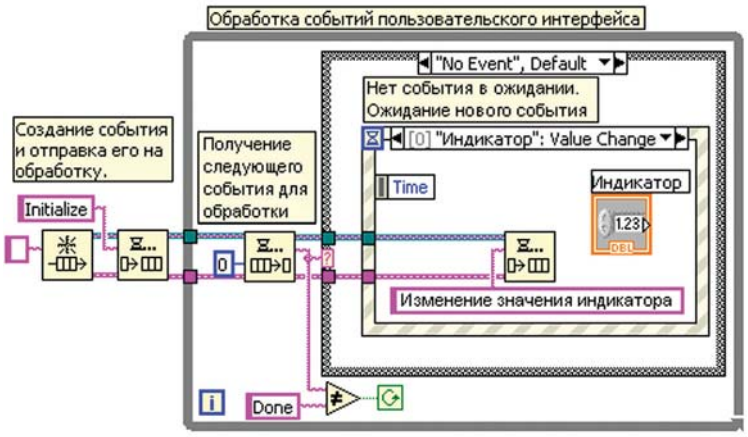
Эти изменения могут быть важными. Как упоминалось ранее, **Event Structure** имеет возможность фиксировать короткие и быстрые изменения. Модификация представленного ниже интерфейса пользователя с помощью **Event Structure** выглядит очень просто. Замените код в месте программы, где проверяются состояния контрольных элементов на **Event Structure**, конфигурируемых для регистрации изменения состояния **Value Changed**, для тех же самых контрольных элементов. Каждое значение, зафиксированное **Event Structure**, занесите в очередь **queue**.



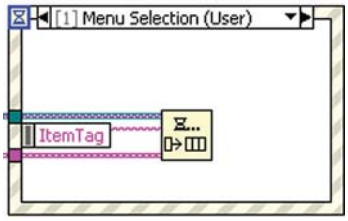
Традиционная State Machine

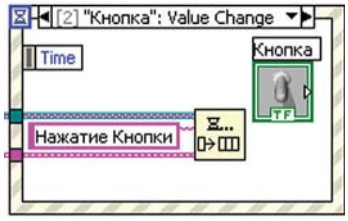
В блоке **No Event** проверяется, какой пункт меню выбран, нажата ли кнопка и изменилось ли значение элемента **ring**. Применяя **Event Structure**, Вы можете следить за выбором пункта меню, состоянием кнопки и изменением элемента **ring** так же успешно.

Итак, Вы можете заменить весь код внутри старого **Case** на одну **Event Structure**. Результирующая диаграмма будет выглядеть следующим образом:



Возможно, ранее Вы не встречались с некоторыми **VI**, которые изображены на рисунках (создание события, получение следующего события). Для их использования необходимо на функциональной панели выбрать **Functions >> Advanced >> Synchronization >> Queue Operations**.





Вдобавок к радикальному упрощению диаграммы, использование **Event Structure** имеет два больших преимущества. Первое, все элементы попадают в очередь только, если **Event Structure** немедленно не обрабатывает событие. Это значит, что Вы ничего не пропустите. Управление событиями происходит синхронно. События будут гарантированно обработаны в порядке их поступления, за исключением случая параллельной работы двух **Event Structure**. Однако помните, несколько потенциальных проблем могут возникнуть в случае использования **Event Structure** с более сложными машинами состояний. **Event Structure** изначально "спит" пока не наступит событие. В примере, рассмотренном выше, машина состояний будет оставаться в состоянии "**No Event**", пока пользователь ничего не делает. В большинстве случаев это желательно для минимизации загрузки процессора. Но если Вы хотите работать с платой АЦП, то это Вам не подходит! Размещение этого кода в "**No Event**" не приведет к успеху, поскольку цикл повторяется до наступления события. Для выхода из такой ситуации Вы можете использовать машину состояний вместе с очередью **queue** в отдельном цикле с использованием **DAQ Event** и записывать данные в очередь по мере их поступления. Лучше всего использовать **timeout** для **Event Structure** и исполнять Ваш код параллельно с **Event Structure**.

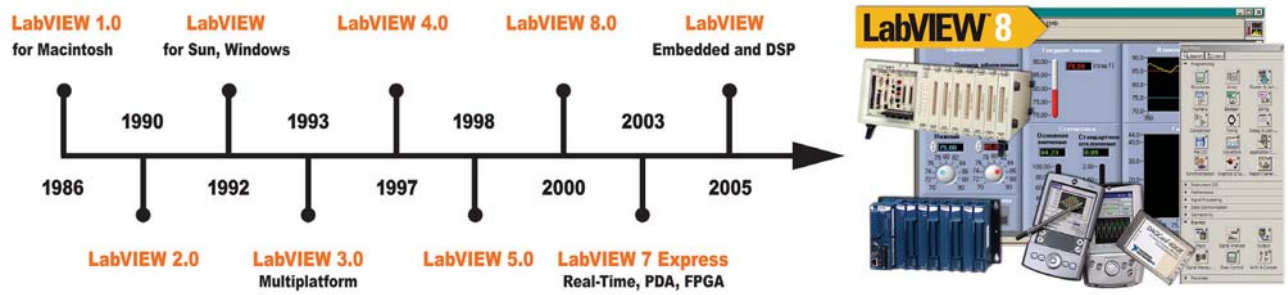
Еще одно предостережение. При использовании **Event Structure** с машиной состояний Вы должны быть уверены, что все **Event Structure** доступны к управлению при любых действиях пользователя. Как упоминалось ранее, если событие наступает, а отсутсвует **Event Structure,** который им управляет, то передняя панель "защелкнется " до тех пор, пока не наступит событие, для которого есть управление **Event Structure**. В лучшем случае это приведет к получению скрытого состояния вашего интерфейса. В худшем - это смерть для Вашей программы. Это очень, очень плохо. Но, к счастью, в меню есть кнопка **Abort.** Вы не должны забывать про эту кнопку.

Вроде как изложенный материал по объему невелик. Однако его освоение потребует от Вас особых усилий и немало времени. И сделать это нужно обязательно. Интерфейсная панель с ее многочисленными в больших проектах органами управления - это лицо всей Вашей системы.

А на следующих уроках Вас ожидают: технологии Express VI, математические функции, обработка и анализ сигналов, ... Но все это уже будет в новой версии программы - **LabVIEW 8!**

То, что базовый курс был изложен и, будем надеяться, освоен в другой среде, а именно в 6-й версии, совсем даже неплохо. LabVIEW 6i считается знаменательной версией, своего рода основой для всей последующей истории развития средств графического программирования. Именно после 6i технология LabVIEW начинает использоваться для программирования Pocket PC, систем реального времени, сигнальных процессоров, FPGA-структур и встраиваемых 32-разрядных микропроцессорных компонент.

Кстати, многие из продвинутых пользователей так и не захотели переходить на LabVIEW 7Express, считая 6i пределом мечтаний. Скорее всего потому, что были озабочены реальными текущими проектами. А в версию 7Express было внесено немало полезных новшеств. Об одном из них - синхронизированных циклах, речь шла на предыдущем уроке. А еще на урок раньше Вам настоятельно было рекомендовано переходить на 7Express. Но, время летит быстро. Так, что одной из задач домашнего задания - знакомство с LabVIEW 8.

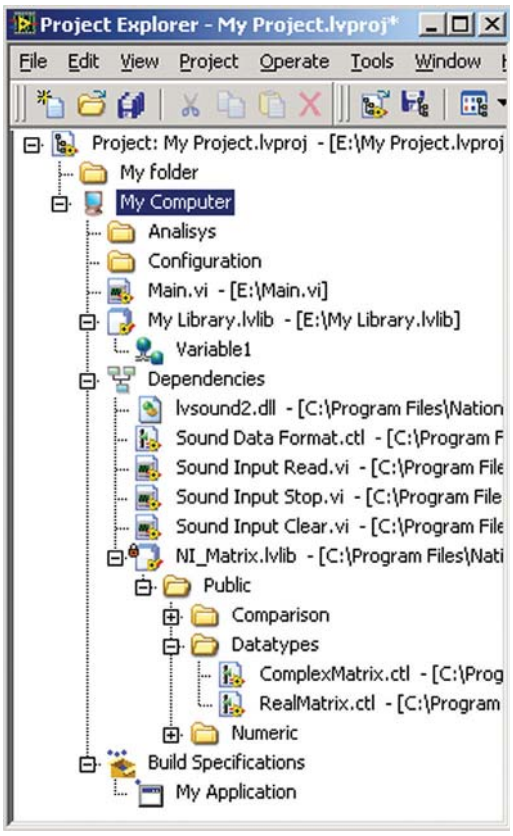


Если Вы внимательно рассмотрите эволюцию LabVIEW, то обязательно отметите для себя - а ведь у этого программного продукта в нынешнем году юбилей, 20 лет. Это серьезно, и заслуживает всяческого уважения.

Что же нового в 8-й версии? В двух словах не скажешь, но, попробовать можно. Давайте и посвятим этому вторую часть этого урока.

Есть новые функции и элементы управления? Конечно есть, и немало. Изменилось также содержание главного меню, состав и структура передней панели и диаграммы, содержание контекстных меню и многое др. Главное - не в этом. В LabVIEW 8 реализована концепция распределенного интеллекта на основе принципиально новых технологий и инструментария, которая включает:

- единую графическую платформу программирования и обычных компьютеров, и аппаратных платформ поддержки реального времени, и программируемую логику, и карманные PC, и микроконтроллеры, и сигнальные процессоры, т.е. все компоненты распределенной системы;
- интерактивную среду управления распределенными системами из единой оболочки **Project Explorer**, которая позволяет из одного окна проекта просматривать, редактировать, запускать и отлаживать код, работающий на любой целевой платформе;
- коммуникационный интерфейс для совместного использования данных **Shared Variable**, сконфигурировав который с помощью несложных диалоговых окон можно передавать данные между системами без потери скорости;
- возможность синхронизации как внутри систем, так и между распределенными устройствами.



А зачем все это надо? Для того, чтобы решать реальные проблемы, с которыми сталкиваются сегодня разработчики распределенных систем:

- программирование приложений, использующих многопроцессорную архитектуру, в том числе и смешанную - с микропроцессорами, FPGA и DSP;
- организацию эффективного обмена данными между несколькими процессорами, объединенными в сеть, или расположенными на одной плате;
- объединение всех узлов в систему, с решением задач тактирования и синхронизации работы составляющих ее узлов;
- интеграцию в единой системе различных видов ввода/вывода, таких как высокоскоростной аналоговый или цифровой В/В, техническое зрение и управление движением;
- добавление сервисных функций по обмену данными между узлами, включая протоколирование, выдачу сигналов тревог и взаимодействие с информационными системами корпоративного уровня.

Способность такого универсального программного продукта как LabVIEW 8 преодолеть перечисленные трудности позволяет резко упростить этап разработки, ее стоимость и сроки выполнения, а также повысить производительность труда разработчика.

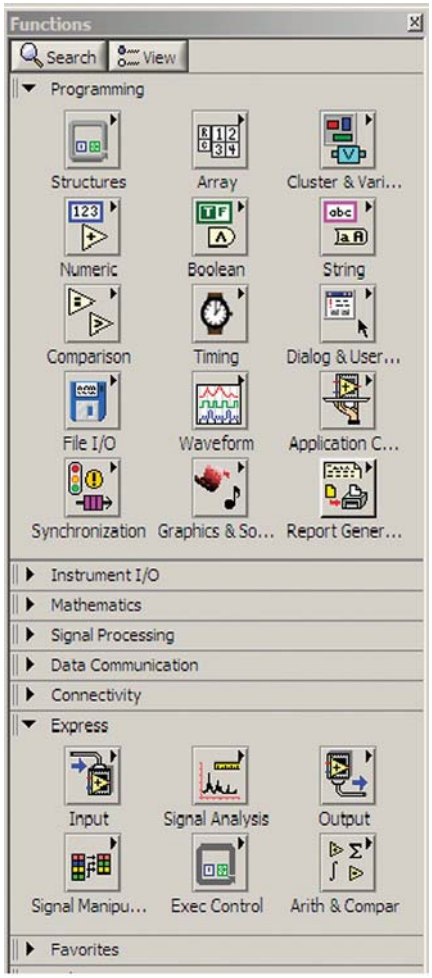
Project Explorer. Проекты, библиотеки и подбиблиотеки проектов необходимы для управления процессом разработки больших приложений. Приведенный на рисунке пример структуры проекта позволяет организовать различные конфигурации папок, файлов, библиотек виртуальных приборов и подбиблиотек проекта, общих переменных и элементов XControl.

Библиотеки проекта могут использоваться для организации файлов в единую иерархию элементов, решения проблемы дублирования имен VI, отслеживания версий, ограничения открытого доступа к определенным файлам и возможности редактирования набора файлов. Информация проекта, включающая ссылки на файлы проекта, данные о конфигурации, создании, развертывании и т.п., сохраняется в файле проекта XML (.lvproj). В проекте могут быть использованы виртуальные папки для организации всех файлов, которые составляют его приложение. Папки проекта являются по сути логическими и не отражают каталоги файлов на диске. Такая организация файлов способствует повторному использованию кода в нескольких проектах, каждый из которых имеет собственную организацию. Для упрощения перемещения существующего кода и сопутствующих файлов в LabVIEW 8 можно импортировать в проект папки вместе с их содержимым. После этого, конечно, теряется синхронизация папок и изменение в виртуальной папке не будут отражаться в папке операционной системы. Для больших приложений в новом LabVIEW предусмотрена интеграция программного обеспечения контроля версий (Source control) от третьих поставщиков вместо встроенной системы аналогичного назначения, имевшейся в LabVIEW 7. И еще. Для распределения компонент приложения используется LabVIEW Application Builder. Он может запускаться в окне проекта с помощью контекстного меню строки Build Specifications.

XControl. Элементы XControl представлены на диаграмме одной иконкой и это позволяет существенно уменьшить размер кода. Элемент XControl содержит 4 обязательных компоненты : **Date, State, Facade и Init**, и ряд дополнительных или определяемых пользователем компонент. Эти элементы являются виртуальными приборами или элементами управления (.ctls), которые и определяют функциональность XControl. С помощью компонент Date и State задается тип данных (скаляр, матрица и т.п.) и его текущее состояние (видимое, элемент управления, индикатор и т.п.). VI Facade управляет различными состояниями, в которые может перейти XControl, и его ответами на события пользователя. Он представляет собой большой VI обработки событий и вызывается LabVIEW только при на-

личии событий, ожидающих обработку. Для элементов XControl наряду с общими для этого класса могут создаваться и устанавливаться собственные свойства и методы.

Shared Variable. Эта новая переменная объединяет функциональность существующих технологий передачи данных, таких как DataSocket, и обеспечивает поддержку разработки распределенных многопроцессорных систем. Shared Variable позволяет передавать текущие данные между различными VI проекта или по сети, при этом источниками или приемниками данных могут быть элементы передней панели или диаграммы.



Shared Variable создается в окне проекта в составе библиотеки проекта с помощью контекстного меню узла этой библиотеки. Настройка параметров Shared Variable выполняетсяс помощью диалогового окна **Shared Variable Properties**, которое позволяет выбрать тип данных, тип переменной, использование и параметры буфера, и параметры подключения к источнику. Тип переменной может быть установлен как глобальный или как сетевой. Shared Variable с сетевым типом обеспечивает обмен данными между VI, удаленными компьютерами и приборами с помощью механизма **Shared Variable Engine**. Этот механизм использует протокол передачи данных NI **Publish-Subscriber-Protocol (PSP)**. Для связи элементов диаграммы VI с общей переменной ее иконка может быть перенесена из окна проекта или установлена из соответствующей подпалитры. Связь элементов передней панели с Shared Variable может устанавливаться также с помощью переноса иконки переменой из окна проекта на лицевую панель или путем конфигурирования свойств этих элементов **Data Binding.** Модули и инструменты LabVIEW, устанавливаемые пользователем, могут добавить Shared Variable типы, опции конфигурации и ограничения.

Организация палитр передней панели и диаграммы. В LabVIEW 8 реализован более удобный интерфейс среды разработки. На верхнем уровне палитры разделены на категории, состав и порядок расположения которых в палитре может быть оперативно изменен. Палитра функций диаграммы содержит такие группы как **Programming, Mathematics, Signal Processing, Instrument I/O, Data Communication, Connectivity и Express**. Произошло также изменение содержания категорий и подпалитр функций диаграммы. Наибольшие изменения в составе функций произошли в категории средств взаимодействия: добавлена подпалитра **Source Control**, а в подпалитрах В/В файлов - группы функций работы с datalog- и zip-файлами, с хранилищем данных, диалога и интерфейса пользователя (перенесены функции работы с меню, курсором и событиями) и линейной алгебры. Подпалитра структур дополнена такими структурами, как узел **Xmath Script**, структура отключения по условию **Conditional Disable Structure**, структура отключения диаграммы **Diagram Disable Structure**, и дополнена вложенной подпалитрой временных структур **Timed Structures**.

К новым элементам интерфейса пользователя следует отнести разделительные полосы **Splitter Burs**, аннотации графиков **Graph Annotations**, график смешанного типа **Mixed-Signal Graph**, возможность экспорта упрощенных изображений графиков, рисунков и таблиц. Расширилось содержание контекстных меню, и пунктов диалогового окна свойств элементов передней панели. При создании узлов методов и свойств этих элементов с помощью контекстного меню обеспечивается доступ ко всему набору методов и свойств. Аналогичная доступность выбора методов и свойств имеется теперь при установке узлов на блок-диаграмме, что обеспечивается с помощью диалогового окна просмотра класса **Class Browser**. Появилась возможность создавать пользовательские контекстные меню объектов передней панели, которые исполняются во время работы программы **Custom Run-Time Shortcut Menu**.

Два слова о нововведениях в функциях анализа и обработки сигналов. Их общее число в новой версии LabVIEW превышает 500. Введено 50 расширенных функций анализа и 75 новых математических функций. Иконки функций имеют однотипное оформление: в верхней полоске с желтым фоном размещается пиктограмма класса функций, в нижней полоске с белым фоном - пиктограмма самой функции. Несомненный интерес вызывают реализация двумерных функций цифровой обработки сигнала (преобразования Фурье, свертки, авто- и кросскорреляции) для действительных и комплексных величин, подпалитра функций преобразования координат, новые типы окон, непрерывных и дискретных распределений вероятностей. В подпалитре функций аппроксимации появились новые линейные параметрические модели, реализованы устойчивые алгоритмы сглаживания и новые инструменты оценки качества аппроксимации.

Наш очередной урок заканчивается. Очень кратко рассмотренные особенности новой версии - это всего лишь часть того многообразия новых технологий, инструментов и элементов, введенных в LabVIEW 8. С новинками 8-й версии мы продолжим знакомство и на следующих уроках.

Материал подготовлен преподавателем высшей категории Литвин С.Ю., Колледж информационных систем и технологий КНЭУ (Киев).

В уроке №10 использованы методические указания сервис-инженеров NI и материалы международной конференции "Образовательные, научные и инженерные приложения в среде LabVIEW и технологии National Instruments" (Москва, Россия, 2005).