КИРКОРОВ С. И.

НОВАЯ БИБЛИОТЕКА OEM КАК БАЗА ДЛЯ ДОВЕРЕННЫХ ПЛАТФОРМ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ADA В WIN32

МИНСК 2010

УДК 681.3

Киркоров С. И.

НОВАЯ БИБЛИОТЕКА OEM КАК БАЗА ДЛЯ ДОВЕРЕННЫХ ПЛАТФОРМ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ADA В WIN32

Аннотация:

Новая библиотека OEM как база для доверенных платформ программирования на языке Ada в Win32.
/ Киркоров С.И. //

Библиотека OEM – новый продукт вобравший в себя опыт и средства многих разработчиков, в том числе и автора свободно доступен высшим учебным заведениям бесплатно использовать основные компоненты технологии GNAT Pro (без каких-либо ограничений по функциональности) в образовательных и научно-исследовательских целях

GAP (GNAT Academic Program) – программа, предоставляющая возможность высшим учебным заведениям бесплатно использовать основные компоненты технологии GNAT Pro.

New library OEM as base for the entrusted platforms of programming in language Ada in Win32.
/ Kirkorov S.I.//

Library OEM - a new product incorporated experience and means of many developers including the author it is freely accessible to higher educational institutions free of charge to use the basic components of technology GNAT Pro (without any restrictions on functionality) in the educational and research purposes.

GAP (GNAT Academic Program) - the program giving possibility to higher educational institutions free of charge to use the basic components of technology GNAT Pro.

КИРКОРОВ Сергей Иванович – вед. специалист УП МедиаСкан, г. Минск, РБ.
Научные интересы:
- информационные технологии в образовании, компьютерное моделирование;
- .безопасность информационных систем и технологий.
Почтовый адрес: 220141, Минск-141, Аб/ящик №111.
Адрес электронной почты: ksiby@mediascan.by

СОДЕРЖАНИЕ

1. Введение 5
Проект Ada-2010_OEM 5
2. Подготовка к работе 10
Установка в системе среды разработки и библиотек 10
Первый запуск графической интегрированной среды разработки GPS 10
Создаем первый проект 11
Подключение библиотеки OEM 20
Выводим сообщения на русском языке на консоль. 22
Совместное использование консоли и графических окон 24
3. Пакет GWindows библиотеки OEM. 25
Обзор обучающих примеров. 25
Обучающая программа №1 – пакет OEM.GWindows.Message_Boxes. 27
Обучающая программа №2 – GUI программа. 28
Обучающая программа №3 – Динамическое создание ОКОН. 29
Обучающая программа №4 – Обработчик событий. 30
Обучающая программа №5 – Управляющие элементы. 38
Обучающая программа №6 – Размещаем ВСЁ или OEM.GWindows.Scroll_Panels. 39
Обучающая программа №7 – Рисование или OEM.GWindows.Drawing_Panels. 41
Обучающая программа №8 – Рисование с обработчиком событий. 43
Обучающая программа №9 – Рисующие объекты или Drawing Objects. 44
Обучающая программа №10 – Печать. 46
Обучающая программа №11 – Меню. 46
Обучающая программа №12 – Шрифты Windows. 49
Обучающая программа №13 – Z упорядочивание. 49
Обучающая программа №14 – Родительское ОКНО. 50
Обучающая программа №15 – Захват управления от МЫШИ. 51
Обучающая программа №16 – Изменение КУРСОРА. 52
Обучающая программа №17 – Создание ДИАЛОГА. 53
Обучающая программа №18 – Доступ к базе данных через ADO. 53
Обучающая программа №19 – Готовый управляющий элемент для БД. 54
Обучающая программа №20 – Упаковка и стыковка ОКОН. 56
Обучающая программа №21 – Воспроизведение звуков. 58
Тестовые примеры GWindows библиотеки OEM. 59
ЛИТЕРАТУРА 64

Киркоров С. И.

1. Введение

Слабым звеном в обеспечении информационной безопасности и надежности прикладного программного обеспечения (ПО), наряду с другими факторами, является использование не контролируемых, закрытых и не сертифицированных по требованиям безопасности информационных технологий (ИТ). Уровень доверия ИТ определяется самым слабым звеном в цепи средств обеспечения информационной безопасности. Поэтому перспективным и востребованным ныне подходом к проблеме формирования собственных доверенных сред разработки ПО является использование инструментальных программных средств с открытым кодом. При этом задачу обеспечения безопасности в ИТ в значительной степени решает наличие средств автоматизации верификации и формального доказательства правильности алгоритмов программы.

Проводившийся автором в течение ряда лет мониторинг качества доступных сред программирования показывает, что разработка программных средств в среде GNAT на языке программирования Ada [1] и применение инструментария SPARK для формального доказательства правильности алгоритмов программы обеспечивают создание программных средств критического назначения соответствующих ISO/IEC 15408-1 с уровнем гарантий EAL5 и выше (в Беларуси СТБ 34.101.1, в России ГОСТ Р ИСО/МЭК 15408).

Однако создание ПО критического назначения (к примеру, для современных медицинских, ядерных и космических технологий) всегда сопровождается разработкой большого объёма кода, который по принятой сейчас терминологии относится к классу бизнес-приложений. Он тоже требует обеспечения высокой согласованности функциональности, защищенности, надёжности (хотя, как правило, формально не верифицируется), а также совместимости по технологии производства с основной продукцией критического назначения.

Имеются в виду потребители программных средств для операционной системы MS Windows, которые требуют определённого, хотя не самого высокого уровня гарантий информационной безопасности. Для них библиотека OEM for Windows дает возможность объединить среду разработки GNAT GPL 2010 на языках программирования Ada-95, Ada-2005, Ada-2012 и существующие технологии для применений в операционной системе Windows от фирмы Microsoft. Библиотека OEM – разумный компромисс между надёжностью, безопасностью, с одной стороны, и простотой применения, разработки, низкой стоимости современных ИТ на базе семейства операционных систем (ОС) MS Windows, с другой стороны.

Библиотека OEM прошла всестороннее тестирование. Некоторые тесты можно рассматривать, как образцы рационального использования её средств. Они были положены в основу примеров, которые поставляются вместе с библиотекой.

Проект Ada-2010_OEM

По мнению автора, чем меньше требуется промежуточного, стороннего не доверенного слоя ПО для функционирования конечной прикладной системы, тем более высокий уровень гарантий можно обеспечить. Напротив, промежуточные средства, неподконтрольные разработчикам прикладного ПО (типичные для услуг, предоставляемых популярными серверами), снижают такой уровень. Подтверждением этого тезиса может служить, например, оговорка в лицензии на ОС MS Windows 2000, которую фирма Sun Microsystems Inc письменно обязала сделать компанию Microsoft (рис. 1).

: ЗАМЕЧАНИЕ ПО ПОДДЕРЖКЕ JAVA

ДАНОЕ ПРОГРАММНОЕ ИЗДЕЛИЕ МОЖЕТ СОДЕРЖАТЬ ПОДДЕРЖКУ ПРОГРАММ, НАПИСАННЫХ НА JAVA .
ТЕХНОЛОГИЯ JAVA - НЕ УСТОЙЧИВАЯ К СБОЯМ И НЕ РАЗРАБОТАНА, ИЗГОТОВЛЕНА, ИЛИ ПРЕДНАЗНАЧЕНА ДЛЯ ИСПОЛЬЗОВАНИЯ ИЛИ ПЕРЕПРОДАЖИ КАК ИНТЕРАКТИВНОЕ ОБОРУДОВАНИЕ УПРАВЛЕНИЯ В ОПАСНЫХ СРЕДАХ, ТРЕБУЮЩИХ ОТКАЗОУСТОЙЧИВОЙ РАБОТЫ, ТАКИХ КАК СИСТЕМЫ УПРАВЛЕНИЯ ЯДЕРНЫМ ОБОРУДОВАНИЕМ, СИСТЕМЫ НАВИГАЦИИ САМОЛЕТА ИЛИ СИСТЕМЫ СВЯЗИ, СИСТЕМЫ УПРАВЛЕНИЯ ВОЗДУШНЫМ ДВИЖЕНИЕМ, МАШИНЫ ПОДДЕРЖАНИЯ ЖИЗНЕОБЕСПЕЧЕНИЯ ИЛИ ОРУЖЕЙНЫЕ СИСТЕМЫ, В КОТОРЫХ СБОЙ В ТЕХНОЛОГИИ JAVA МОЖЕТ ПРИВЕСТИ НЕПОСРЕДСТВЕННО К СМЕРТИ, ТЕЛЕСНОМУ ПОВРЕЖДЕНИЮ, ИЛИ СЕРЬЕЗНОМУ ФИЗИЧЕСКОМУ ИЛИ ЭКОЛОГИЧЕСКОМУ УЩЕРБУ.

Рис. 1 Sun Microsystems, Inc письменно обязал Microsoft делать эту оговорку

Интересная ситуация сложилась с повсеместным использованием программной платформы Microsoft .NET Framework и Microsoft Visual Studio 2008 и более поздних версий. Практически все программные продукты Microsoft от серверной (например MS SQL Server 2005) до прикладных и клиентских приложений обеспечивают свой функционал при обязательной установке Microsoft .NET Framework. Платформа разработки Microsoft Visual Studio 2008 позволяет выполнять так называемую «AGILITY» разработку и более того провоцирует программистов к такому подходу. Последняя эффективна в бизнесе для выманивания финансовых средств у потребителя. В результате потребитель-заказчик получает «вечно» недоделанную программу или комплекс программ – «программу сегодня на сегодня». Разработчики 80% своего времени проводят за отладкой программы в отладчике, а потребитель только и успевает ставить «сервиспаки» и новые версии. При этом на каждой бесконечной стадии цикла разработки можно наблюдать картину как в общеизвестном шарже «качели» родившемся на свет лет 35 назад – (рис. 2). Цикл заканчивается, когда финансирование проекта прекращается, а потребитель работает на системе с огромным количеством «костылей».

Мало что дает такой подход и при подготовке программистов. Фактически, программиста, прекрасно знающего прежнюю версию Microsoft Visual Studio приходится в корне переучивать (ещё в большей степени это относится к ПО фирмы BORLAND). А то и заменять на более молодого выпускника ВУЗа! Тот начинал с изучения недавней последней версии и не тянет за собой не нужный старый багаж узкого специалиста. Последний подход экономически более выгоден. Чаще всего так и поступают руководители, но очень скоро ситуация повторяется!

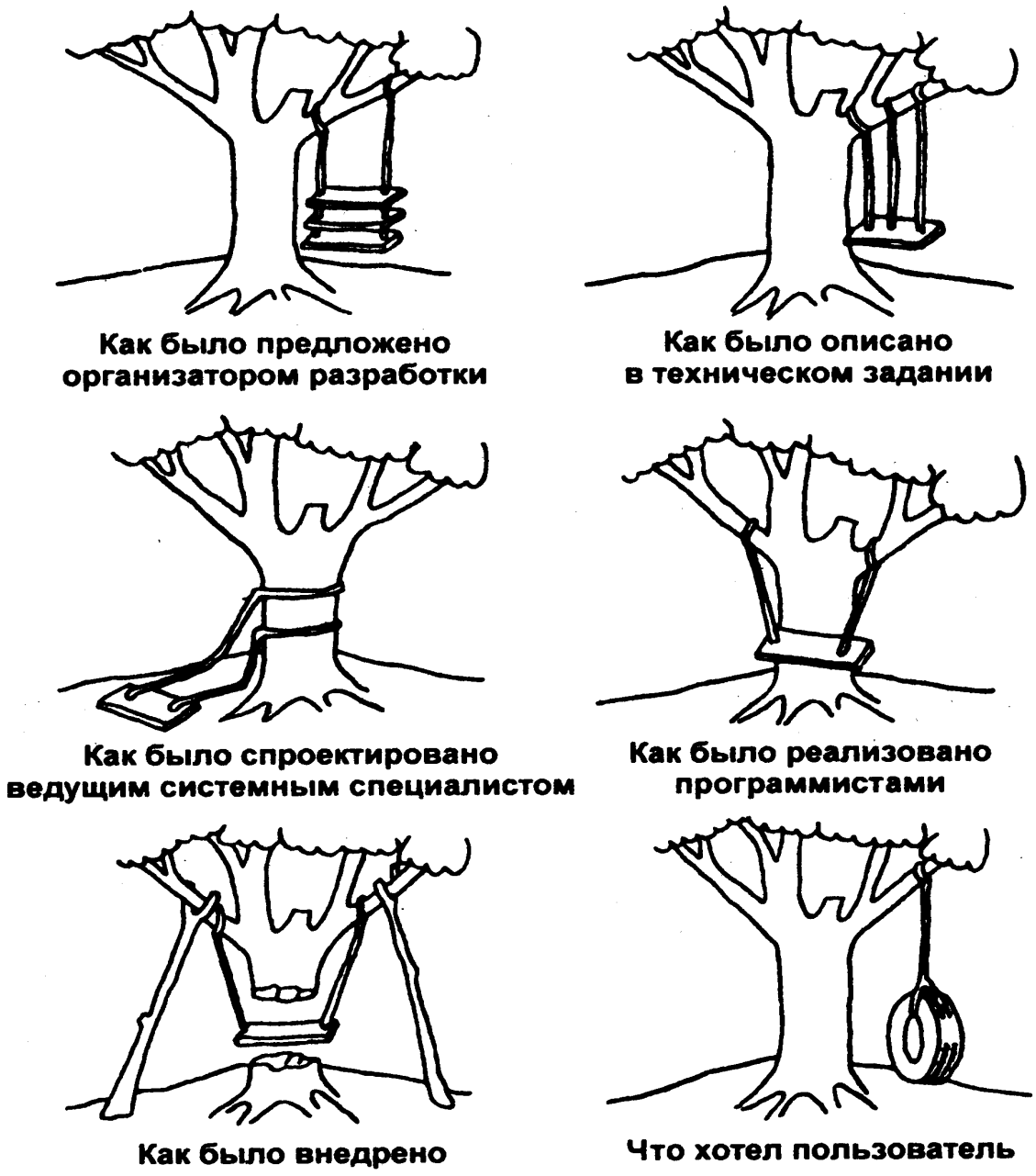

Рис. 2 Качели

Серьёзную проблему для безопасности ИТ в дальней перспективе, по мнению автора, представляет платформа Microsoft **.**NET Framework в связи её повсеместным применением. В основе её лежит «виртуальный компилятор». Программа при каждом запуске транслирует виртуальный код из файла в исполняемый код аппаратной платформы с помощью **.**NET Framework. Только после этого ОС загружает фактически не контролируемый код для выполнения в оперативную память. Разработчик программы может доказать правильность работы только виртуального кода программы на «своем рабочем месте». Фактически **.**NET Framework может выступать как «ГЛОБАЛЬНЫЙ ВЫКЛЮЧАТЕЛЬ» при необходимости. Злоумышленник может произвести «обновление платформы» или активировать уже существующую программную «закладку» с целью повысить уязвимость

платформы .NET Framework (никто не дает гарантий что таких «закладок» нет, а потребителю не будет возмещен урон от результата её применения – так написано в лицензии!!!).

Альтернативой на основных индустриальных платформах может служить современная среда разработки программ на языке Ada – GNAT Pro [2]. Она позволяет разрабатывать как программные системы, полностью реализованные на языке Ada, так и создавать многоязыковые системы. Технология присутствует на основных индустриальных платформах (включая MS Windows, различные версии UNIX и GNU/Linux, Mac OS, HP-UX), она включает кросс-компиляторы для популярных встроенных архитектур. GNAT Pro распространяется с исходными кодами всех своих компонент. Важным аспектом технологии является постоянная техническая поддержка клиентов (в том числе – бесплатная для университетв, участвующих в программе GAP (GNAT Academic Program) [3]).

GAP – это программа, предоставляющая возможность высшим учебным заведениям [3] бесплатно использовать основные компоненты технологии GNAT Pro (без каких-либо ограничений по функциональности) в образовательных и научно-исследовательских целях [4]. В этом случае используется версия GNAT GPL.

GNAT Pro удовлетворяет реальным требованиям индустриальной разработки программного обеспечения:

Разработка и сопровождение больших программных систем: GNAT Pro в состоянии работать с проектами, включающими тысячи модулей, предлагая при этом удобные средства структурирования и конфигурирования компонент разрабатываемой системы с поддержкой коллективной разработки.

Разработка многоязыковых систем: использование единой для всех компиляторов семейства GNU GCC технологии генерации объектного кода облегчает использование в рамках одной системы модулей, реализованных на разных языках (Ада, Си, Си++, Фортран). GNAT Pro поддерживает все средства межъязыкового взаимодействия, определяемые последней ревизией стандарта языка Ada (Ada 2005, в перспективе Ada-2012).

Высокая эффективность и производительность: эффективность объектного кода обеспечивается сочетанием технологий оптимизации кода, специфических для языка Ada, с мощными средствами оптимизации кодогенератора семейства компиляторов GNU GCC. При реализации библиотек поддержки периода исполнения особое внимание уделено оптимизации обработки исключительных ситуаций и межпроцессного взаимодействия.

Простота перехода на другую платформу: основные компоненты GNAT Pro имеют одинаковый интерфейс на всех платформах, на которых реализована технология, тем самым существенно упрощается перенос разработки с одной платформы на другую.

Эффективная поддержка со стороны разработчиков: неотъемлемой компонентой технологии GNAT Pro является комплекс услуг, включающий консультации по языку Ada и программным компонентам GNAT Pro, помощь в установке и настройке GNAT Pro, оперативное устранение замеченных пользователями дефектов технологии вплоть до поставки исправленных версий компонент GNAT Pro.

Накоплен богатый успешный опыт использования GNAT Pro в индустриальных проектах различного объема и сложности, включая разработку систем с повышенными требованиями к надежности, в том числе в проектах, выполняемых по государственным заказам в России.

В поиске инструментария позволяющего решить выше описанные проблемы как в образовании так и в научно-исследовательских целях, автором была выбрана система программирования GNAT GPL 2010 для Win32. Технология GNAT GPL включает в себя:

компилятор, полностью реализующий все официальные ревизии стандарта языка Ada (ISO 8652):

две интегрированных среды разработки: GPS (GNAT Programming Studio) и GNATbench (plug-in для Eclipse);

графический отладчик, позволяющий работать с программами, содержащими асинхронные процессы и осуществлять удаленную отладку встроенных приложений;

развитый инструментарий (включающий, в частности, монитор использования программного стека, средства автоматизации тестирования, систему подсчета метрик, средства анализа программного кода, средства контроля стиля кодирования и др.);

набор библиотек и программных интерфейсов, в частности: ASIS (интерфейс к синтаксису и семантике Ada-программ, основа для быстрой разработки инструментов статического анализа исходного текста), AWS (основа для разработки WEB-приложений), GtkAda (библиотека для создания графических пользовательских интерфейсов,Win32Ada (позволяет на уровне программного интерфейса Win32 взаимодействовать ОС MS Windows), XML/Ada (библиотека для построения XML-анализаторов).

Набор решений, облегчающих верификацию и сертификацию программных систем.

SPARK Pro – технологию, обеспечивающую доказательство корректности разрабатываемого кода.

Решения GNAT GPL 2010 платформа независимые, но пакет Win32Ada позволяет использовать специфические функции ОС на уровне программного интерфейса Win32 и взаимодействовать с ОС MS Windows учитывая особенности последней. Это, однако, требует от программиста глубоких знаний интерфейса ОС Win32. Пакет GNATCOM/GWINDOWS позволял формализовать и облегчить задачу для программиста, но совместим только с Ada-95 – не отвечает современным требованиям и сложен в установке.

Совместив достоинства этих двух пакетов, мы заложили основу библиотеки OEM – платформы для программирования на языке Ada-95/2005/2012 в GNAT GPL 2010 Win32 с возможностью локализации для русского языка.

Основные характеристики библиотеки OEM (по современному состоянию):

Интеграция со средой разработки GPS 4.3.1 (20090114);

Наличие средств перекодировки русского (OEM для консоли, ANSI (CP-1251) и Unicode-16 для графической консоли), а также BASE64 для электронной почты;

Собственный менеджер графических окон (исключительно на функциях Win32);

Собственный менеджер COM компонентов (исключительно на функциях Win32);

Собственные функции работы с USB устройствами;

Собственные функции работы по протоколу TCP/IP;

Собственные функции работы с СУБД по интерфейсу компонента ADO;

Средства генерации интерфейсов вызова зарегистрированных, создания новых COM объектов на языке Ada-2005/95/2012 (например OPS сервера для SCADA);

Платформа зависима от Win32.

Сформированная нами библиотека OEM – новый продукт, вобравший в себя опыт и средства многих разработчиков, в том числе и автора [5]. Он свободно доступен в образовании, позволяя бесплатно (и особенно удобно для участников GAP [3, 6]) использовать основные компоненты технологии GNAT Pro под Win32 (без каких-либо ограничений по функциональности) в образовательных и научно-исследовательских целях (http://www.mediascan.by/index.files/CD_Ada-2009_OEM.zip) [7].

С выходом GNAT GPL 2010 в июне 2010 года, разработчики выполнили адаптацию библиотеки OEM в части инсталляции и документации. Изменения коснулись в основном начала 2 главы.

2. Подготовка к работе

Глава "Подготовка к работе" написана в предположении, что читатель впервые решил изучить GNAT GPL 2010 for Windows. Если это не так, то можно пропустить весь или частично материал главы 2.

Установка в системе среды разработки и библиотек

Для установки комплекта среды разработки Вам понадобится минимум три файла со всего дистрибутива GNAT GPL 2010 for Windows, собственно библиотека OEM и файл архива примеров для этой книги – файл для установки собственно среды разработки (gnat-gpl-2010-i686-pc-mingw32-bin.exe 68514 Kbyte 08.06.2010 18:31), файл для установки библиотеки (win32ada-gpl-2010.exe 2606 Kbyte 19.01.2010 18:01), файл утилиты MAKE (gnumake-3.79.1-pentium-mingw32msv.exe 122 Kbyte 09.01.2006 19:22), файл архива библиотеки OEM (oem2010.zip 4815 Kbyte 15.07.2010 11:14), файл утилит и тестов (oemtools2010.zip 17764 Kbyte 15.07.2010 11:14) и файл архива примеров (oemtutorials-gui2010.zip 8796 Kbyte 15.07.2010 11:15). Естественно здесь указаны самые старые версии файлов. Весь полный дистрибутив GNAT GPL 2010 for Windows, имеет размер более 210 Mbyte. Дистрибутивы можно получить и по интернету с сайтов http://libre.adacore.com и http://www.mediascan.by , соответственно GNAT GPL 2010 for Windows и библиотеки OEM.

В первую очередь устанавливается дистрибутив GNAT GPL 2010 for Windows. Для этого просто запускается на выполнение файл " gnat-gpl-2010-i686-pc-mingw32-bin.exe " и далее следовать инструкции "мастера" установки. Необходимо установить среду разработки на диск "`C:\`". В результату на диске появится каталог GNAT с подкаталогом 2010 ("`C:\GNAT\2010\`"). Этот подкаталог является корнем для всех остальных.

Скопируйте файл "gnumake-3.79.1-pentium-mingw32msv.exe" в подкаталог "`C:\GNAT\2010\bin\`" под именем "make.exe".

У Вас должны быть права локального администратора. Сейчас желательно перезагрузить ОС, в этом случае гарантированно войдут в действия пути поиска программ назначенные при установке дистрибутива GNAT GPL 2010 for Windows.

Следующим шагом можно устанавливать библиотеки и другие ресурсы дистрибутива.

После установки GNAT GPL 2010 for Windows необходимо установить библиотеку "win32ada". Для этого запускается на выполнение файл " win32ada-gpl-2010.exe " и далее следовать инструкции "мастера" установки.

Теперь всё готово для установки библиотеки OEM. Выберите диск и каталог для установки. Например: "`C:\Root305\ProjKA\2010\`". Разархивируйте в него файл oem2010.zip, oemtools2010.zip и файл oemtutorials-gui2010.zip. Должны появиться три подкаталога: "`C:\Root305\ProjKA\2010\oem2010\`", "`C:\Root305\ProjKA\2010\oemtools2010\`" и "`C:\Root305\ProjKA\2010\oemtutorials-gui2010\`". Запустите скрипт "install.cmd" на выполнение из каталога `oem\`. После выполнения скрипта библиотека OEM установлена.

Первый запуск графической интегрированной среды разработки GPS

Зарегистрируйтесь в ОС под именем под которым Вы будете работать в GPS. Ваше имя должно быть обязательно на латинице. Это необходимо для формирования файлов состояния GPS в Вашем профиле ОС. GPS не распознает русскую кодировку в именах файлов и каталогов (это касается текущей версии GPS).

Запустите на выполнение файл GPS.EXE через меню или через иконку на рабочем столе графической консоли (рисунки приведены для версии GPS входящей в состав GNAT GPL 2009 for Windows, в данном случае это не влияет на понимание материала).

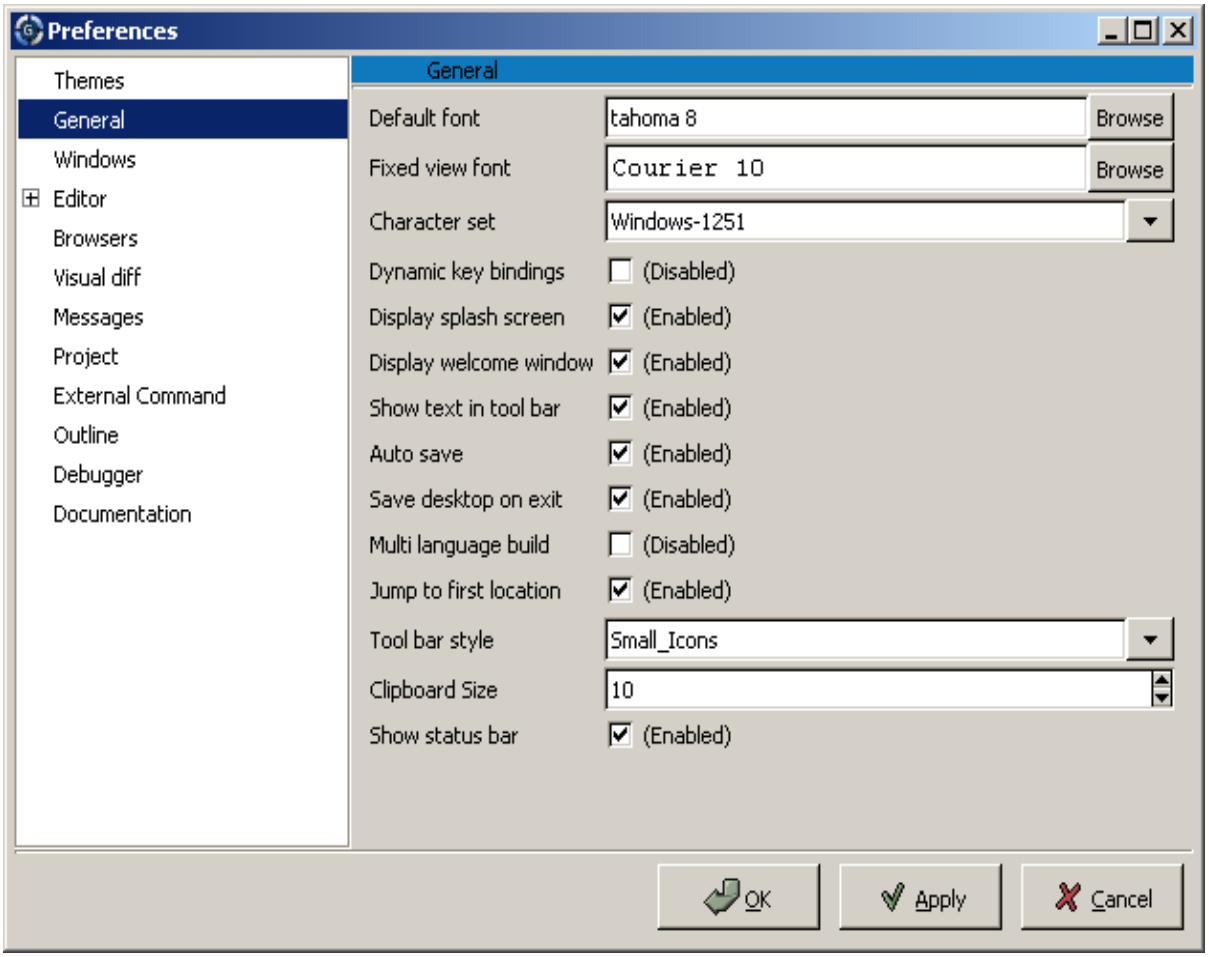

Рис. 3 Наберите вручную Windows-1251 в поле ввода Character set.

Наберите вручную Windows-1251 в поле ввода **Character set** при первом запуске GPS как показано на Рис 3 и выйдите из программы. После этого можно применять эту русскоязычную кодировку в комментариях и в строках.

Создаем первый проект

Вначале сразу желательно сразу определиться: где на диске вы будете создавать проект. Автор предпочитает для этой цели использовать отдельный подкаталог, который создается заранее и имеет смысловое значение определяющее время его создания по годам. Например: `C:\Root305\ProjKA\2010\` – проекты созданные в 2010 году. Ещё раз хочется обратить внимание на то, что все имена файлов и каталогов должны быть на латинице. Создадим каталог проекта и подкаталог для промежуточных файлов трансляции Рис. 4.

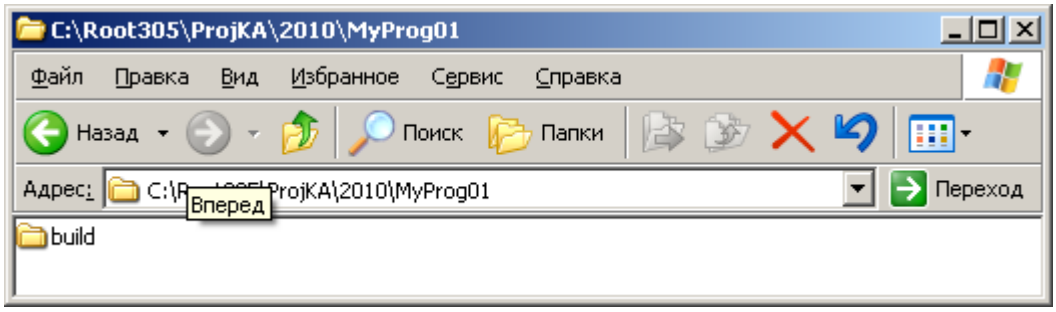

Рис. 4 Создадим каталог проекта

Запустим на выполнение GPS, должно появиться окно диалога Рис.5.

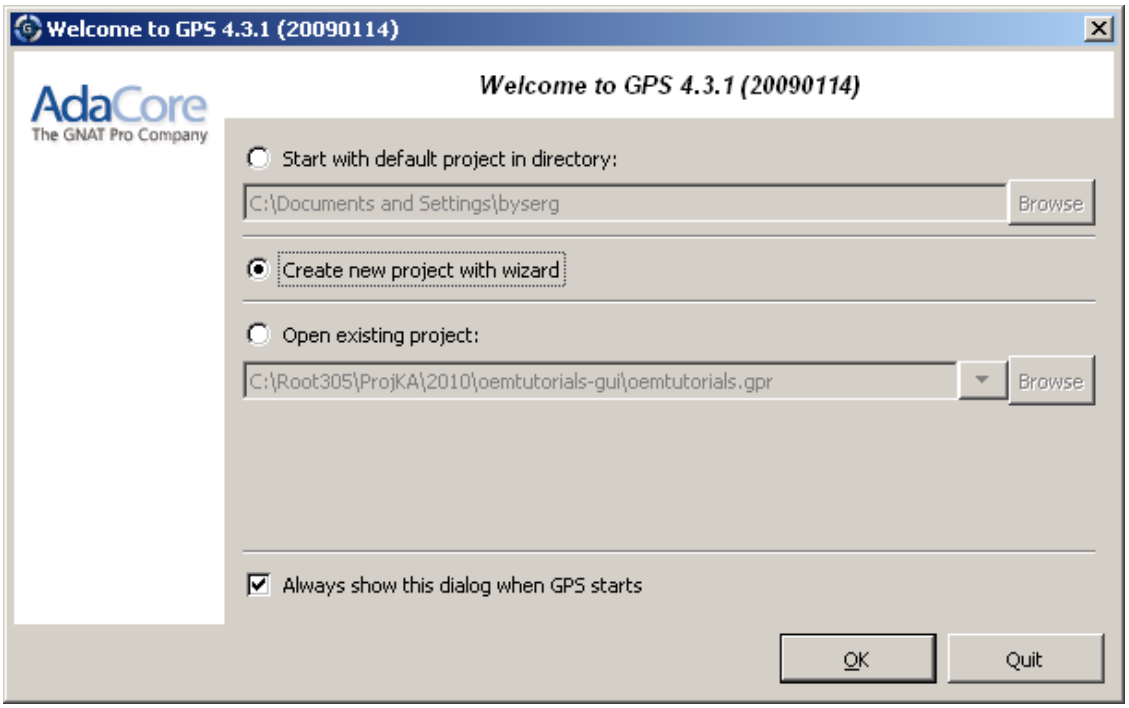

Рис. 5. Окно приветствия программы GPS

Выберем **Create new project with wizard** и нажмем кнопку **OK**. В следующем диалоге оставим всё не изменяя и нажмем кнопку **Forward** Рис. 6.

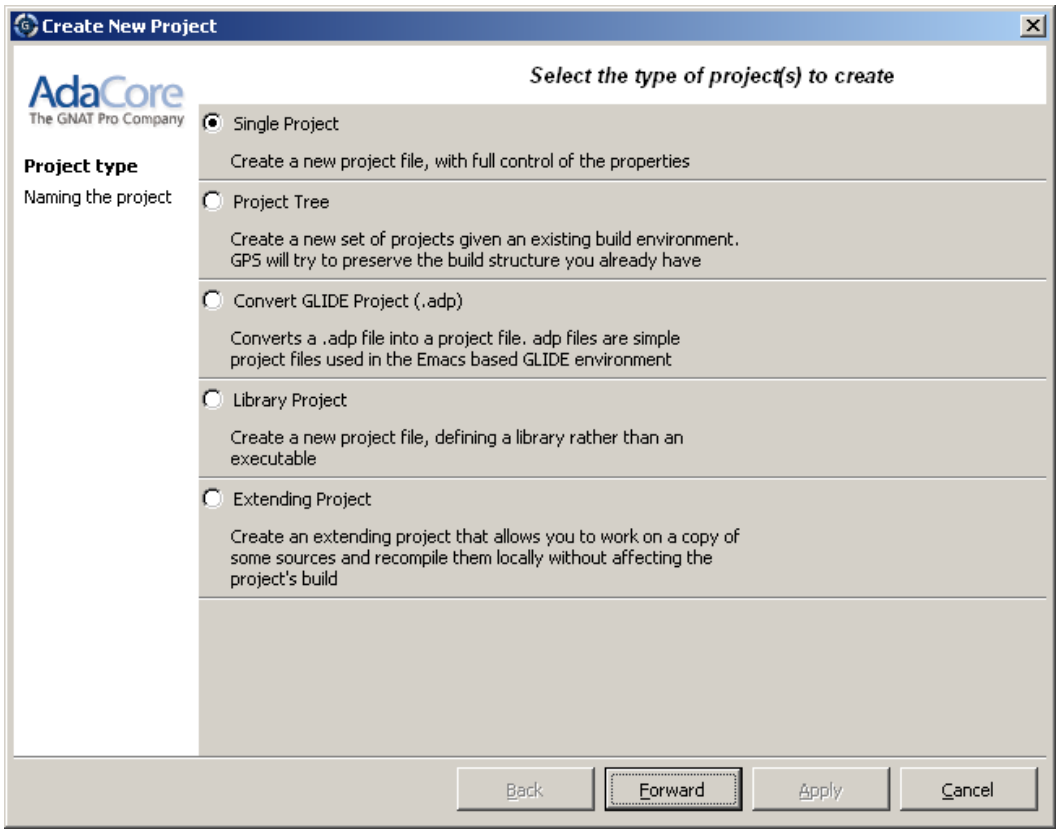

Рис. 6. Создадим Single Project

В следующем диалоговом окне определим имя проекта (имя проекта должно как правило совпадать с именем главной процедуры и файла где эта процедура будет описана), место расположения файла проекта Рис. 7.

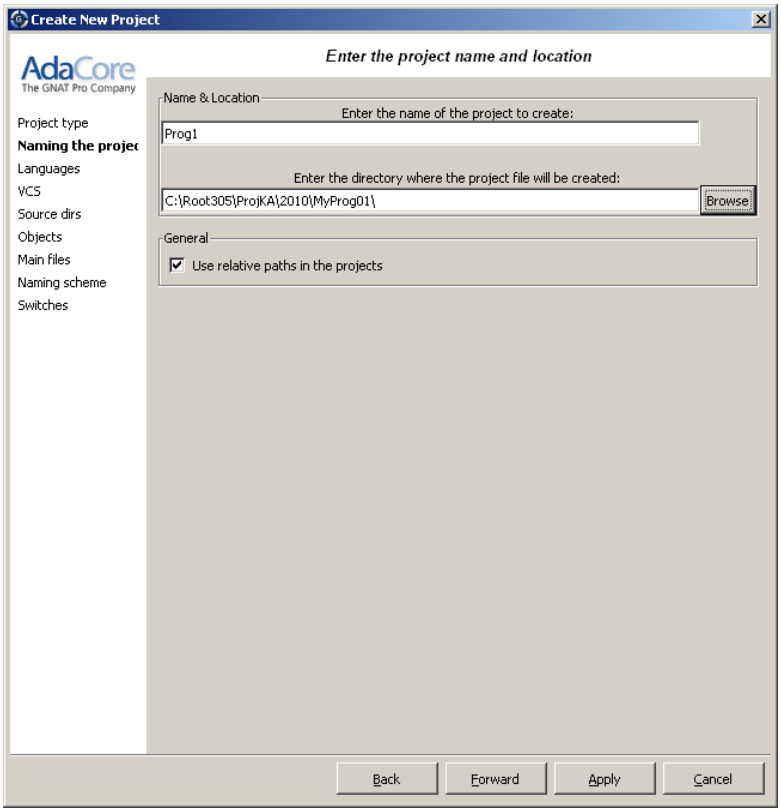

Рис. 7. Диалог выбора имени проекта и место его расположения

Кнопка **Browse** позволяет выбрать директорий проекта через диалоговое окно Рис. 8.

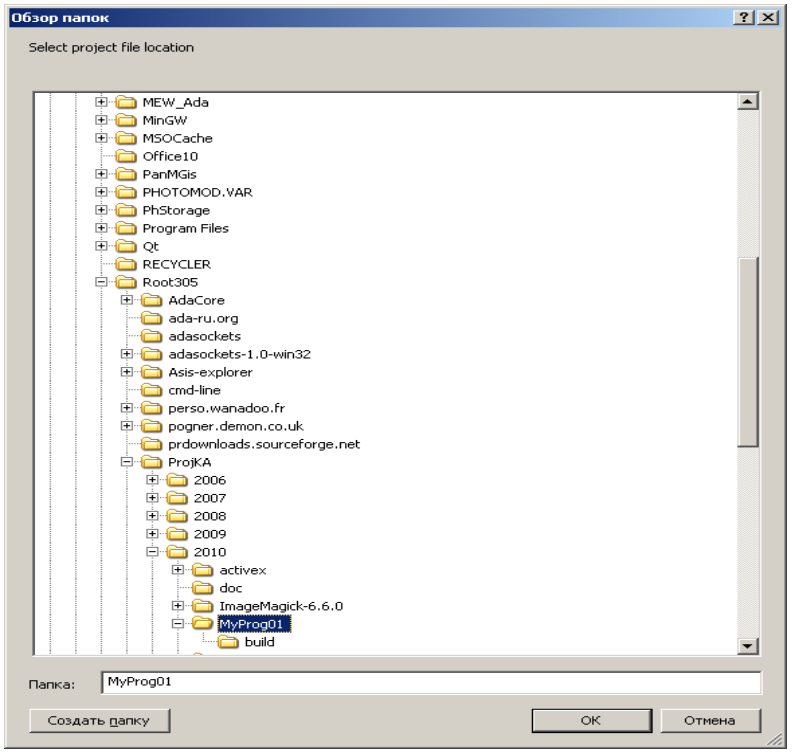

Рис. 8. Диалог выбора или создания директория

Нажмем кнопку **Forward** Рис. 7. В следующем диалоговом окне оставим всё без изменения и нажнем кнопку **Forward** Рис. 8.

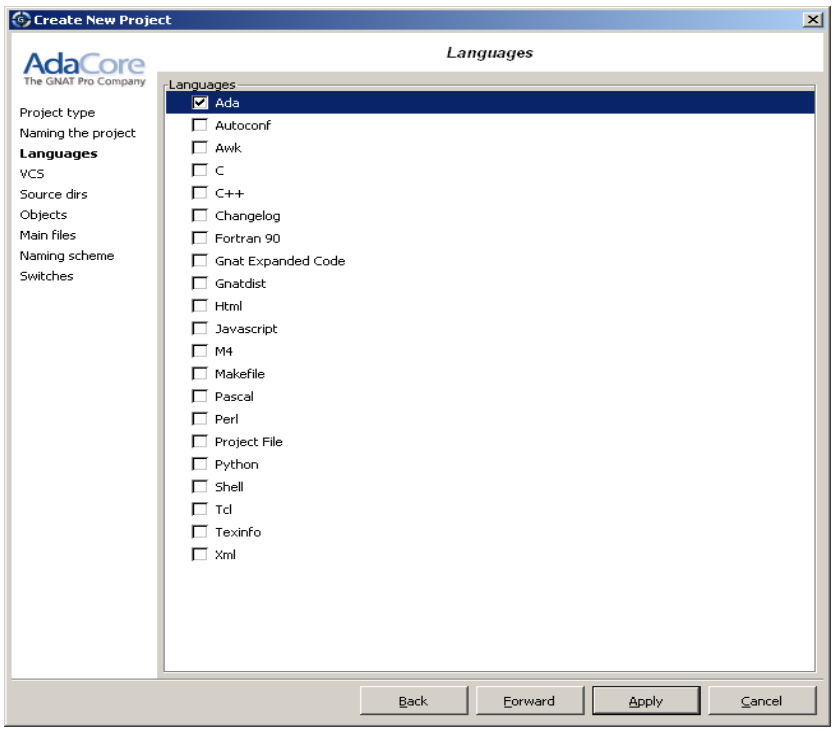

Рис. 9. Диалог выбора языков программирования в проекте

Следующих два окна диалогов также оставляем без изменения, последовательно нажимая кнопку **Forward**. Необходимо только убедиться в правильности выбора каталога размещения файлов программы (`C:\Root305\ProjKA\2010\MyProg01`) Рис. 10.

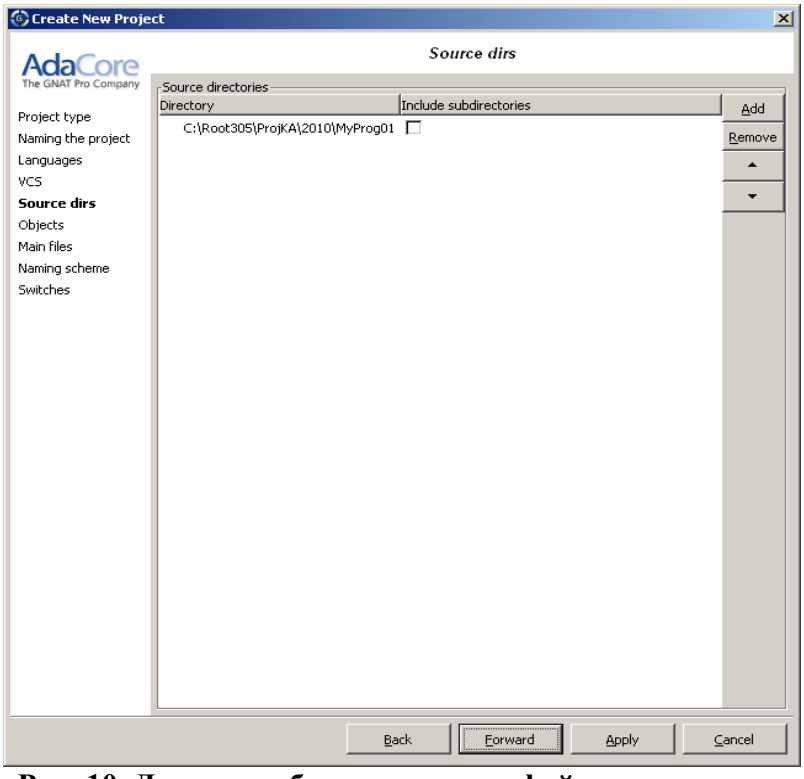

Рис. 10. Диалог выбора каталогов файлов программ.

На следующем шаге необходимо определить каталог для файлов трансляции и каталог для исполнения программы. Для этого, используя кнопку **Browse,** лучше переопределить каталоги по умолчанию Рис. 11,

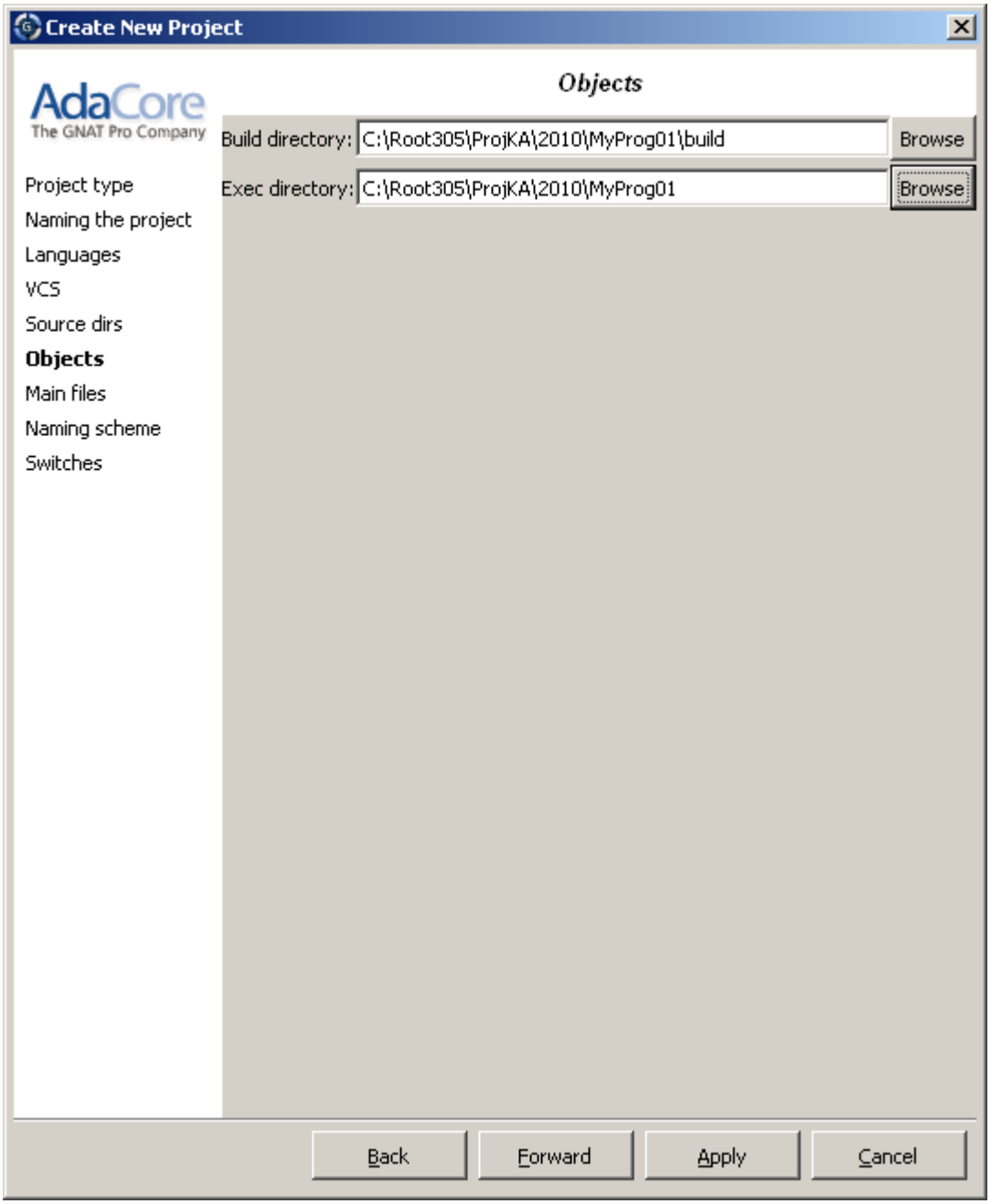

Рис. 11. Определим каталог Build и Exec

В следующем диалоговом окне определяем главный файл проекта. Для этого в диалоге **Main Files** нажимаем кнопку **Add**. В появившемся диалоге Select a File в поле Имя файла вводим с клавиатуры Prog1.adb. Рис. 12. Нажимаем кнопку открыть – должны получить результат как на Рис. 13. Необходимо понимать, что на данном этапе мы прописали только имя файла в проекте. Физически сам файл `Prog1.adb` не создан и в нашем случае не существует.

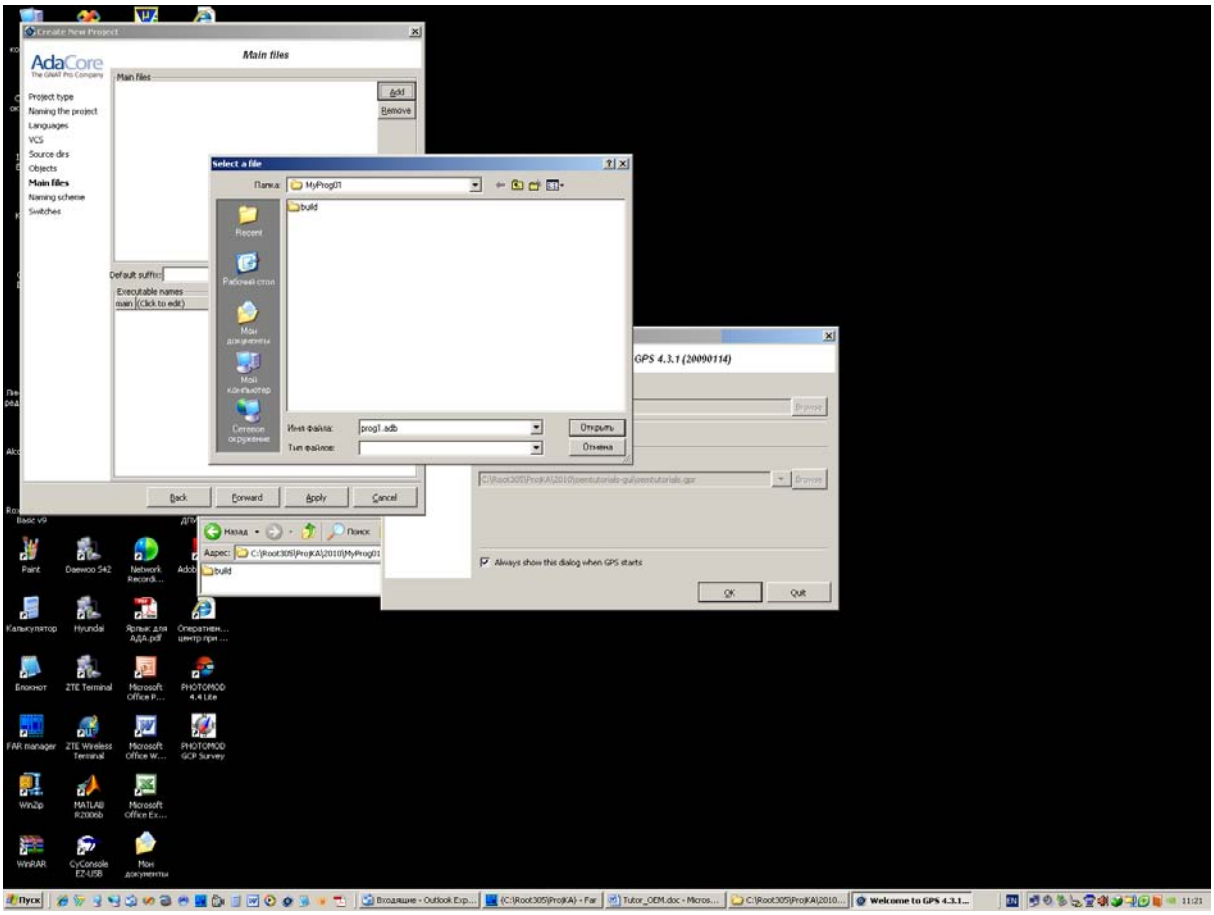

Рис. 12. Определение имени главного файла программы в проекте

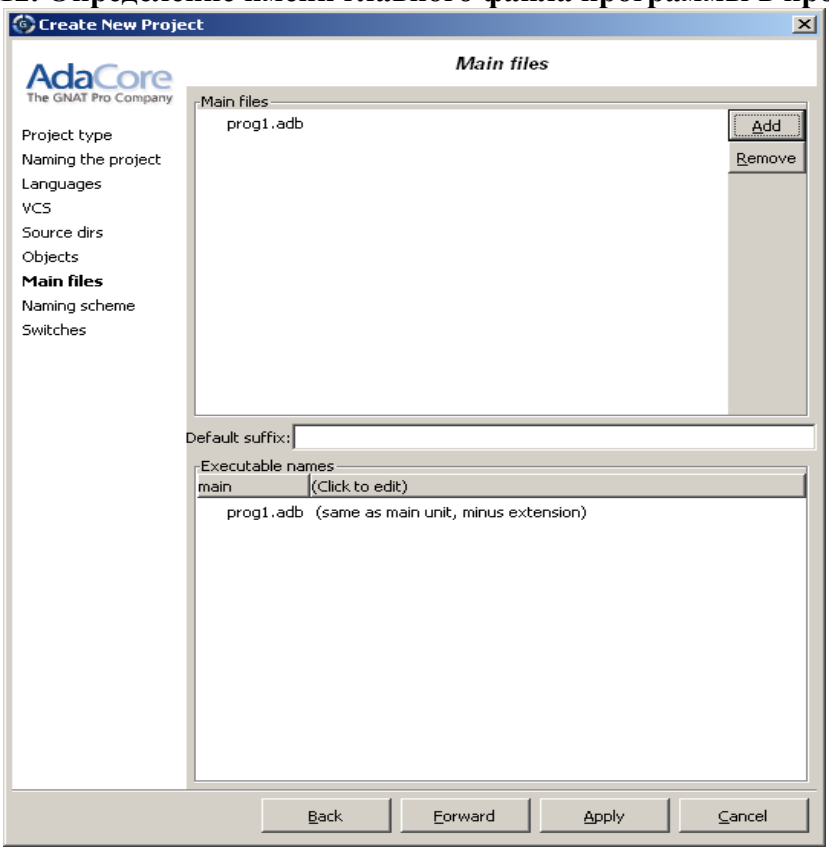

Рис. 13. Диалог определения главного файла программы в проекте

Следующее диалоговое окно определяет соглашения о именовании файлов программы для выбранного языка программирования и пакетов которые в неё входят. Оставляем всё по умолчанию и переходим к следующему завершающему диалогу. На данном этапе можно тоже оставить всё по умолчанию, поэтому нажимаем кнопку **Apply**.

Проект создан и загружен в GPS. В окне Message одно предупреждение об отсутствии файла программы на языке Ada (красным цветом) Рис.14.

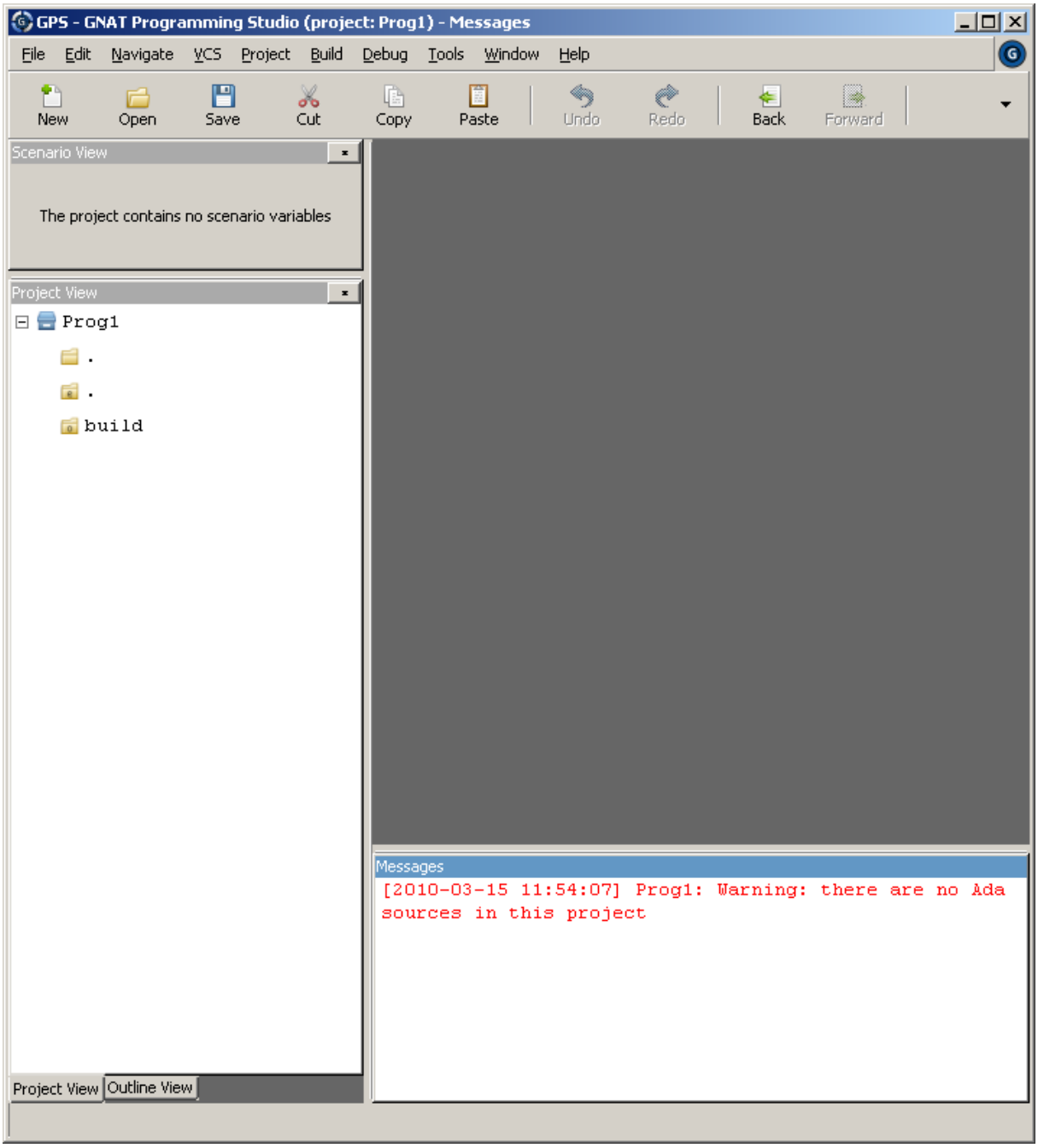

Рис. 14. Окно GNAT Programming Studio с предупреждением об отсутствии файла исходного текста программы в каталоге назначенном в проекте

Создадим файл программы, для этого нажмем последовательно кнопку New и Save. В поле **Имя файла** набираем `Prog1.adb` и нажимаем кнопку **Сохранить,** Рис 15.

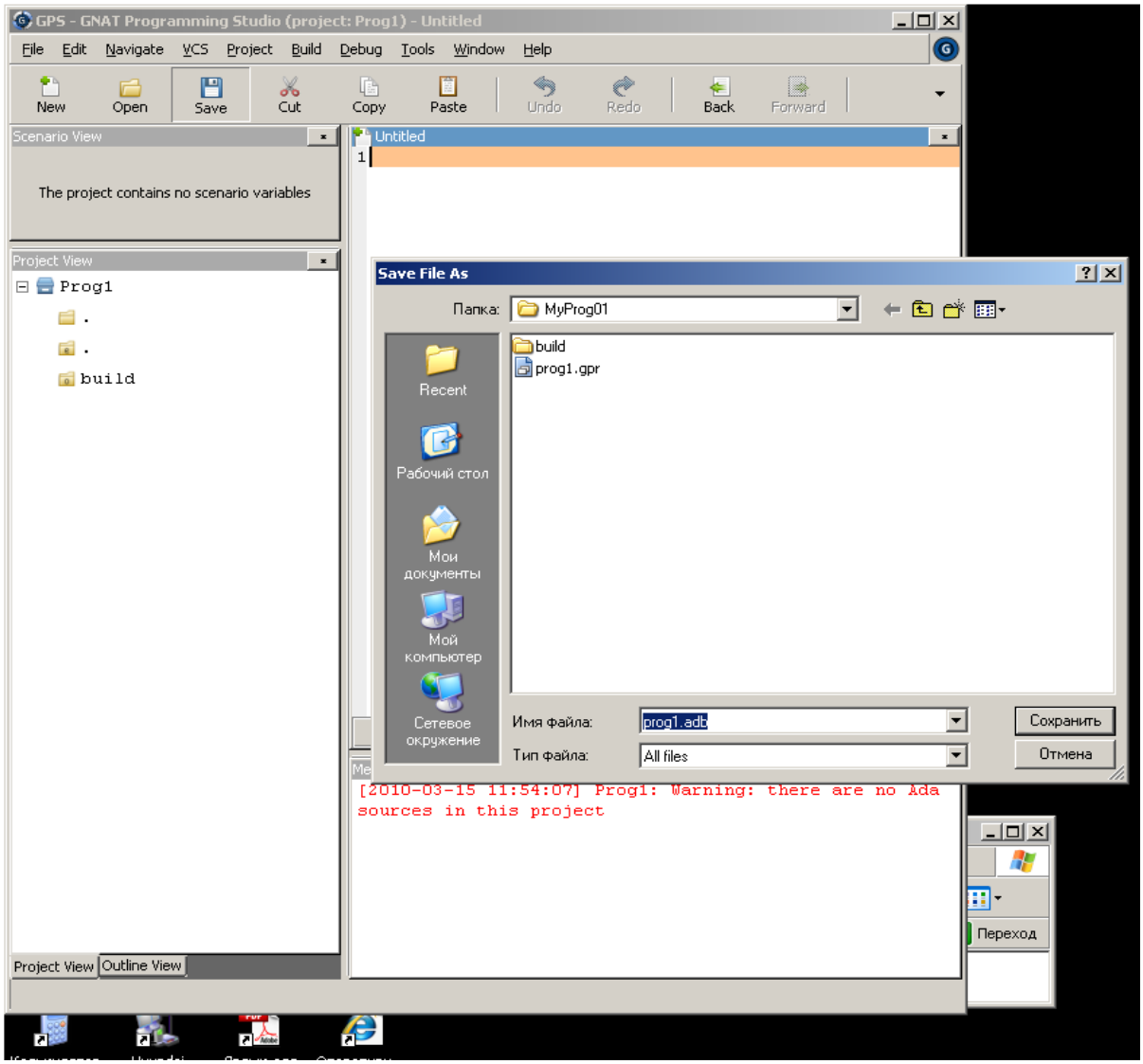

Рис. 15. Создание нового файла в проекте

Наберем текст программы в соответствующем окне.

```
With Ada.Text_IO;

procedure prog1 is
begin
   Ada.Text_IO.Put_Line("My first program");
end prog1;
```

Нажмем кнопку **Build Main**. В результате мы получим готовую программу к запуску `Prog1.exe` Рис. 16. и Рис. 17.

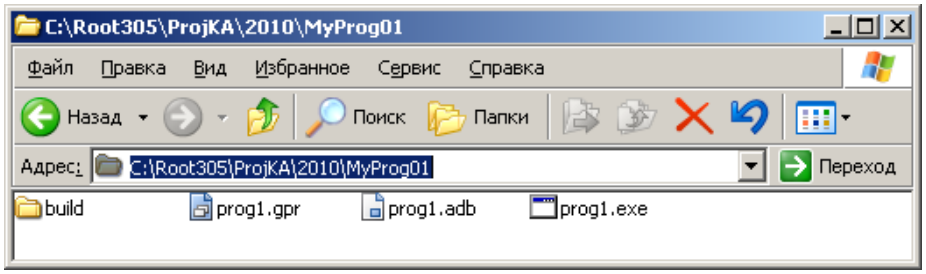

Рис. 16. Каталог проекта после трансляции

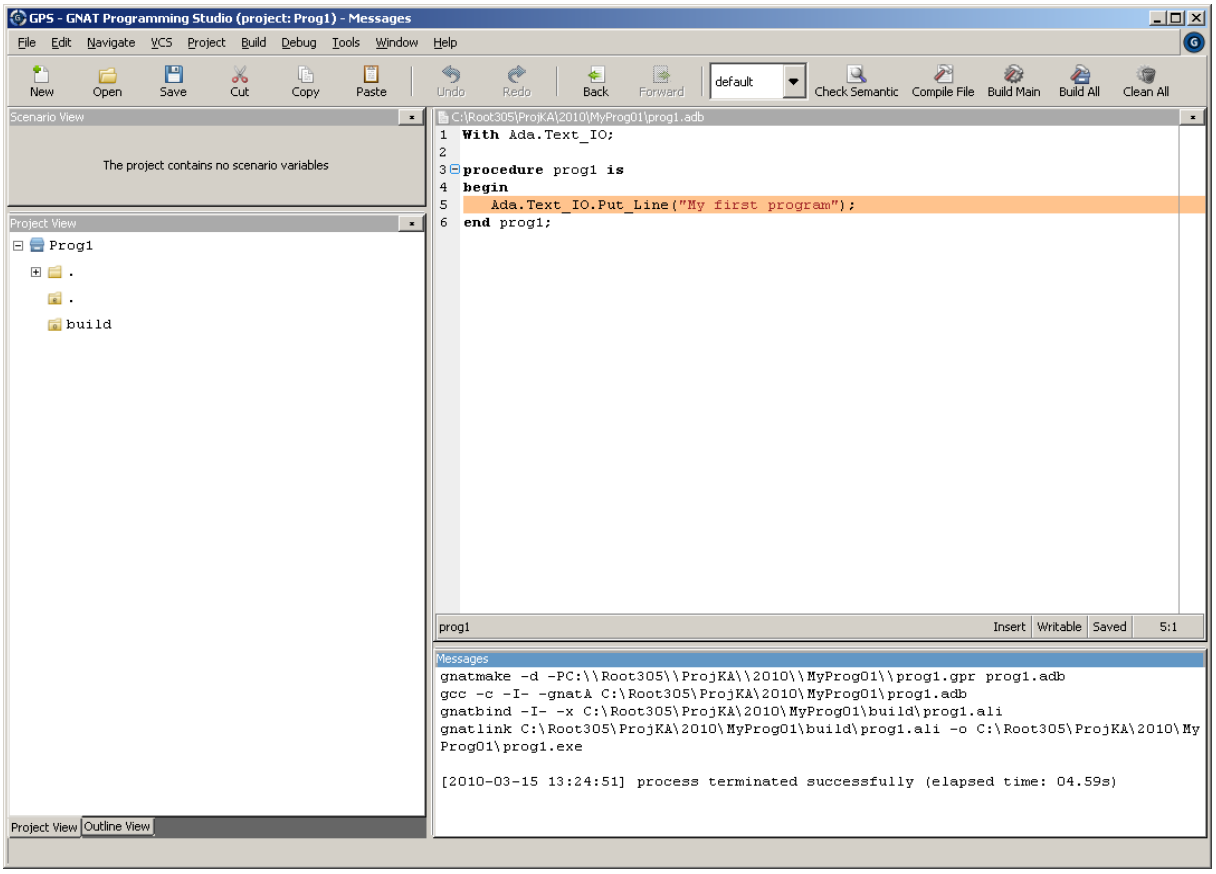

Рис. 17. Окно GPS после успешной трансляции программы

Запустим стандартную консоль CMD.EXE, перейдем в каталог проекта и запускаем программу `Prog1.exe`. Результат выполнения программы показан на Рис. 18.

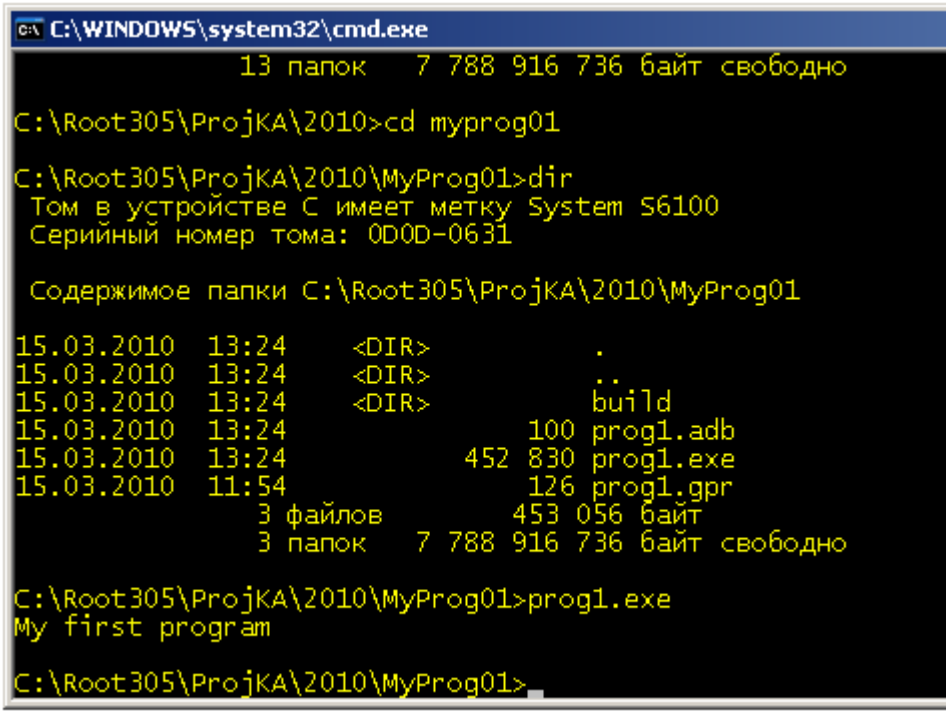

Рис. 18. Программа печатает на стандартной консоли "My first program"

Подключение библиотеки OEM

К существующему проекту подключить установленную библиотеку OEM достаточно просто. Откроем проект в режиме редактирования текста Рис. 19 и Рис.20.

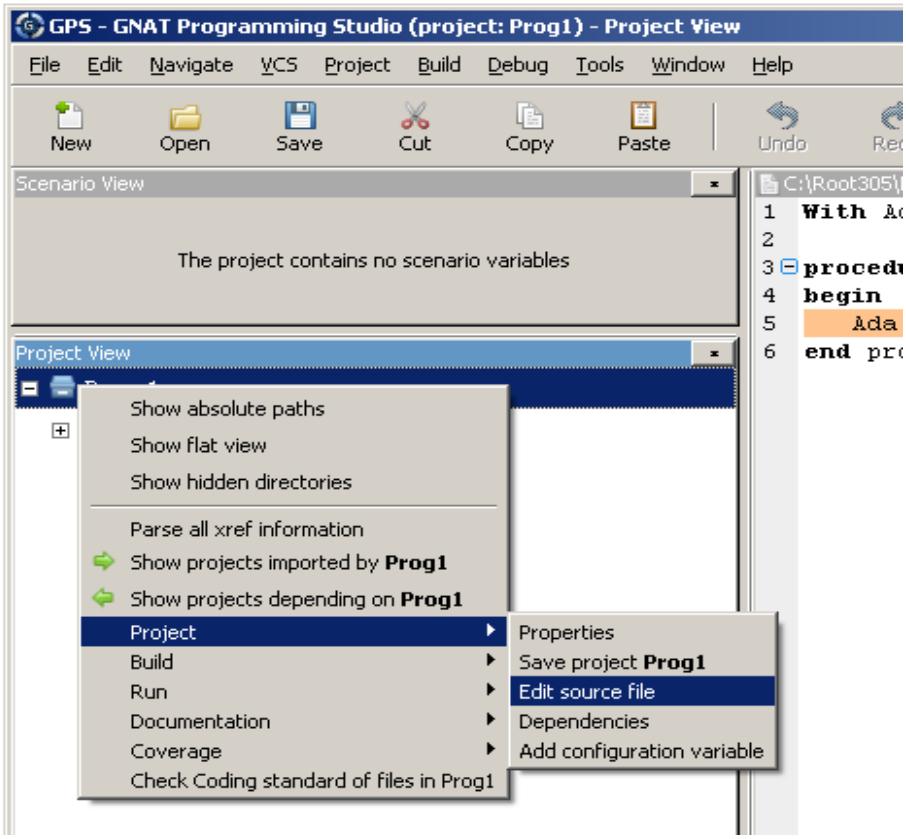

Рис. 19. Открытие файла проекта в режиме редактирования текста

Контекстное меню проекта вызывается путем нажатия правой клавиши мыши. Наберем в первой строке: ***with "OEM";*** .

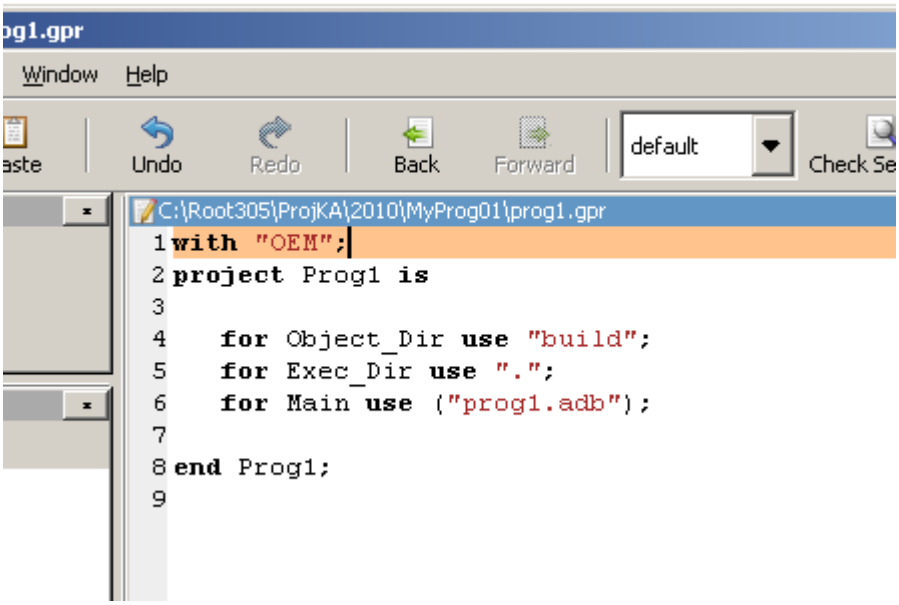

Рис. 20. Подключение в проект библиотеки OEM

Затем сохраняем файл проекта и перезагружаем его Рис. 21.

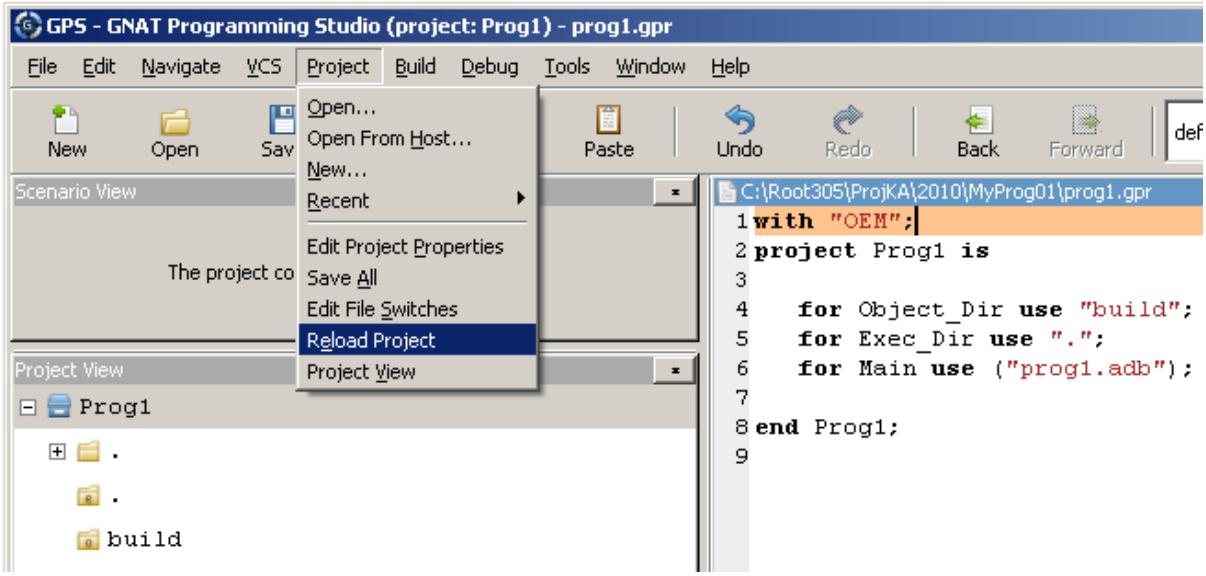

Рис. 21. Перезагрузка файла проекта через меню GPS

Файл проекта в редакторе GPS можно закрыть. Теперь в нашей программе можно использовать все пакеты библиотеки OEM. Рис. 22. показывает дерево проектов во вкладке Project View программы GPS.

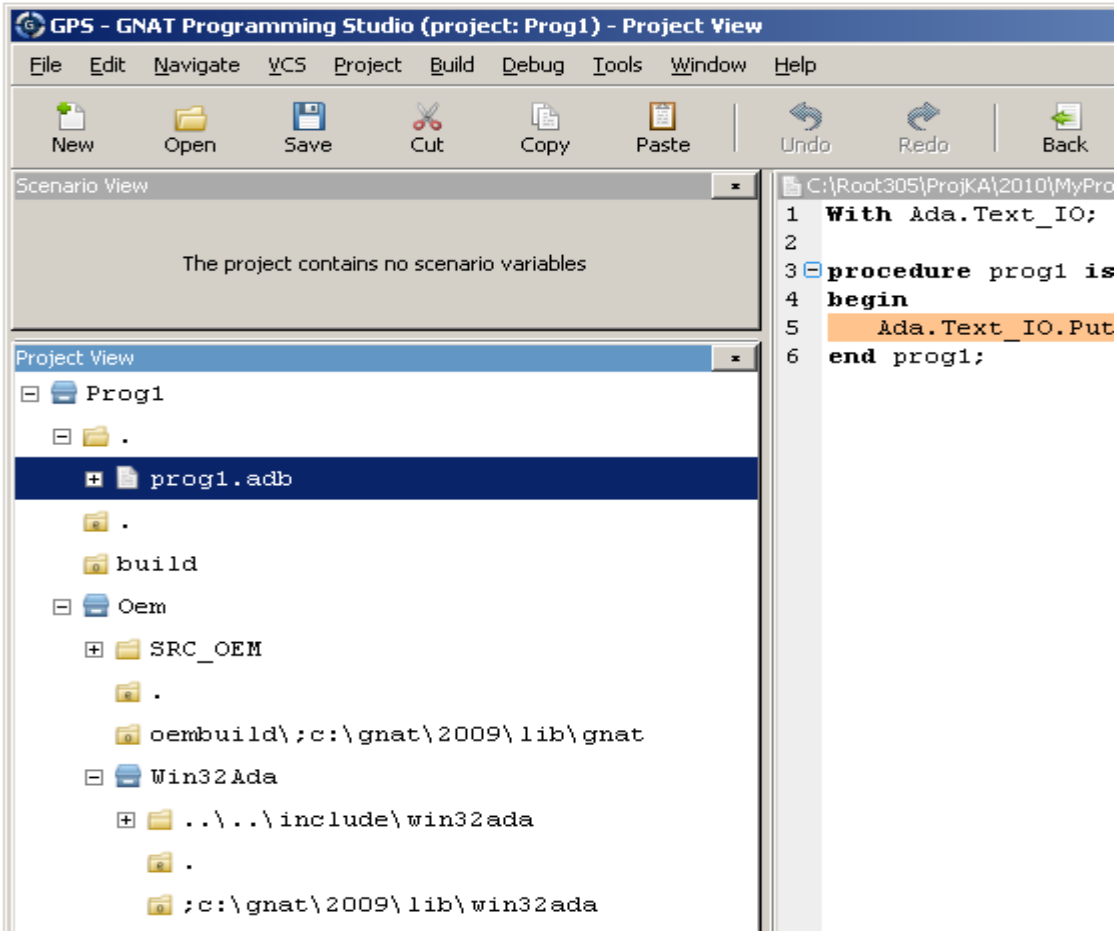

Рис. 22. Дерево проекта Prog1 с подключенной библиотекой OEM

Выводим сообщения на русском языке на консоль.

Необходимо убедиться что в GPS установлена нужная кодировка. Для этого вызываем через меню диалог **Preferences** Рис. 23. и убеждаемся что кодировка Windows-1251 установлена Рис. 3.

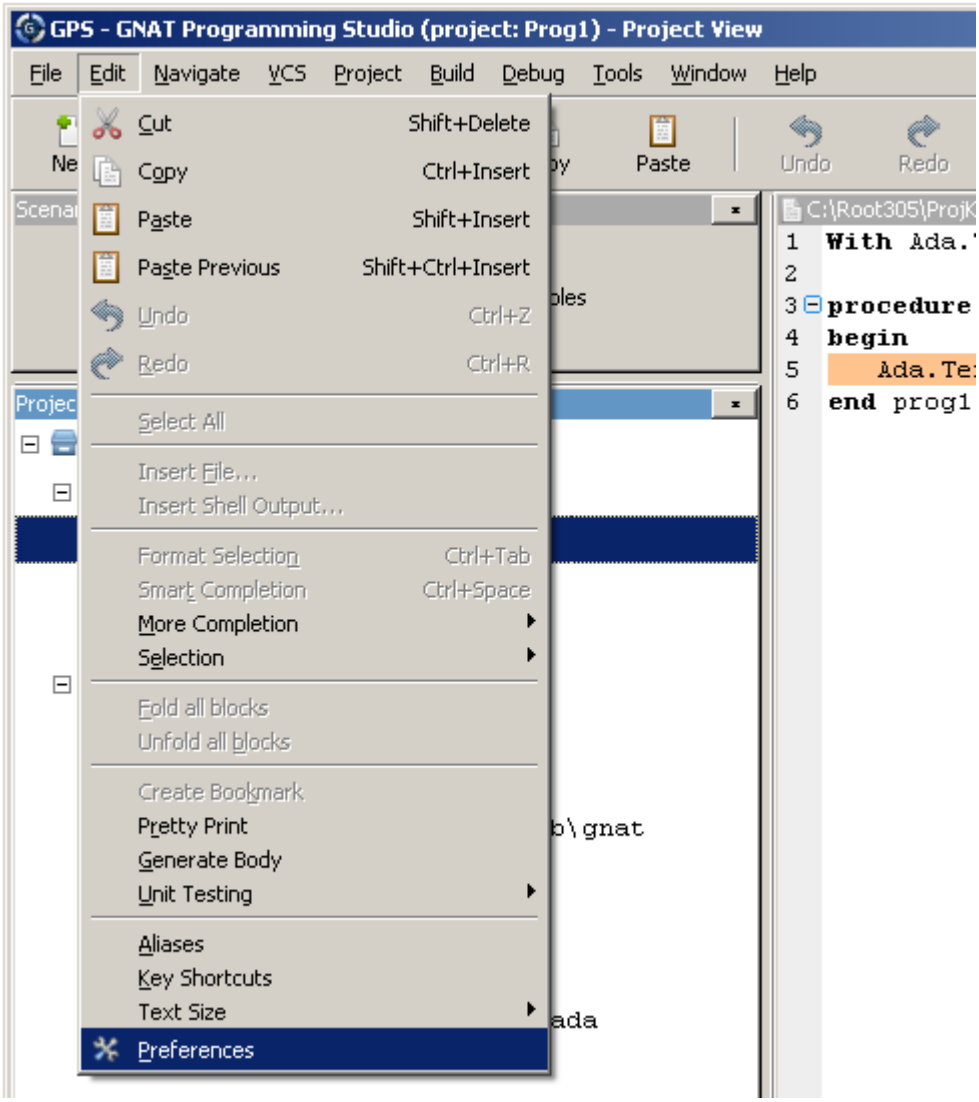

Рис. 23. Вызов из меню диалога Preferences

Если кодировка установлена, то можно смело переключаться на русскую раскладку и модифицировать текст программы. Нажимаем кнопку Build Main и запускаем её в консоли на выполнение Рис. 24., но прочитать сообщение невозможно.

```
With Ada.Text_IO;

procedure prog1 is
begin
   Ada.Text_IO.Put_Line("My first program - Это мая первая программа");
end prog1;
```

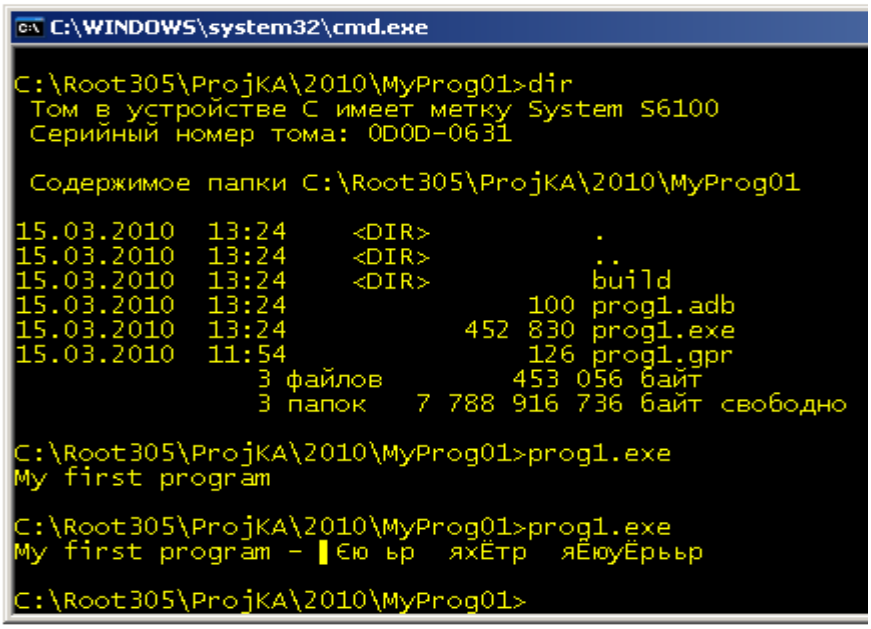

Рис. 24. Видно что прочитать сообщение на консоле невозможно

Дело в том что консоль работает в режиме совместимости с DOS программами и использует другую - OEM кодировку (иногда её ещё называют "ДОСовской") в отличии от "Графической КОНСОЛИ" оконной системы MS Windows.

Для того чтобы всё таки вывести на консоль текст на русском языке доработаем программу:

```
With Ada.Text_IO;
With OEM;

procedure prog1 is

   function SO(s : String) return String is
      sl : String := s;
   begin
      OEM.To_OEM(sl);
      return sl;
   end SO;

begin
   Ada.Text_IO.Put_Line(SO("My first program - Это первая программа"));
end prog1;
```

Теперь русский текст выводится правильно на консоль Рис. 25.

Примечание:

Для изучения синтаксиса языка программирования Ada можно почитать книгу Гавва А. Е. “Адское” программирование. Ada-95. Компилятор GNAT: [Электрон. ресурс]. – http://www.ada-ru.org. [1].

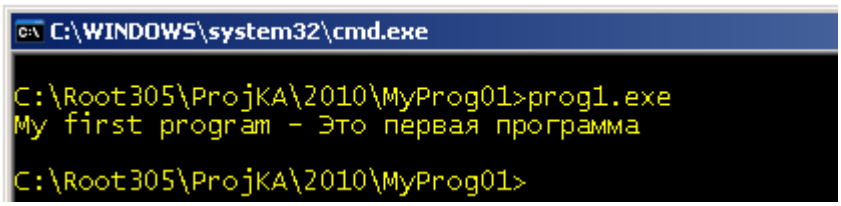

Рис. 25. Доработанная программы выводит текст на русском языке в читаемом виде

Совместное использование консоли и графических окон

Небольшая доработка программы позволяет выдавать сообщения в графическом окне:

```
with Ada.Text_IO;
with OEM.GWindows.GStrings;
with OEM.GWindows.Message_Boxes;

procedure prog1 is
   use OEM.GWindows.Message_Boxes;

   function SO(s : String) return String is
      sl : String := s;
   begin
      OEM.To_OEM(sl);
      return sl;
   end SO;

begin
   Ada.Text_IO.Put_Line(SO("My first program - Это первая программа"));
   Message_Box (Title =>
                  OEM.GWindows.GStrings.To_GString_From_String
                    ("Tutorial1 - учебный пример №1"),
                Text  => OEM.GWindows.GStrings.To_GString_From_String
                  ("My first program - Это первая программа"),
                Icon  => Exclamation_Icon);
end prog1;
```

На Рис. 26 показан результат работы программы. Более подробно пакет GWindows библиотеки OEM будет рассмотрен в главе 3.

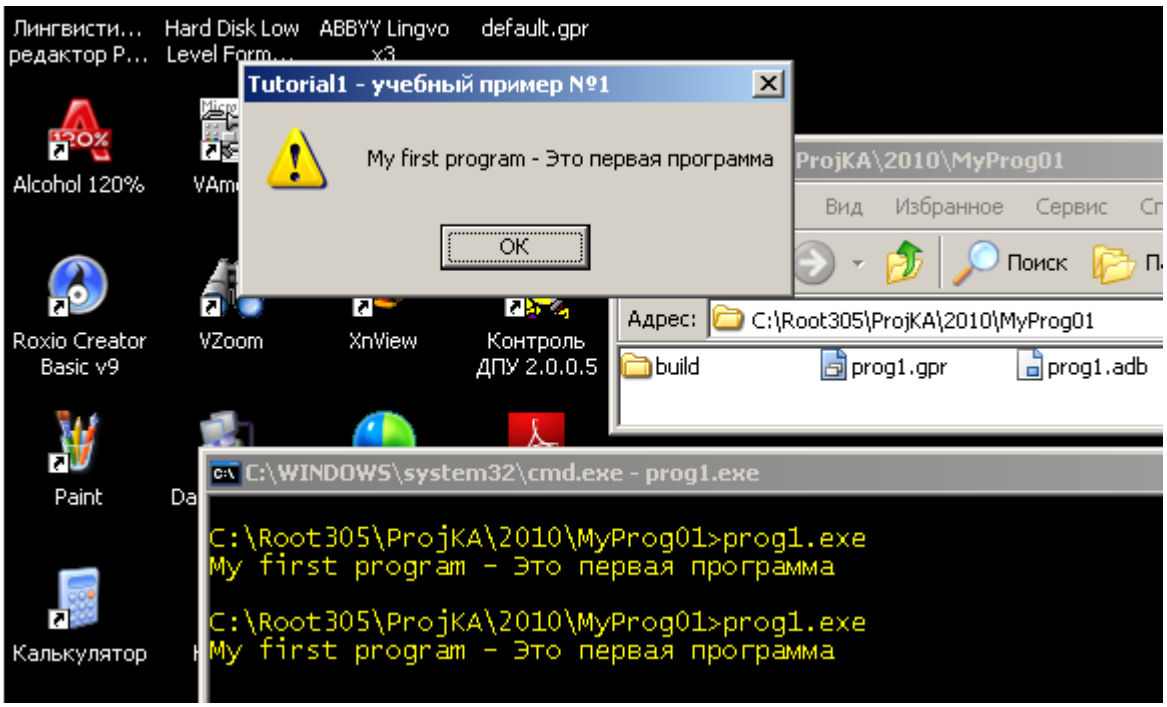

Рис. 26. Производим вывод сообщений на консоль и в окно диалога

3. Пакет GWindows библиотеки OEM.

Глава "Пакет GWindows библиотеки OEM" посвящена адаптированному под библиотеку OEM пакету GWindows/GNATCOM. Глубокая переработка была проведена и утилит, примеров и тестов. В результате следующие цели были достигнуты:

1) Без конфликтная работа в новых версиях GNAT GPL, которая ориентирована на новый стандарт Ada-2005/2012 и IDE GPS.
2) Поддержка национальных кодировок (в первую очередь русской, но должно работать с украинской и другими установленными как основные языки в ОС).
3) Простата повторного применения "старых" исходных текстов программ, использующие оригинальный пакет GWindows/GNATCOM.

Оригинальный пакет GWindows/GNATCOM в версии GPL не развивается автором этого пакета с 2004 года (http://www.gnavi.org). Тем не менее, идея, которая лежала в его основе, актуальна и сейчас, так как позволяет базироваться только на уровне "чистого" WIN32 (обоснование смотри Глава 1) и языке программирования Ada. При этом достаточно эффективно создается сложная комплексная программа. Примером может служить [5] распределенная научно-исследовательская система для отработки и тестирования встраиваемых в FPGA алгоритмов сжатия видеоинформации. Программные средства двух-мониторной графической станции этой системы полностью разработаны на языке Ada-95 и пакета GWindows/GNATCOM (Рис. 27).

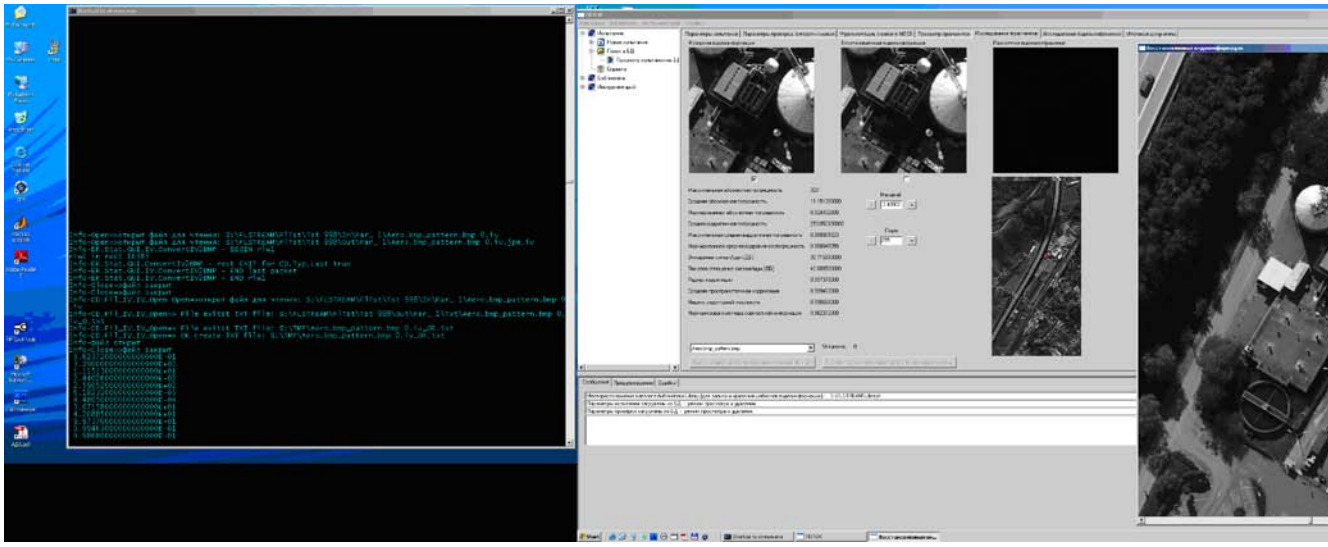

Рис. 27. Копия экранов двух-мониторной графической станции на базе пакета GWindows/GNATCOM

Пакет GWindows позиционируется как "Professional Open Source Ada 95 Win32 RAD Framework". В отличие от оригинала, в библиотеке OEM оконная подсистема сразу настроена на UNICOD-16, включен только функционал, который действительно востребован в рамках идеологии библиотеки OEM – как открытой для развития платформы приложений Win32 созданных на языке программирования Ada (эта идея будет раскрыта в последующих главах).

Обзор обучающих примеров.

Обучающие примеры находятся в каталоге `\oemtutorials-gui2010` (далее по тексту идет ссылка на каталог `\oemtutorials-gui`, имя базового каталога в данном случае не имеет значение) и объединены одним проектом `oemtutorials.gpr` все 21 программа. Для загрузки проекта в GPS достаточно дважды щелкнуть по нему левой клавишей мыши.

Примеры программ для обучения поставляются готовые к исполнению, также для их изучения их можно модифицировать по своему усмотрению.

Запуск обучающего примера программы можно выполнить из среды GPS Рис. 28

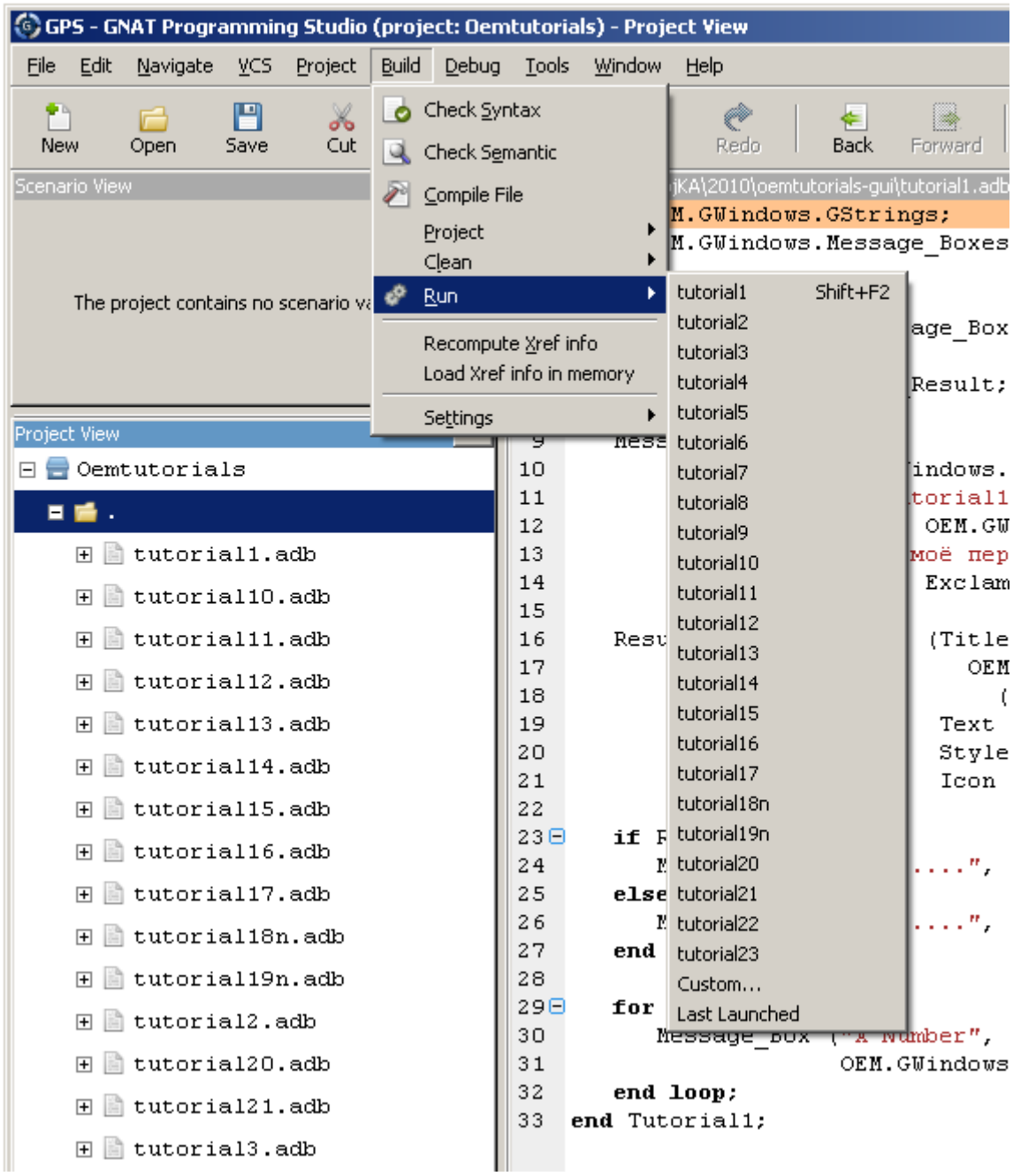

Рис. 28. Выбор программы для запуска через меню GPS

Необходимо при этом установить "птичку" в поле выбора **Use external terminal** Рис. 29.

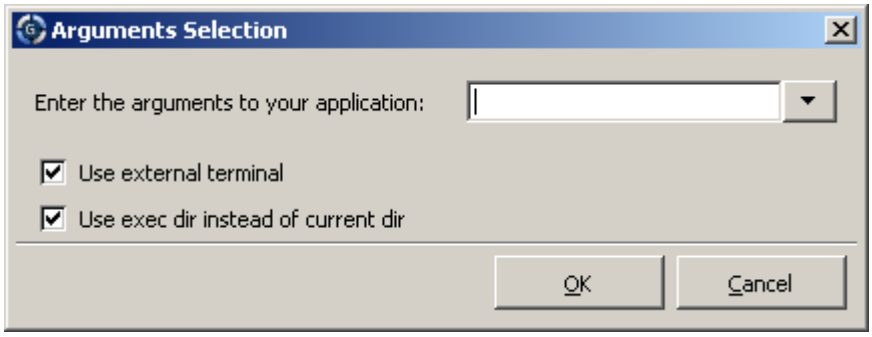

Рис. 29. Установите "птичку" в поле выбора Use external terminal

Обучающая программа №1 – пакет OEM.GWindows.Message_Boxes.

Первая программа – несколько обобщенный вариант примера, который был создан в Главе 2. Текст программы приведен ниже:

```
with OEM.GWindows.GStrings;
with OEM.GWindows.Message_Boxes;

procedure Tutorial1 is
   use OEM.GWindows.Message_Boxes;

   Result : Message_Box_Result;
begin
   Message_Box (Title =>
                  OEM.GWindows.GStrings.To_GString_From_String
                    ("Tutorial1 - учебный пример №1"),
                Text  => OEM.GWindows.GStrings.To_GString_From_String
                  ("Это моё первое OEM.GWindows Application"),
                Icon  => Exclamation_Icon);

   Result := Message_Box (Title =>
                            OEM.GWindows.GStrings.To_GString_From_String
                              ("Tutorial1 - учебный пример №1"),
                          Text  => "Nice GUI huh?",
                          Style => Yes_No_Box,
                          Icon  => Question_Icon);

   if Result = Yes then
      Message_Box ("Cool....", "I like your answer");
   else
      Message_Box ("What....", "You have no taste");
   end if;

   for I in 1 .. 3 loop
      Message_Box ("A Number",
                   OEM.GWindows.GStrings.To_GString_From_String (I'Img));
   end loop;
end Tutorial1;
```

Первой строкой подключаем пакет `OEM.GWindows.GStrings` . Функция этого пакета To_GString_From_String преобразует строку типа String в строку UNICODE-16 типа GString с учетом локализации операционной системы к национальному языку (в рассмотренном примере предполагается, что это русский язык).

Пакет **OEM.GWindows.Message_Boxes** состоит из трех разновидностей прототипов:

Диалог для выдачи сообщения.

Диалог для ввода строки.

Выдача стандартных (назначенных в ОС) звуковых сообщений.

Причем Диалоги могут быть как процедурами (первый) так функцией (второй). В дополнение они могут быть связаны с родительским окном. В нашем примере не связаны с окном приложения. Подробное описание пакета **OEM.GWindows.Message_Boxes** смотрите в Приложении А.

Используя контекстное меню редактора текста программы в GPS легко открыть файл описания пакета OEM.GWindows.Message_Boxes . Для этого наведем курсор мыши на имя пакета и нажмем правую клавишу. Выберем курсором "Goto declaration of Message_Boxes" и счёлкнем левой кнопкой мыши Рис. 30. Загрузившемся файле можно найти и скопировать в программу имена с идентификаторов Style или Icon, если Вы их забыли, а также прототипы функций и процедур, определенных в пакете. Выбрав в контекстном меню "Goto body of Message_Boxes" загрузится "реализация" пакета. Здесь можно посмотреть каким образом реализована та или другая функция или процедура. Если Вы внесете изменения в исходный текст пакета библиотеки, то при очередной трансляции он будет учтен в Вашей программе.

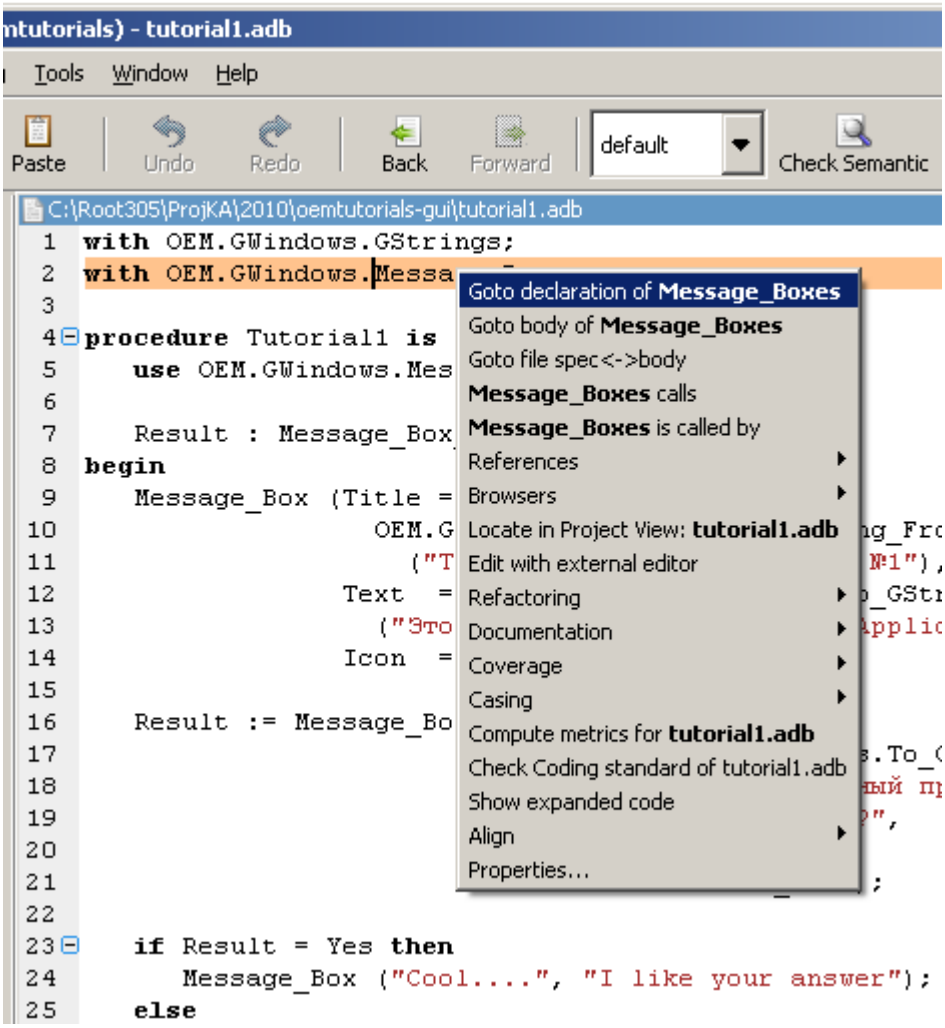

Рис. 30. Вызов контекстного меню для загрузки пакета

Подробно о возможностях GPS можно найти в Help документе Using the GNAT Programming Studio.

Обучающая программа №2 – GUI программа.

Следующая программа – это самая простая оконная графическая программа, которую можно написать, используя OEM.GWindows. Вот её исходный текст:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Application;
with OEM.GWindows.GStrings;

procedure Tutorial2 is
   pragma Linker_Options ("-mwindows");

   Main_Window : Main_Window_Type;
begin
   Create (Main_Window, OEM.GWindows.GStrings.To_GString_From_String
                              ("My First Window - Это моё первое ОКНО"));
   Visible (Main_Window, True);

   OEM.GWindows.Application.Message_Loop;
end Tutorial2;
```

В первую очередь необходимо создать объект типа Main_Window_Type . ОС MS Windows – объектная ОС, которая рассматривает "ОКНА" как объекты с определенными свойствами. Тип Base_Window_Type является родителем (это корень всей иерархии оконных объектов) для Main_Window_Type, который имеет свойства и методы главного окна приложения. Основное его свойство – это закрытие окна завершает работу приложения. По умолчанию окно создается не видимым, поэтому ему необходимо послать сообщение стать видимым. Для этого вызываем процедуру Visible (вызывается процедура Win32 - "ShowWindow").

Важной частью программы является вызов процедуры Message_Loop из пакета OEM.GWindows.Application. Для главного окна приложения в этой процедуре отрабатывается стандартный цикл обработки оконных сообщений в MS Windows:

```
while GetMessage (pMSG, 0, 0, 0) /= 0 loop
   Process_Message (pMSG, Window);
end loop;
```

Каждая задача (Task) программы, в которой используется GUI, должна иметь Message_Loop. Созданное "ОКНО" может управляться с любой задачи программы (данный функционал обеспечивает Win32 через посылку и обработку сообщений).

Как было показано в Главе 2, в программе всегда создаётся консоль, только после этого, в процессе работы приложения, создаются "ОКНА" GUI. Если консоль не нужна в программе, то необходимо запретить её создание через соответствующий ключ к компоновщику:

```
pragma Linker_Options ("-mwindows");
```

В этом случае нельзя использовать процедуры стандартного ввода/вывода из таких пакетов как Ada.Text_IO и аналогичных, так как это приведет к генерации исключения (через raise) по ошибке ввода/вывода. В тоже время удобно иметь консоль для отладки программы и для мониторинга выполняемых задач в мульти задачной программе.

Обучающая программа №3 – Динамическое создание ОКОН.

Создав главное окно программы, можно "динамически" создавать объекты ОКНО по мере необходимости. Для этого используется ссылочный тип Window_Access из пакета OEM.GWindows.Windows. Ниже приведён пример создания трех динамических окон:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Windows; use OEM.GWindows.Windows;
with OEM.GWindows.GStrings;
with OEM.GWindows.Application;

procedure Tutorial3 is
   pragma Linker_Options ("-mwindows");

   function "+"(S : String) return OEM.GWindows.GString
     renames OEM.GWindows.GStrings.To_GString_From_String;
```

```
   Main_Window : Main_Window_Type;
begin
   Create (Main_Window, +"The Main Window - Это главное ОКНО");
   Visible (Main_Window, True);

   declare
      A_Window : Window_Access;
   begin
      for N in 1 .. 3 loop
         A_Window := new Window_Type;

         Create (A_Window.all,
                 +("Dynamic Window - Это окно создано через NEW" & N'Img),
                 Width      => 375,
                 Height     => 175,
                 Is_Dynamic => True);
         Visible (A_Window.all);
      end loop;
   end;

   OEM.GWindows.Application.Message_Loop;
end Tutorial3;
```

Во первых определяется область видимости ссылочного типа операторными скобками DECLARE. Затем создаём собственно объект ОКНА: выделяем и инициализируем память используя `A_Window := new Window_Type;`, вызываем процедуру `Create` в которой `Is_Dynamic => True`. Переменная `A_Window` будет уничтожена, как только она выйдет за область видимости. Выделенная память созданного объекта ОКНА будет уничтожена самой `OEM.GWindows` и Win32.

Попутно в примере продемонстрирован приём переименования имен функций:

```
   function "+"(S : String) return OEM.GWindows.GString
     renames OEM.GWindows.GStrings.To_GString_From_String;
```

Теперь внутри процедуры `Tutorial3` для вызова функции `To_GString_From_String` достаточно написать +(…). Например:

```
+("Dynamic Window - Это окно создано через NEW" & N'Img)
```

Подробней об этом можно почитать в [1].

Обучающая программа №4 – Обработчик событий.

В предыдущих примерах показали, как создать окно и отобразить его. Следующий шаг – это создание своих обработчиков событий для вновь создаваемых окон.

OEM.GWindows позволяет для этого использовать два подхода:

1) Модель наследования.
2) Создание своего обработчика событий.

В первый подход удобен в тех случаях, когда необходимо расширить имеющийся функционал методов объекта окна. Второй – это задание новых свойств для уже созданных окон и элементов управления.

В случае модели наследования необходимо создать новый тип окна:

```
   type My_Window_Type is
     new OEM.GWindows.Windows.Main.Main_Window_Type with private;
   type My_Window_Access is access all My_Window_Type;
   type Pointer_To_My_Window_Class is access all My_Window_Type'Class;

private
   type My_Window_Type is
     new OEM.GWindows.Windows.Main.Main_Window_Type with null record;
```

И переопределить один или несколько методов родительского типа:

```
   procedure On_Close (Window    : in out My_Window_Type;
                       Can_Close :    out Boolean);
```

Оформляем всё это в отдельный пакет и в файлы (оформлять в отдельные файлы не обязательно, но в данном случае целесообразно) [1]. В результате получаем:

Файл описания пакета (имя файла в соответствии с соглашениями о именах которые были назначены в проекте):

```
with OEM.GWindows.Windows.Main;

package Tutorial4_Window is

   type My_Window_Type is
     new OEM.GWindows.Windows.Main.Main_Window_Type with private;
   type My_Window_Access is access all My_Window_Type;
   type Pointer_To_My_Window_Class is access all My_Window_Type'Class;

   procedure On_Close (Window    : in out My_Window_Type;
                       Can_Close :    out Boolean);

private
   type My_Window_Type is
     new OEM.GWindows.Windows.Main.Main_Window_Type with null record;
end Tutorial4_Window;
```

Файл реализации пакета:

```
with OEM.GWindows.Message_Boxes;
with OEM.GWindows.GStrings;

package body Tutorial4_Window is
   function "+"(S : String) return OEM.GWindows.GString
     renames OEM.GWindows.GStrings.To_GString_From_String;

   --------------
   -- On_Close --
   --------------

   procedure On_Close
     (Window    : in out My_Window_Type;
      Can_Close :    out Boolean)
   is
      use OEM.GWindows.Message_Boxes;
   begin
      Can_Close := Message_Box (Window, +("Tutorial4 - учебный пример №4"),
                                +("Ok to close? - Да чтобы закрыть?"),
                                Yes_No_Box, Question_Icon) =  Yes;
   end On_Close;

end Tutorial4_Window;
```

Сохранить эти файлы проще всего в тот же каталог где находится файл главной программы. Если предполагается использовать пакет `Tutorial4_Window` и в других программах, то можно сохранить и в другом, отдельном каталоге. В этом случае необходимо добавить путь поиска исходных файлов программы в проект Рис. 31 и нажав кнопку Add Рис. 32.

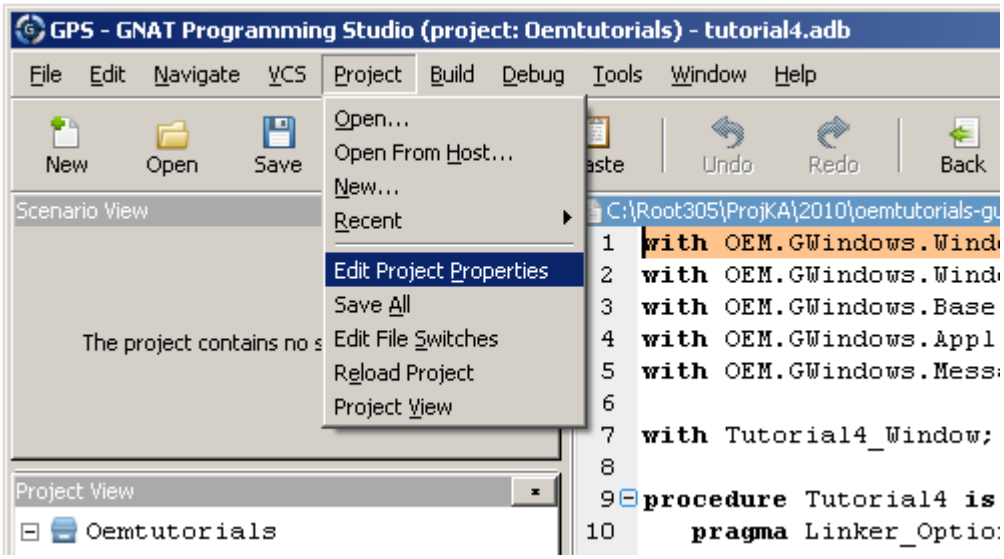

Рис. 31. Выбор через меню GPS диалога установки свойств проекта

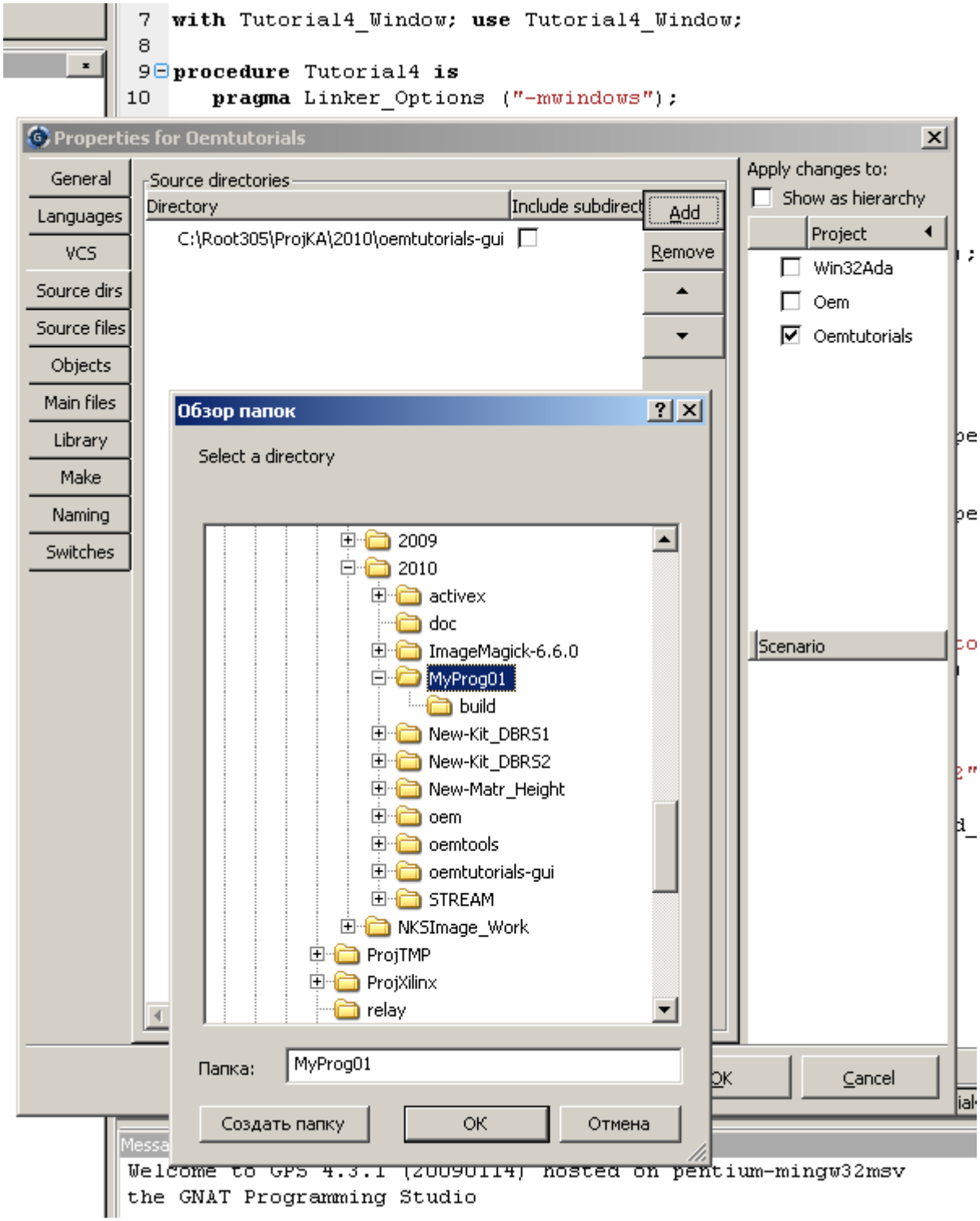

Рис. 32. Добавление в текущий проект каталога для поиска файлов программы

Сохраните и перезагрузите проект (как это сделать описано в Главе 2, желательно также изучить документацию по GPS).

Теперь можно проверить работу нового окна. Для этого достаточно немного модифицировать пример №2.

Сохраним файл программы №2 под другим именем, например tutorial2_4.adb в каталоге, где расположены все примеры Рис. 33.

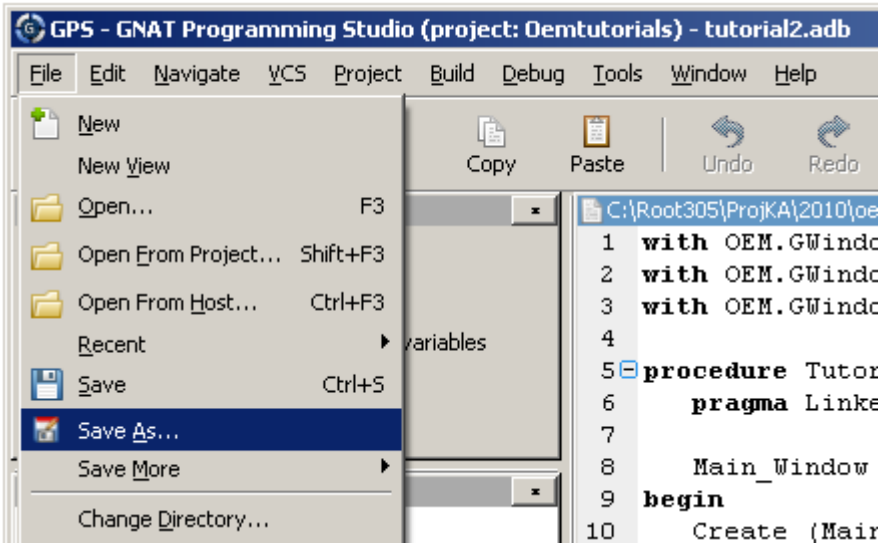

Рис. 33. Сохранение текущего файла под другим именем через меню GPS.

Изменим текст программы:

1) Имя процедуры исправим на Tutorial2_4.
2) Удалим первую строку за не надобностью. Добавим строку в начале файла **with Tutorial4_Window; use Tutorial4_Window;**.
3) Изменим тип переменной **Main_Window : My_Window_Type;**.
4) Сохраним внесенные изменения в файле Tutorial2_4.adb.

В результате получаем следующую программу:

```
with Tutorial4_Window; use Tutorial4_Window;
with OEM.GWindows.Application;
with OEM.GWindows.GStrings;

procedure Tutorial2_4 is
   pragma Linker_Options ("-mwindows");

   Main_Window : My_Window_Type;
begin
   Create (Main_Window, OEM.GWindows.GStrings.To_GString_From_String
                              ("My First Window - Это моё первое ОКНО"));
   Visible (Main_Window, True);

   OEM.GWindows.Application.Message_Loop;
end Tutorial2_4;
```

Откроем диалог свойств проекта Рис. 31. и добавим файл Tutorial2_4.adb в список Main files., нажав кнопку Add и выбрав файл Tutorial2_4.adb в списке Select files Рис. 34.

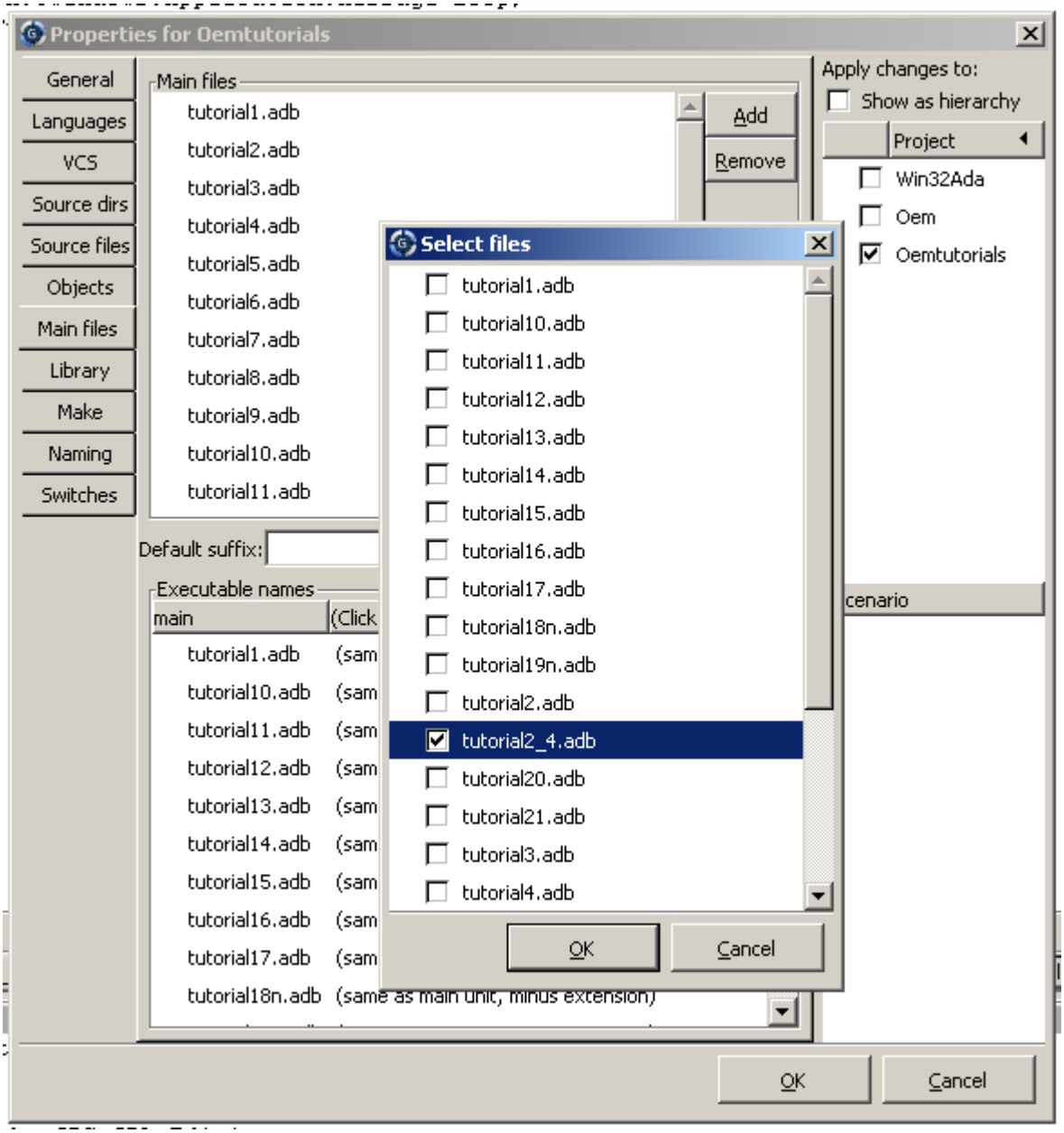

Рис. 34. Добавление Main files в текущий проект.

Построим новый пример Рис. 35. и запустим его на выполнение Рис. 28, Рис. 29.

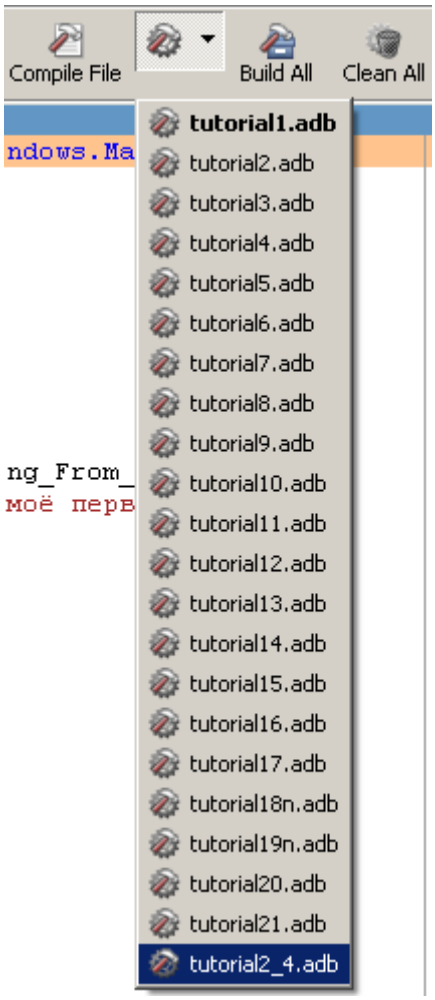

Рис. 35. Выбор программы для трансляции (build)

Теперь при закрытии окна программы появляется предупреждение Рис. 36.

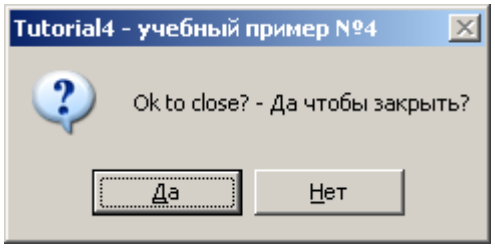

Рис. 36. Созданный новый тип имеет другой метод, который обрабатывает событие закрытия объекта окна, выдавая предупреждение на экран.

Первый подход, создание нового типа окна, оправдывает себя в том случае, когда необходимо реализовать несколько однотипных окон. Создание своего обработчика событий (второй подход) не требует нового типа окна, а переопределяет в уже созданном окне существующий метод обработки события новым. Когда создается одно специфическое окно, то второй подход конечно проще в написании и использовании.

Каждый метод обработки события существует в двух взаимодополняющих процедурах (имеется ввиду в OEM.GWindows):

1) Процедура обработки события.
2) Процедура для встраивания нового обработчика событий окна.

По соглашению об именовании, процедуры для встраивания обработчика имеют суффикс "_Handler" и имеют два параметра: объект и обработчик событий. Использование процедуры для встраивания обработчика можно только после создания экземпляра окна. Ниже приведен пример такой процедуры:

```
:
   procedure On_Close_Handler (Window  : in out Window_Type;
                               Handler : in Close_Event);
```

Для того чтобы реализовать обработчик событий On_Close необходимо создать процедуру с тем же прототипом как и тип обработчика для этого события. Тип обработчика события – это не более чем ссылка на процедуру, например:

```
type Close_Event is access
  procedure (Window    : in out GWindows.Base.Base_Window_Type'Class;
             Can_Close :    out Boolean);
```

В прототипе процедуры обработчика событий первым параметром является всегда **Window : in out GWindows.Base.Base_Window_Type'Class** . Остальные параметры, если они существуют, передаются для интерпретации наследникам процедуры обработчика событий окна (вообще-то это стандартная схема Win32). В приведенном примере второй параметр прототипа – это **Can_Close**. Если процедура возвращает True, то окно закрывается.

Можно продемонстрировать всё вышеизложенное на примере:

```
declare
   Main_Window : OEM.GWindows.Windows.Main.Main_Window_Type;

   procedure Do_On_Close
     (Window    : in out OEM.GWindows.Base.Base_Window_Type'Class;
      Can_Close :    out Boolean)
   is
      use OEM.GWindows.Message_Boxes;
   begin
      Can_Close := Message_Box (Window, "Tutorial4", "Ok to close?",
                                Yes_No_Box, Question_Icon) =  Yes;
   end Do_On_Close;

begin
   Create (Main_Window, "Event Handling Window - Version 2");
   Visible (Main_Window, True);
   On_Close_Handler (Main_Window, Do_On_Close'Unrestricted_Access);

   OEM.GWindows.Application.Message_Loop;
end;
```

Необходимо обратить внимание, что процедура **Do_On_Close** не определена на уровне библиотеки (имя процедуры не имеет значение, но лучше придерживать рекомендации в виде префикса **DO_**), поэтому необходимо использовать **Unrestricted_Access**. Важно чтобы процедура **Do_On_Close** не выходила за пределы видимости пока существует **Main_Window** .

Полный пример обоих типов наследования приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Windows; use OEM.GWindows.Windows;
with OEM.GWindows.Base;
with OEM.GWindows.Application;
with OEM.GWindows.Message_Boxes;

with Tutorial4_Window; use Tutorial4_Window;

procedure Tutorial4 is
   pragma Linker_Options ("-mwindows");
begin
```

```
      declare
         My_Window   : Tutorial4_Window.My_Window_Type;
      begin
         Create (My_Window, "Event Handling Window - Version 1");
         Visible (My_Window, True);

         OEM.GWindows.Application.Message_Loop;
      end;

      declare
         Main_Window : OEM.GWindows.Windows.Main.Main_Window_Type;

         procedure Do_On_Close
           (Window    : in out OEM.GWindows.Base.Base_Window_Type'Class;
            Can_Close :    out Boolean)
         is
            use OEM.GWindows.Message_Boxes;
         begin
            Can_Close := Message_Box (Window, "Tutorial4", "Ok to close?",
                                      Yes_No_Box, Question_Icon) =  Yes;
         end Do_On_Close;

      begin
         Create (Main_Window, "Event Handling Window - Version 2");
         Visible (Main_Window, True);
         On_Close_Handler (Main_Window, Do_On_Close'Unrestricted_Access);

         OEM.GWindows.Application.Message_Loop;
      end;

   end Tutorial4;
```

К особенностям примера можно ещё отнести использование declare и `Message_Loop` . Сначала создаётся первое окно и только после его закрытия создаётся второе.

Обучающая программа №5 – Управляющие элементы.

Сейчас, когда понятны основы пакета OEM.GWindows и его модели обработки событий, мы можем перейти к созданию окон немного более годных к употреблению добавляя некоторые средства управления. Управляющие элементы – это другой тип оконных объектов, чем те, которые мы рассматривали раньше.

Обучающая программа №5 – это простой пример добавления элементов управления к окну приложения. Ниже приведен текст программы №5:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
with OEM.GWindows.Edit_Boxes; use OEM.GWindows.Edit_Boxes;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.Message_Boxes;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial5 is
   pragma Linker_Options ("-mwindows");

   Main_Window : OEM.GWindows.Windows.Main.Main_Window_Type;
   User_Name   : OEM.GWindows.Edit_Boxes.Edit_Box_Type;
   Disp_Button : OEM.GWindows.Buttons.Button_Type;

   procedure Do_Display
     (Window    : in out OEM.GWindows.Base.Base_Window_Type'Class)
   is
   begin
    OEM.GWindows.Message_Boxes.Message_Box ("Controls Window", Text (User_Name));
   end Do_Display;
```

```
begin
   Create (Main_Window, "Controls Window", Width => 200, Height => 125);
   Visible (Main_Window, True);
   Keyboard_Support (Main_Window, True);

   Create_Label (Main_Window, "Name :", 10, 10, 50, 25);

   Create (User_Name, Main_Window, "", 70, 10, 100, 25);

   Create (Disp_Button, Main_Window, "&Display", 10, 50, 75, 30);
   On_Click_Handler (Disp_Button, Do_Display'Unrestricted_Access);

   OEM.GWindows.Application.Message_Loop;
end Tutorial5;
```

Как можно видеть, добавление управляющего элемента сводится к созданию процедуры обработки событий окна управляющего элемента, создание собственно экземпляра объекта и определение расположения его в родительском окне (управляющие элементы имеют в основе тех же предков, но имеют предопределенный внешний вид, дополнительные методы обработки событий и свойства). Для статических управляющих элементов (например: текстовые метки, иконки, изображения и т.д.) существует возможность не создавать экземпляра объекта в основной программе, если не будут переопределяться его обработчики событий. В нашем случае вместо процедуры **Create** для текстовой метки используется процедура **Create_Label.**. Экземпляр объекта типа **Label_Type** создается динамически в теле процедуры и размещается в куче (см. пример №3).

Новым свойством для главного окна, которое мы активизируем, это **Keyboard_Support**. По умолчанию **Keyboard_Support** имеет значение False. Вызвав процедуру Keyboard_Support (Main_Window, True) разрешается стандартная поддержка управления клавиатурой, такой как клавиша Tab или другого прямого управления событиями окна посредством “hot key” ("горячих клавиш").

Необходимо обратить внимание, что процедура обработки события нажатия клавиши названа **Do_Display**. Как видим название самой процедуры несет смысловое значение выбранное программистом. Возможно, правильней было бы назвать её **Do_On_Click**.

Обучающая программа №6 – Размещаем ВСЁ или OEM.GWindows.Scroll_Panels.

Необходимость применить OEM.GWindows.Scroll_Panels возникает тогда, когда не достаточно свободного пространства окна для всех управляющих элементов, размещаемых в нем. В этом случае создается виртуальное пространство окна с помощью Scroll Panel, в котором пользователь может перемещаться, используя scroll bars. Ниже следующий код программы демонстрирует применение Scroll Panel как в главном окне, так и как отдельного управляющего элемента:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Scroll_Panels; use OEM.GWindows.Scroll_Panels;
with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
with OEM.GWindows.Edit_Boxes; use OEM.GWindows.Edit_Boxes;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.Message_Boxes;
with OEM.GWindows.Events;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial6 is
   pragma Linker_Options ("-mwindows");

begin
   --  Using a Scroll Panel as a Window

   declare
      Main_Window : OEM.GWindows.Scroll_Panels.Scroll_Panel_Type;
      User_Name   : OEM.GWindows.Edit_Boxes.Edit_Box_Type;
```

```
   Disp_Button : OEM.GWindows.Buttons.Button_Type;

   procedure Do_Display
     (Window    : in out OEM.GWindows.Base.Base_Window_Type'Class)
   is
   begin
      OEM.GWindows.Message_Boxes.Message_Box
        ("Scroll Window", Text (User_Name));
   end Do_Display;
begin
   Create (Main_Window, "Scroll Window", Width => 200, Height => 150);
   Visible (Main_Window, True);
   On_Destroy_Handler (Main_Window,
                       OEM.GWindows.Events.Do_End_Application'Access);
   --  Since Scroll_Panel_Type is not derived from Main_Window_Type
   --  it will not automaticly close the application when the window
   --  is destroyed. This handler will do that for us.

   Panel_Size (Main_Window, 500, 500);

   Keyboard_Support (Main_Window.Panel, True);

   Create_Label (Main_Window.Panel, "Name :", 150, 10, 50, 25);

   Create (User_Name, Main_Window.Panel, "", 230, 10, 100, 25);

   Create (Disp_Button, Main_Window.Panel, "&Display", 150, 50, 75, 30);
   On_Click_Handler (Disp_Button, Do_Display'Unrestricted_Access);

   Focus (User_Name);

   OEM.GWindows.Application.Message_Loop;
end;

--  Using a Scroll Panel as a control

declare
   Main_Window  : OEM.GWindows.Windows.Main.Main_Window_Type;
   Scroll_Panel : OEM.GWindows.Scroll_Panels.Scroll_Panel_Type;
   User_Name    : OEM.GWindows.Edit_Boxes.Edit_Box_Type;
   Disp_Button : OEM.GWindows.Buttons.Button_Type;

   procedure Do_Display
     (Window    : in out OEM.GWindows.Base.Base_Window_Type'Class)
   is
   begin
      OEM.GWindows.Message_Boxes.Message_Box
        ("Scroll Window", Text (User_Name));
   end Do_Display;
begin
  Create (Main_Window, "Scrolling Window 2", Width => 400, Height => 400);
  Visible (Main_Window, True);

  Create_As_Control (Scroll_Panel, Main_Window,
                     Top    => 20,
                     Left   => 20,
                     Width  => 300,
                     Height => 300);

  Panel_Size (Scroll_Panel, 500, 500);

  Keyboard_Support (Scroll_Panel.Panel, True);

  Create_Label (Scroll_Panel.Panel, "Name :", 150, 10, 50, 25);

  Create (User_Name, Scroll_Panel.Panel, "", 230, 10, 100, 25);

  Create (Disp_Button, Scroll_Panel.Panel, "&Display", 150, 50, 75, 30);
```

```
      On_Click_Handler (Disp_Button, Do_Display'Unrestricted_Access);

      Focus (User_Name);

      OEM.GWindows.Application.Message_Loop;
    end;

end Tutorial6;
```

В примере размер виртуальной области вывода окна задаётся процедурой **Panel_Size** из пакета **OEM.GWindows.Scroll_Panels**, соответственно для типа **Scroll_Panel_Type**. В первой части примера Главное окно – это экземпляр объекта типа **Scroll_Panel_Type** из пакета **OEM.GWindows.Scroll_Panels**. В отличии от типа **Main_Window_Type** из пакета **OEM.GWindows.Windows.Main**, тип **Scroll_Panel_Type** не имеет "правильного" предопределённого обработчика событий по умолчанию, поэтому его необходимо установить:

```
On_Destroy_Handler (Main_Window,
                    OEM.GWindows.Events.Do_End_Application'Access);
```

Во второй части примера **Scroll_Panel_Type** – это обычный управляющий элемент, важно чтобы он инициализировался раньше, чем остальные, которые ссылаются на него. Дело в том что сам экземпляр типа окна созданный в программе ещё не создаёт его в **Win32**, Создание экземпляра окна в Win32 происходит после вызова процедуры Create соответствующего типа окна (см. пример №3).

В обоих случаях родительским окном остальных управляющих элементов является окно типа **Scroll_Panel_Type.**

Как видно из примера, любой объект являющийся наследником **OEM.GWindows.Windows.Window_Type** может создаваться как окно верхнего уровня, так и как управляющий элемент.

Попутно обратим внимание, что после разрешения поддержки стандартных событий от клавиатуры для окна (в данном примере тип окна **Scroll_Panel_Type**) можно установить положение курсора/"фокуса" на нужном управляющем элементе:

```
Focus (User_Name);
```

Данная процедура устанавливает курсор и фокус ввода клавиатуры на управляющий элемент текстовое окно ввода типа Edit_Box_Type.

Обучающая программа №7 – Рисование или OEM.GWindows.Drawing_Panels.

В дополнение к размещению управляющих элементов в окне мы можем рисовать на окне. В OEM.GWindows поддерживается две основных возможности рисования:

1) Использование Drawing_Panel.

2) Непосредственно напрямую рисовать в окне.

Во втором случае, необходимо установить свой собственный метод обработки событий On_Paint, последовательно перерисовывать всё ранее нарисованное по соответствующему запросу ОС (это классический вариант процедуры окна в ОС MS Windows). В первом случае всю эту работу выполнит Drawing_Panel.

В обучающей программе №7 мы разберем, как использовать Drawing_Panel. В следующем №8 будет продемонстрировано применение метода обработки событий On_Paint. В последующем будут продемонстрированы другие аспекты применения графического вывода и объектов с ним связанных в OEM.GWindows.

Текст обучающей программы №7 приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Drawing_Panels; use OEM.GWindows.Drawing_Panels;
with OEM.GWindows.Scroll_Panels; use OEM.GWindows.Scroll_Panels;
with OEM.GWindows.Drawing;
with OEM.GWindows.Colors; use OEM.GWindows.Colors;
with OEM.GWindows.Events;
with OEM.GWindows.Base;
with OEM.GWindows.Application;
```

```
procedure Tutorial7 is
   pragma Linker_Options ("-mwindows");

   use OEM.GWindows.Drawing;
   use OEM.GWindows.Drawing_Panels;

   procedure Draw_Something
     (Canvas : in out OEM.GWindows.Drawing.Canvas_Type'Class)
   is
   begin
      for N in 1 .. 100 loop
         Fill_Rectangle (Canvas,
                         (N * 2, N * 2, N + 20, N + 20),
                         COLOR_3DHILIGHT);
         Fill_Rectangle (Canvas,
                         (N * 2 + 200, N * 2 + 200, N + 180, N + 180),
                         COLOR_DESKTOP);
      end loop;
   end Draw_Something;

begin
   --  Using a Drawing Panel as a Window

   declare
      Main_Window : OEM.GWindows.Drawing_Panels.Drawing_Panel_Type;
      Canvas      : OEM.GWindows.Drawing_Panels.Drawing_Canvas_Type;
   begin
      Create (Main_Window, "Drawing Window", Width => 200, Height => 125);
      Visible (Main_Window, True);
      On_Destroy_Handler (Main_Window,
                          OEM.GWindows.Events.Do_End_Application'Access);
      --  Since Drawing_Panel_Type is not derived from Main_Window_Type
      --  it will not automaticly close the application when the window
      --  is destroyed. This handler will do that for us.

      Auto_Resize (Main_Window, False);
      Resize_Canvas (Main_Window,
                     OEM.GWindows.Application.Desktop_Width,
                     OEM.GWindows.Application.Desktop_Height);
      --  By turning off auto resize and setting canvas to the size of
      --  of the desktop contents will be saved no matter how we
      --  resize the window

      Get_Canvas (Main_Window, Canvas);

      Draw_Something (Canvas);

      OEM.GWindows.Application.Message_Loop;
   end;

   --  Using a Drawing Panel as a Control

   declare
      Main_Window  : OEM.GWindows.Windows.Main.Main_Window_Type;
      Draw_Control : OEM.GWindows.Drawing_Panels.Drawing_Panel_Type;
      Canvas       : OEM.GWindows.Drawing_Panels.Drawing_Canvas_Type;
   begin
      Create (Main_Window, "Drawing Window", Width => 400, Height => 400);
      Visible (Main_Window, True);

      Create_As_Control (Draw_Control, Main_Window,
                         Top    => 20,
                         Left   => 20,
                         Width  => 300,
                         Height => 300);

      Get_Canvas (Draw_Control, Canvas);
```

```
      Draw_Something (Canvas);

      OEM.GWindows.Application.Message_Loop;
   end;

   --  Using a Drawing Panel in a Scroll Panel

   declare
      Main_Window  : OEM.GWindows.Windows.Main.Main_Window_Type;
      Scroll_Panel : OEM.GWindows.Scroll_Panels.Scroll_Panel_Type;
      Draw_Control : OEM.GWindows.Drawing_Panels.Drawing_Panel_Type;
      Canvas       : OEM.GWindows.Drawing_Panels.Drawing_Canvas_Type;
   begin
      Create (Main_Window, "Drawing Window", Width => 400, Height => 400);
      Visible (Main_Window, True);

      Create_As_Control (Scroll_Panel, Main_Window,
                         Top    => 20,
                         Left   => 20,
                         Width  => 300,
                         Height => 300);

      Panel_Size (Scroll_Panel, 500, 500);

      Create_As_Control (Draw_Control, Scroll_Panel.Panel,
                         Top    => 0,
                         Left   => 0,
                         Width  => 0,
                         Height => 0);
      Dock (Draw_Control, OEM.GWindows.Base.Fill);
      Dock_Children (Scroll_Panel.Panel);

      Get_Canvas (Draw_Control, Canvas);

      Draw_Something (Canvas);

      OEM.GWindows.Application.Message_Loop;
   end;

end Tutorial7;
```

В целом пример похож на предыдущий за исключением использования управляющего элемента типа **Drawing_Panel_Type** и типа **Drawing_Canvas_Type** из пакета **OEM.GWindows.Drawing_Panels**. В то же время методы для рисования находятся в пакете **OEM.GWindows.Drawing**. **Drawing_Canvas_Type** – это наследник типа из пакета **OEM.GWindows.Drawing**. Соответственно процедура рисования залитого прямоугольника **Fill_Rectangle** определена в пакете **OEM.GWindows.Drawing** где и определен базовый тип **Information_Canvas_Type** (все остальные являются его потомками).

Для правильной работы в третьей части программы используется методы **Dock** и **Dock_Children**.

Обучающая программа №8 – Рисование с обработчиком событий.

Как принято в ОС Windows для рисования, необходимо создать обработчик событий Paint (см. Обучающая программа №4 – Обработчик событий), который будет обрабатывать запросы ОС (сообщения ОС с кодом Paint). Этот подход доступен в OEM.GWindows как альтернатива Drawing Panels. Обучающая программа №8 демонстрирует, как это сделать:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Drawing;
with OEM.GWindows.Colors;
with OEM.GWindows.Types;
with OEM.GWindows.Base;
with OEM.GWindows.Application;
```

```
procedure Tutorial8 is
   pragma Linker_Options ("-mwindows");

   procedure Do_Paint
     (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
      Canvas : in out OEM.GWindows.Drawing.Canvas_Type;
      Area   : in     OEM.GWindows.Types.Rectangle_Type)
   is
     use OEM.GWindows.Drawing;
     use OEM.GWindows.Colors;
   begin
      for N in 1 .. 100 loop
         Fill_Rectangle (Canvas,
                         (N * 2, N * 2, N + 20, N + 20),
                         COLOR_3DHILIGHT);
         Fill_Rectangle (Canvas,
                         (N * 2 + 200, N * 2 + 200, N + 180, N + 180),
                         COLOR_DESKTOP);
      end loop;
   end Do_Paint;

   Main_Window : OEM.GWindows.Windows.Main.Main_Window_Type;
begin
   Create (Main_Window, "On_Paint Drawing Window",
          Width => 200, Height => 200);
   Visible (Main_Window, True);
   On_Paint_Handler (Main_Window, Do_Paint'Unrestricted_Access);

   OEM.GWindows.Application.Message_Loop;
end Tutorial8;
```

Как видно из примера, здесь нет никаких сюрпризов. Прототип процедуры обработки событий, кроме самого объекта окна, имеет "полотно" (*Canvas*) для рисования и прямоугольник области вывода (*Area*). Собственно сами процедуры для рисования на "полотне" как и сам тип **Canvas_Type** находятся в пакете **OEM.GWindows.Drawing**. Тип **Canvas_Type** является наследником типа **Information_Canvas_Type** из того же пакета **OEM.GWindows.Drawing**.

Подробное описание пакета **OEM.GWindows.Drawing** смотрите в Приложении А.

Обучающая программа №9 – Рисующие объекты или Drawing Objects.

Два важных понятия необходимо знать, чтобы рисовать в окнах ОС MS Windows и **OEM.GWindows** в частности – это *Canvas* и *Drawing Object*. Вы рисуете на *Canvas* с помощью *Drawing Object*. Можно получить **Canvas_Type** с любого окна **OEM.GWindows** напрямую через метод **Get_Canvas** или получить его на вход обработчиков событий **On_Paint** или **On_Erase_Background** как часть сообщения (в прототипах этих процедур вторым аргументом):

```
type Paint_Event is access
  procedure (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
             Canvas : in out OEM.GWindows.Drawing.Canvas_Type;
             Area   : in     OEM.GWindows.Types.Rectangle_Type);
```

Printer_Canvas_Type получаем в результате вызова процедур **Choose_Printer** или **Choose_Default_Printer** которые находятся в пакете **OEM.GWindows.Common_Dialogs**. Вы получите **Memory_Canvas_Type** из окна типа **Drawing Panels**.

Во всех этих случаях используются те же самые методы рисования из пакета **OEM.GWindows.Drawing** (так как все они являются наследниками типа **Information_Canvas_Type** из этого пакета).

В GWindows.Drawing_Objects существуют некоторое количество рисующих объектов Drawing_Objects экземпляры, которых необходимо назначить в *canvas* с помощью процедуры Select_Object. Только после создания, инициализации назначения (или другими словами

выбора) их, они будут использоваться процедурами рисования. Примерами Drawing_Objects могут служить *Brushes*, *pens*, *bitmaps* и *icons*.

Обучающая программа №9 демонстрирует использование некоторых из них:

```
with OEM.GWindows.Drawing_Panels; use OEM.GWindows.Drawing_Panels;
with OEM.GWindows.Drawing_Objects; use OEM.GWindows.Drawing_Objects;
with OEM.GWindows.Colors; use OEM.GWindows.Colors;
with OEM.GWindows.Events;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial9 is
   pragma Linker_Options ("-mwindows");

   Main_Window : OEM.GWindows.Drawing_Panels.Drawing_Panel_Type;
   Canvas      : OEM.GWindows.Drawing_Panels.Drawing_Canvas_Type;
   Brush       : OEM.GWindows.Drawing_Objects.Brush_Type;
   Pen         : OEM.GWindows.Drawing_Objects.Pen_Type;
begin
   Create (Main_Window, "On_Paint Drawing Window",
           Width => 200, Height => 200);
   Visible (Main_Window, True);
   On_Destroy_Handler (Main_Window,
                       OEM.GWindows.Events.Do_End_Application'Access);
   Auto_Resize (Main_Window, False);
   Resize_Canvas (Main_Window,
                  OEM.GWindows.Application.Desktop_Width,
                  OEM.GWindows.Application.Desktop_Height,
                  False);

   Get_Canvas (Main_Window, Canvas);

   Create_Pen (Pen,
               Style => Solid,
               Width => 3,
               Color => Blue);
   Select_Object (Canvas, Pen);

   Create_Solid_Brush (Brush,
                       Color => Orange);
   Select_Object (Canvas, Brush);

   Ellipse (Canvas,
            10, 10,
            Client_Area_Width (Main_Window) - 10,
            Client_Area_Height (Main_Window) - 10);

   Create_Hatch_Brush (Brush,
                       Style => Cross,
                       Color => Red);
   Select_Object (Canvas, Brush);

   Create_Stock_Pen (Pen, Null_Pen);
   Select_Object (Canvas, Pen);

   Rectangle (Canvas,
              25, 25,
              Client_Area_Width (Main_Window) - 25,
              Client_Area_Height (Main_Window) - 25);

   Redraw (Main_Window);
   --  Tell windows that a redraw of the panel is needed.

   OEM.GWindows.Application.Message_Loop;
end Tutorial9;
```

Обучающая программа №10 – Печать.

Печать в **OEM.GWindows** – это только расширение методов рисования на объекте *Canvas* с произвольными свойствами. Вы можете выбрать одну из двух процедур из пакета **OEM.GWindows.Common_Dialogs** для того чтобы выбрать принтер на который будет производиться вывод. Для выбора принтерного *Canvas* , Вам необходимо начать задание для печати документа и дать ему имя для спулера (spooler) печати используя процедуру **Start_Document**. Затем использовать процедуры **Start_Page** и **End_Page** обозначать окончание страницы.

Текст обучающей программы №10 приведен ниже:

```
with Interfaces.C;

with OEM.GWindows.Drawing; use OEM.GWindows.Drawing;
with OEM.GWindows.Common_Dialogs; use OEM.GWindows.Common_Dialogs;
with OEM.GWindows.Message_Boxes;
with OEM.GWindows.Windows;
with OEM.GWindows.GStrings;

procedure Tutorial10 is
   Canvas    : OEM.GWindows.Drawing.Printer_Canvas_Type;
   Settings  : OEM.GWindows.Common_Dialogs.DEVMODE;
   Flags     : Interfaces.C.unsigned := 0;
   From_Page : Natural := 1;
   To_Page   : Natural := 1;
   Copies    : Natural := 1;
   Success   : Boolean;
   Null_Win  : OEM.GWindows.Windows.Window_Type;

   function "+"(S : String) return OEM.GWindows.GString
     renames OEM.GWindows.GStrings.To_GString_From_String;
begin
   Choose_Printer (Null_Win, Canvas, Settings, Flags,
                   From_Page, To_Page, 1, 1, Copies, Success);

   if Success then
      Start_Document (Canvas, "Tutorial10 Document");
      Start_Page (Canvas);

      Put (Canvas, 100, 100, "Hello World!");

      Put (Canvas, 100, 800, +("Hello World! - Это значит ПРИВЕТ МИР русского
языка"));

      Ellipse (Canvas, 300, 300, 700, 700);

      End_Page (Canvas);
      End_Document (Canvas);
      OEM.GWindows.Message_Boxes.Message_Box ("Tutorial 10", "Done");
   else
      OEM.GWindows.Message_Boxes.Message_Box ("Tutorial 10",
                                              "Printing Canceled");
   end if;
end Tutorial10;
```

Обучающая программа №11 – Меню.

OEM.GWindows может создавать меню непосредственно в коде программы или загружать меню из встроенного в приложение так называемого *ресурса* по коду идентификатора меню в этом *ресурсе*. Для встраивания файла ресурса в программу необходимо задать соответствующее указание компоновщику. Например:

```
pragma Linker_Options ("dlgtest.coff");
```

Для создания файла типа COFF необходимо применять консольную программу **`windres.exe`** из комплекта дистрибутива GNAT GPL 2010. Ниже приведен один из возможных вызовов программы **`windres.exe.`**

```
windres -i dlgtest.rc -o ./build/dlgtest.coff
```

Удобно этот вызов оформить в виде MAKEFILE, например возможное содержание файла с именем MAKEFILE:

```
dialog_example.coff: dialog_example.rc
        windres -i dialog_example.rc -o ./build/dialog_example.coff

dlgtest.coff: dlgtest.rc
        windres -i dlgtest.rc -o ./build/dlgtest.coff
```

Тогда в GPS появится возможность непосредственно компилировать файл ресурсов. На Рис. 37 показано как это сделать.

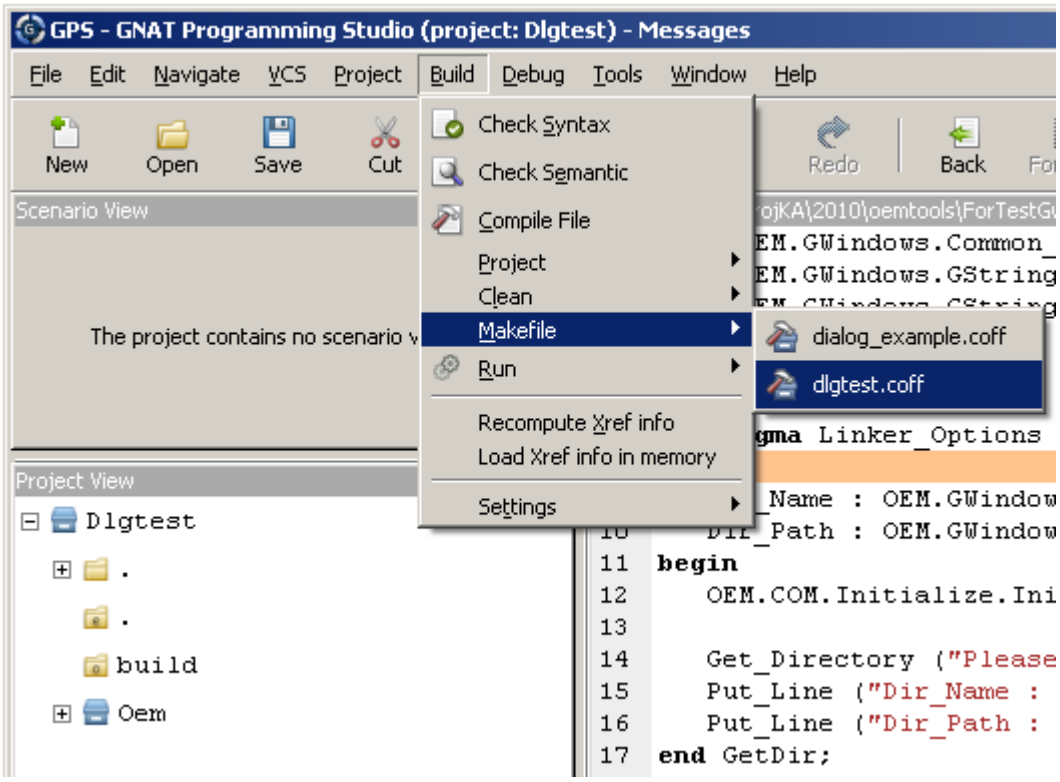

Рис. 37. Запуск MAKEFILE на исполнение в GPS.

Файл с расширением RC можно создать непосредственно в текстовом редакторе. Более подробно об этом будет написано в следующих Главах (см. примеры в каталоге \oemtools\ForTestGwindows\).

Меню, однажды созданное/загруженное может использоваться как меню ОКНА, так и как контекстное меню для этого окна. На Рис. 38 демонстрируется работа обучающей программы №11 демонстрирующая вызов контекстного меню по нажатию правой клавиши мыши.

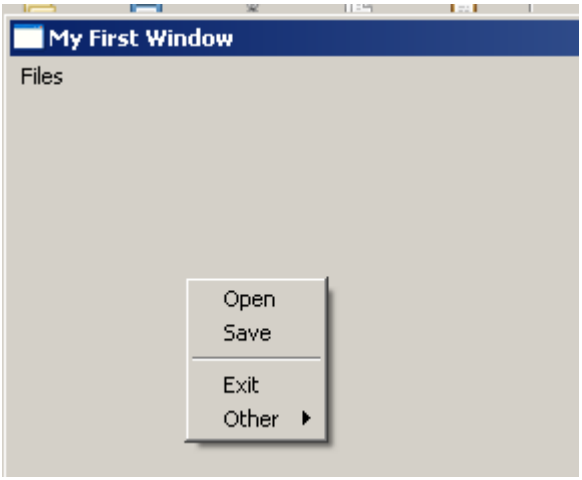

Рис. 38. Вызов контекстного меню в примере №11.

Текст обучающей программы №11 приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Menus; use OEM.GWindows.Menus;
with OEM.GWindows.Application;
with OEM.GWindows.Base;

procedure Tutorial11 is
   pragma Linker_Options ("-mwindows");

   Main_Window : Main_Window_Type;
   Main_Menu   : Menu_Type := Create_Menu;
   File_Menu   : Menu_Type := Create_Popup;
   Sub_Menu    : Menu_Type := Create_Popup;

   ID_Exit     : constant := 100;
   ID_Open     : constant := 101;
   ID_Save     : constant := 102;
   ID_About    : constant := 103;

   procedure Do_Menu_Select
     (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
      Item   : in     Integer)
   is
   begin
      case Item is
         when ID_Exit =>
            OEM.GWindows.Application.End_Application;
         when others =>
            null;
      end case;
   end Do_Menu_Select;

   procedure Do_Context_Menu
     (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
      X      : in     Integer;
      Y      : in     Integer)
   is
   begin
      Display_Context_Menu (Main_Window_Type (Window), File_Menu, 0, X, Y);
   end Do_Context_Menu;

begin
   Create (Main_Window, "My First Window");
   Visible (Main_Window, True);

   Append_Item (File_Menu, "&Open", ID_Open);
```

```
   Append_Item (File_Menu, "&Save", ID_Save);
   Append_Separator (File_Menu);

   Append_Item (File_Menu, "E&xit", ID_Exit);
   Append_Menu (Main_Menu, "&Files", File_Menu);

   Append_Item (Sub_Menu, "A&bout", ID_About);
   Append_Menu (File_Menu, "Other", Sub_Menu);

   Menu (Main_Window, Main_Menu);

   On_Menu_Select_Handler (Main_Window, Do_Menu_Select'Unrestricted_Access);
   On_Context_Menu_Handler (Main_Window, Do_Context_Menu'Unrestricted_Access);

   OEM.GWindows.Application.Message_Loop;
end Tutorial11;
```

Обучающая программа №12 – Шрифты Windows.

Вы можете заметить, что шрифт в программы №5 отличается от большинства Windows приложений. Это связано с тем, что по умолчанию, вновь созданному ОКНУ назначается шрифт System. Пакет **OEM.GWindows** позволяет назначить любой шрифт доступный в ОС Windows. Во время своего создания, управляющий элемент берет шрифт родительского окна, но это можно изменить в любое время.

Добавим в Обучающую программу №5 после строки создания ОКНА (процедура **Create**) следующий текст программы:

```
declare
   Window_Font : OEM.GWindows.Drawing_Objects.Font_Type;
begin
   --  Use Standard Windows GUI font instead of system font
   OEM.GWindows.Drawing_Objects.Create_Stock_Font
  (Window_Font, OEM.GWindows.Drawing_Objects.Default_GUI);
   Set_Font (Main_Window, Window_Font);
end;
```

Если необходимо изменить шрифт для отдельного элемента, то соответственно меняется объект окна и место в программе по установке шрифта. Например:

```
Set_Font (Disp_Button, Window_Font);
```

Обучающая программа №13 – Z упорядочивание.

Последовательность, в которой управляющие элементы появляются на экране, легко изменяется с помощью процедуры-метода **Order.** Программа №13 также демонстрирует, что типы пакета **OEM.GWindows** подобно как и любые другие типа Ada могут быть использованы при определении массивов, записей и т.д. процедуры-метода **Order** также изменяет последовательность перехода клавише TAB (табуляции) от одного управляющего элемента к другому. Текст Обучающей программы №13 приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Windows; use OEM.GWindows.Windows;
with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
with OEM.GWindows.GStrings; use OEM.GWindows.GStrings;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial13 is
   pragma Linker_Options ("-mwindows");
```

```
      subtype Window_Number_Type is Positive range 1 .. 5;

      type Window_Array_Type is array (Window_Number_Type'Range) of
        OEM.GWindows.Windows.Window_Type;

      Main_Window    : OEM.GWindows.Windows.Main.Main_Window_Type;
      Windows        : Window_Array_Type;
      Next_Button    : OEM.GWindows.Buttons.Button_Type;
      Current_Window : Window_Number_Type := Window_Number_Type'Last;

      procedure Do_Click
        (Window : in out OEM.GWindows.Base.Base_Window_Type'Class)
      is
      begin
         Order (Windows (Current_Window), OEM.GWindows.Base.Bottom);
         if Current_Window = Window_Number_Type'Last then
           Current_Window := Window_Number_Type'First;
         else
           Current_Window := Window_Number_Type'Succ (Current_Window);
         end if;
      end Do_Click;

   begin
      Create (Main_Window, "Z-Order Window", Width => 200, Height => 100);
      Visible (Main_Window, True);
      Center (Main_Window);

      for N in Window_Array_Type'Range loop
         Create (Windows (N), To_GString_From_String ("TEST" & N'Img),
                Width => 175, Height => 75);
         Visible (Windows (N));
      end loop;

      Create (Next_Button, Main_Window, "&Next", 10, 10,
            Client_Area_Width (Main_Window) - 20,
            Client_Area_Height (Main_Window) - 20);
      On_Click_Handler (Next_Button, Do_Click'Unrestricted_Access);

      OEM.GWindows.Application.Message_Loop;
   end Tutorial13;
```

Обучающая программа №14 – Родительское ОКНО.

Пакет OEM.GWindows имеет возможности ссылаться на родительское ОКНО из дочернего или управлять им и его обработчиком событий и даже переопределить дочернее окно в родительское со всеми его атрибутами и обработчиками событий. Текст Обучающей программы №14 приведен ниже:

```
   with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
   with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
   with OEM.GWindows.Base;
   with OEM.GWindows.Application;

   procedure Tutorial14 is
      pragma Linker_Options ("-mwindows");

      Window1     : Main_Window_Type;
      Window2     : Main_Window_Type;
      Jump_Button : Button_Type;

      procedure         Do_Click         (Window         :         in         out
OEM.GWindows.Base.Base_Window_Type'Class) is
         use OEM.GWindows.Base;
      begin
         if Text (Parent (Window).all) = "Window 1" then
```

```
         Parent (Window, Window2);
      else
         Parent (Window, Window1);
      end if;
   end Do_Click;

begin
   Create (Window1, "Window 1", Width => 100, Height => 100);
   Visible (Window1, True);

   Create (Window2, "Window 2",
           Top   => Top (Window1),
           Left  => Left (Window1) + 200,
           Width => 100, Height => 100);
   Visible (Window2, True);

   Create (Jump_Button, Window1, "Jump!", 10, 10,
           Client_Area_Width (Window1) - 20,
           Client_Area_Height (Window1) - 20);
   On_Click_Handler (Jump_Button, Do_Click'Unrestricted_Access);

   OEM.GWindows.Application.Message_Loop;
end Tutorial14;
```

Обучающая программа №15 – Захват управления от МЫШИ.

В ОС Windows, обычно имеется возможность захвата управления от устройства ввода координат МЫШЬ только в внутри ОКНА. Посредством использования процедур-методов Capture и Release в обработчике событий нажатия и отпускания клавиши МЫШИ, вы можете захватить управление событием перемещения мыши и за пределами области вывода ОКНА (ОС Windows позволяет это сделать, только по событию клавиши МЫШИ нажата). Текст Обучающей программы №15 приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.GStrings; use OEM.GWindows.GStrings;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial15 is
   pragma Linker_Options ("-mwindows");

   Main_Window : Main_Window_Type;
   X_Label     : Label_Type;
   Y_Label     : Label_Type;

   procedure Do_Mouse_Move
     (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
      X      : in     Integer;
      Y      : in     Integer;
      Keys   : in     OEM.GWindows.Windows.Mouse_Key_States)
   is
   begin
      Text (X_Label, OEM.GWindows.GStrings.Image (X));
      Text (Y_Label, OEM.GWindows.GStrings.Image (Y));
   end Do_Mouse_Move;

   procedure Do_Left_Mouse_Button_Down
     (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
      X      : in     Integer;
      Y      : in     Integer;
      Keys   : in     OEM.GWindows.Windows.Mouse_Key_States)
   is
   begin
      Capture_Mouse (Main_Window);
```

```
      end Do_Left_Mouse_Button_Down;

      procedure Do_Left_Mouse_Button_Up
        (Window : in out OEM.GWindows.Base.Base_Window_Type'Class;
         X      : in     Integer;
         Y      : in     Integer;
         Keys   : in     OEM.GWindows.Windows.Mouse_Key_States)
      is
      begin
         OEM.GWindows.Base.Release_Mouse;
      end Do_Left_Mouse_Button_Up;

   begin
      Create (Main_Window, "Mouse Demo Window", Width => 200, Height => 200);
      Visible (Main_Window, True);

      Create (X_Label, Main_Window, "0", 10, 10, 40, 25, Center);
      Create (Y_Label, Main_Window, "0", 60, 10, 40, 25, Center);

      On_Mouse_Move_Handler (Main_Window, Do_Mouse_Move'Unrestricted_Access);
      On_Left_Mouse_Button_Down_Handler (Main_Window,

Do_Left_Mouse_Button_Down'Unrestricted_Access);
      On_Left_Mouse_Button_Up_Handler (Main_Window,
                                 Do_Left_Mouse_Button_Up'Unrestricted_Access);

      OEM.GWindows.Application.Message_Loop;
   end Tutorial15;
```

Обучающая программа №16 – Изменение КУРСОРА.

Для того чтобы изменить изображение курсора в пакете OEM.GWindows необходимо создать свой обработчик событий КУРСОРА On_Cursor_Change и использовать в этом обработчике событий процедуру Set_Cursor из пакета OEM.GWindows.Cursors. Текст Обучающей программы №16 приведен ниже:

```
   with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
   with OEM.GWindows.Cursors;
   with OEM.GWindows.Base;
   with OEM.GWindows.Application;

   procedure Tutorial16 is
      pragma Linker_Options ("-mwindows");

      Main_Window : Main_Window_Type;

      procedure Do_Change_Cursor
        (Window : in out OEM.GWindows.Base.Base_Window_Type'Class)
      is
         use OEM.GWindows.Cursors;
      begin
         Set_Cursor (Load_System_Cursor (IDC_CROSS));
      end Do_Change_Cursor;

   begin
      Create (Main_Window, "Cursor Window", Width => 200, Height => 200);
      Visible (Main_Window, True);
      On_Change_Cursor_Handler (Main_Window,
                                 Do_Change_Cursor'Unrestricted_Access);

      OEM.GWindows.Application.Message_Loop;
   end Tutorial16;
```

Обучающая программа №17 – Создание ДИАЛОГА.

Создание ДИАЛОГОВОГО ОКНА в OEM.GWindows просто. Достаточно использовать процедуру Show_Dialog для его отображения на экране вместо установки свойства видимости ОКНА в TRUE. Если использовать тип управляющего элемента кнопки Dialog_Button_Type, то можно установить код возврата из диалога ID по нажатию этой кнопки. Текст Обучающей программы №17 приведен ниже:

```
with OEM.GWindows.GStrings.IO; use OEM.GWindows.GStrings.IO;

with OEM.GWindows.Windows; use OEM.GWindows.Windows;
with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.Application;
with OEM.GWindows.Constants; use OEM.GWindows.Constants;

procedure Tutorial17 is
   Dialog        : Window_Type;
   OK_Button     : Default_Button_Type;
   Cancel_Button : Cancel_Button_Type;
   More_Button   : Dialog_Button_Type;

   ID_MORE       : constant := 101;

   Result        : Integer := ID_MORE;

begin
   while Result = ID_MORE loop
     Create_As_Dialog (Dialog, "My Dialog Window",
                       Width => 200, Height => 100);

     Create_Label (Dialog, "Have you had enough yet?",
                   10, 10,
                   Client_Area_Width (Dialog) - 20,
                   25, Center);

     Create (OK_Button, Dialog, "O&k",
             10, Client_Area_Height (Dialog) - 40,
             50, 25, ID => IDOK);

     Create (Cancel_Button, Dialog, "&Cancel",
             70, Client_Area_Height (Dialog) - 40,
             50, 25, ID => IDCANCEL);

     Create (More_Button, Dialog, "&More",
             130, Client_Area_Height (Dialog) - 40,
             50, 25, ID => ID_MORE);

     Result := OEM.GWindows.Application.Show_Dialog (Dialog);
   end loop;

   if Result = IDOK then
      Put_Line ("Have a nice day!");
   else
      Put_Line ("Sorry for the trouble...");
   end if;
end Tutorial17;
```

Необходимо обратить внимание что в этом примере используется стандартная консоль (см. Главу 2 и Обучающую программу №1).

Обучающая программа №18 – Доступ к базе данных через ADO.

Пример не выводит ОКОН - это консольная программа, которая демонстрирует возможности доступа к базе данных интегрированными средствами библиотеки OEM в пакете

OEM.GWindows. Обучающая программа №18 использует COM объект ADO для доступа к OLEDB базе данных (в примере используется файл формата Microsoft Office Access 2003). Текст Обучающей программы №18 приведен ниже:

```
with OEM.COM.Initialize;
with OEM.GWindows.GStrings.IO; use OEM.GWindows.GStrings.IO;
with OEM.GWindows.Databases; use OEM.GWindows.Databases;

procedure Tutorial18n is
   Connection : Database_Type;
   Recordset  : Recordset_Type;
begin
   OEM.COM.Initialize.Initialize_COM;

   Open (Connection,
         "Provider=Microsoft.Jet.OLEDB.4.0; " &
         "Data Source=adotest.mdb");

   Open (Recordset,
         Connection,
         "SELECT * FROM People",
         Dynamic,
         Optimistic);

   while not EOF (Recordset) loop
      for N in 1 .. Field_Count (Recordset) loop
         Put_Line
           (Field_Name (Recordset, N) & " = " & Field_Value (Recordset, N));
      end loop;

      if Field_Value (Recordset, "LastName") = "TestLName" then
         Put_Line ("This record is being deleted");
         Delete (Recordset);
      end if;
      Move_Next (Recordset);
      New_Line;
   end loop;

   Add_New (Recordset);
   Field_Value (Recordset, "LastName", "TestLName");
   Field_Value (Recordset, "FirstName", "TestLName");
   Update (Recordset);

   Close (Recordset);
   Close (Connection);
end Tutorial18n;
```

Более подробно работа с базами данных будет изложена в отдельной Главе.

Обучающая программа №19 – Готовый управляющий элемент для БД.

Обучающая программа №19 использует готовый управляющий элемент Database_Control_Type. Более подробно работа с базами данных будет изложена в отдельной Главе. Текст Обучающей программы №19 приведен ниже:

```
with OEM.COM.Initialize;
with OEM.COM.IErrorInfo;

with OEM.GWindows.GStrings; use OEM.GWindows.GStrings;
with OEM.GWindows.Databases.Controls;
use OEM.GWindows.Databases.Controls; use OEM.GWindows.Databases;
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Edit_Boxes; use OEM.GWindows.Edit_Boxes;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.Message_Boxes; use OEM.GWindows.Message_Boxes;
with OEM.GWindows.Base; use OEM.GWindows.Base;
```

```
with OEM.GWindows.Application;

procedure Tutorial19n is
   Main_Window : Main_Window_Type;
   ID          : aliased Edit_Box_Type;
   LastName    : aliased Edit_Box_Type;
   FirstName   : aliased Edit_Box_Type;
   Address     : aliased Edit_Box_Type;
   City        : aliased Edit_Box_Type;
   State       : aliased Edit_Box_Type;
   Zip         : aliased Edit_Box_Type;
   Country     : aliased Edit_Box_Type;
   DB_Control  : Database_Control_Type;
begin
   OEM.COM.Initialize.Initialize_COM;

   --  Setup Controls
   Create (Main_Window, "Database Window", Width => 400, Height => 500);
   Keyboard_Support (Main_Window);
   Center (Main_Window);

   Create_As_Control (DB_Control, Main_Window, "", 0, 0, 0, 30);
   Dock (DB_Control, At_Top);

   Create_Label (Main_Window, "ID", 10, 40, 100, 25);
   Create_Label (Main_Window, "Last Name", 10, 80, 100, 25);
   Create_Label (Main_Window, "First Name", 10, 120, 100, 25);
   Create_Label (Main_Window, "Address", 10, 160, 100, 25);
   Create_Label (Main_Window, "City", 10, 200, 100, 25);
   Create_Label (Main_Window, "State", 10, 240, 100, 25);
   Create_Label (Main_Window, "Zip", 10, 280, 100, 25);
   Create_Label (Main_Window, "Country", 10, 320, 100, 25);

   Create (ID, Main_Window, "",
           120, 40, 200, 30);
   Read_Only (ID);
   Create (LastName, Main_Window, "",
           120, 80, 200, 30);
   Create (FirstName, Main_Window, "",
           120, 120, 200, 30);
   Create (Address, Main_Window, "",
           120, 160, 200, 30);
   Create (City, Main_Window, "",
           120, 200, 200, 30);
   Create (State, Main_Window, "",
           120, 240, 200, 30);
   Create (Zip, Main_Window, "",
           120, 280, 200, 30);
   Create (Country, Main_Window, "",
           120, 320, 200, 30);

   --  Setup Database and Bindings

   Open (DB_Control.Database,
         "Provider=Microsoft.Jet.OLEDB.4.0; " &
         "Data Source=adotest.mdb");

   Open (DB_Control.Recordset,
         DB_Control.Database,
         "SELECT * FROM People",
         Dynamic,
         Optimistic);

   Bind_Text_Control (DB_Control.Recordset,
                      "ID",
                      ID'Unchecked_Access,
                      Read_Only);
   Bind_Text_Control (DB_Control.Recordset,
```

```
                        "LastName",
                        LastName'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "FirstName",
                        FirstName'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "Address",
                        Address'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "City",
                        City'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "State",
                        State'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "Zip",
                        Zip'Unchecked_Access);
         Bind_Text_Control (DB_Control.Recordset,
                        "Country",
                        Country'Unchecked_Access);

         --  Get Started
         Fill_Bindings (DB_Control.Recordset);
         Dock_Children (Main_Window);
         Visible (Main_Window, True);

         OEM.GWindows.Application.Message_Loop;
      exception
         when others =>
            Message_Box ("Error",
                         To_GString_From_String
(OEM.COM.IErrorInfo.Get_IErrorInfo),
                         Icon => Error_Icon);
      end Tutorial19n;
```

Обучающая программа №20 – Упаковка и стыковка ОКОН.

Пакет **OEM.GWindows** предоставляет дополнительные гибкие средства для установки и стыковки управляющих элементов в области их отображения в родительском/главном ОКНЕ. Обучающая программа №20 показывает как их можно применять вместе. Текст Обучающей программы №20 приведен ниже:

```
with OEM.GWindows.Windows.Main; use OEM.GWindows.Windows.Main;
with OEM.GWindows.Buttons; use OEM.GWindows.Buttons;
with OEM.GWindows.Static_Controls; use OEM.GWindows.Static_Controls;
with OEM.GWindows.Edit_Boxes; use OEM.GWindows.Edit_Boxes;
with OEM.GWindows.Packing_Boxes; use OEM.GWindows.Packing_Boxes;
with OEM.GWindows.Base;
with OEM.GWindows.Application;

procedure Tutorial20 is
   Main_Window : Main_Window_Type;
   Box         : Packing_Box_Type;
   Box2        : Packing_Box_Type;
   Box3        : Packing_Box_Type;
   Box4        : Packing_Box_Type;
   Button1     : Button_Type;
   Button2     : Button_Type;
   Button3     : Button_Type;
   Button4     : Button_Type;
   Button5     : Button_Type;
   Button6     : Button_Type;
   Edit1       : Edit_Box_Type;
   Edit2       : Edit_Box_Type;
   Edit3       : Edit_Box_Type;
```

```
begin
   Create (Main_Window, "Packing Boxes in a Window",
           Width => 500, Height => 250);

   Create (Box, Main_Window, 0, 0, 100, 0, Vertical);
   Dock (Box, OEM.GWindows.Base.At_Left);

   Fill_Width (Box);
   Fill_Height (Box);
   Padding (Box, 5);
   Insets (Box, (10, 10, 10, 10));

   Create (Box2, Main_Window, 0, 0, 0, 50, Horizontal);
   Dock (Box2, OEM.GWindows.Base.At_Top);

   Fill_Width (Box2);
   Fill_Height (Box2);
   Padding (Box2, 5);
   Insets (Box2, (10, 10, 10, 10));

   Create (Box3, Main_Window, 0, 0, 100, 0, Vertical);
   Dock (Box3, OEM.GWindows.Base.At_Left);

   Fill_Width (Box3);
   Padding (Box3, 5);
   Insets (Box3, (10, 15, 10, 10));

   Create (Box4, Main_Window, 0, 0, 0, 0, Vertical);
   Dock (Box4, OEM.GWindows.Base.Fill);

   Fill_Width (Box4);
   Padding (Box4, 5);
   Insets (Box4, (10, 10, 10, 10));

   Dock_Children (Main_Window);

   Create (Button1, Box, "Button1", 0, 0, 0, 0);
   Create (Button2, Box, "Button2", 0, 0, 0, 0);
   Create (Button3, Box, "Button3", 0, 0, 0, 0);

   Pack (Box);

   Create (Button4, Box2, "Button3", 0, 0, 0, 0);
   Create (Button5, Box2, "Button4", 0, 0, 0, 0);
   Create (Button6, Box2, "Button5", 0, 0, 0, 0);

   Pack (Box2);

   Create_Label (Box3, "Field 1", 0, 0, 0, 30);
   Create_Label (Box3, "Field 2", 0, 0, 0, 30);
   Create_Label (Box3, "Field 3", 0, 0, 0, 30);

   Pack (Box3);

   Create (Edit1, Box4, "", 0, 0, 0, 30);
   Create (Edit2, Box4, "", 0, 0, 0, 30);
   Create (Edit3, Box4, "", 0, 0, 0, 30);

   Pack (Box4);

   Visible (Main_Window);
   OEM.GWindows.Application.Message_Loop;
end Tutorial20;
```

Обучающая программа №21 – Воспроизведение звуков.

Пакет **OEM.GWindows** имеет несколько процедур для воспроизведения звуков из файлов, ресурсов или звуков системных событий ОС Windows. Обучающая программа №21 показывает, как их можно применять. Текст Обучающей программы №21 приведен ниже:

```
with OEM.GWindows.Multi_Media; use OEM.GWindows.Multi_Media;

procedure Tutorial21 is
begin
   Play_Sound_From_Alias ("SystemStart");
   Play_Sound_From_File ("hello.wav");
end Tutorial21;
```

Файлы связанные событиями ОС в аплете управляющей панели (Рис. 39) будут воспроизведены. Идентификаторы событий определяются в РЕЕСТРЕ ОС (Рис. 40).

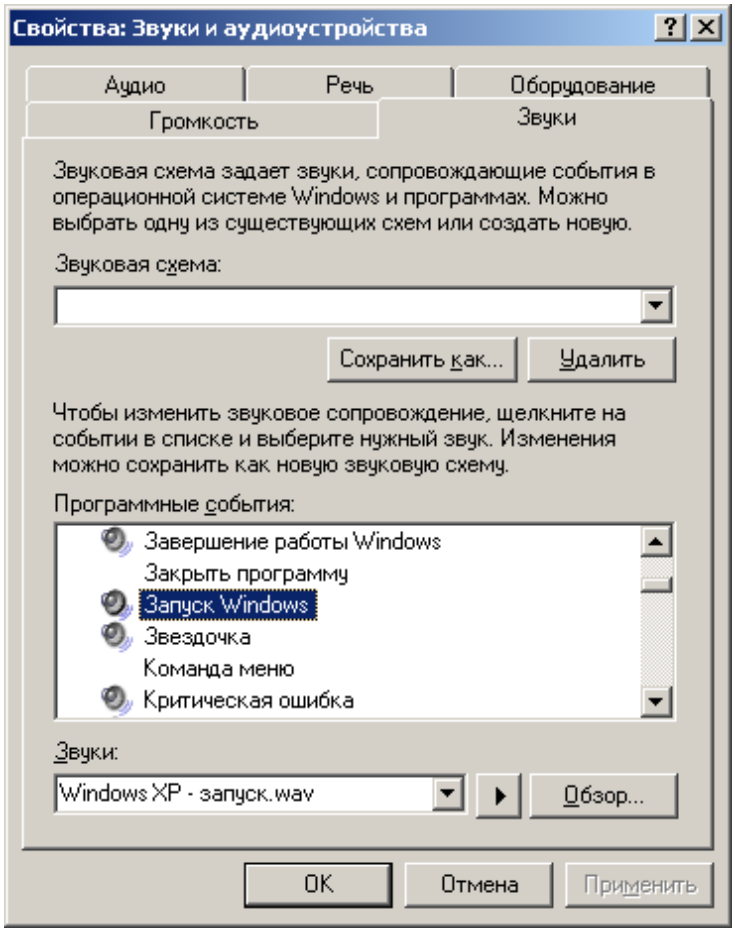

Рис. 39. Настройка звука "Запуск Windows".

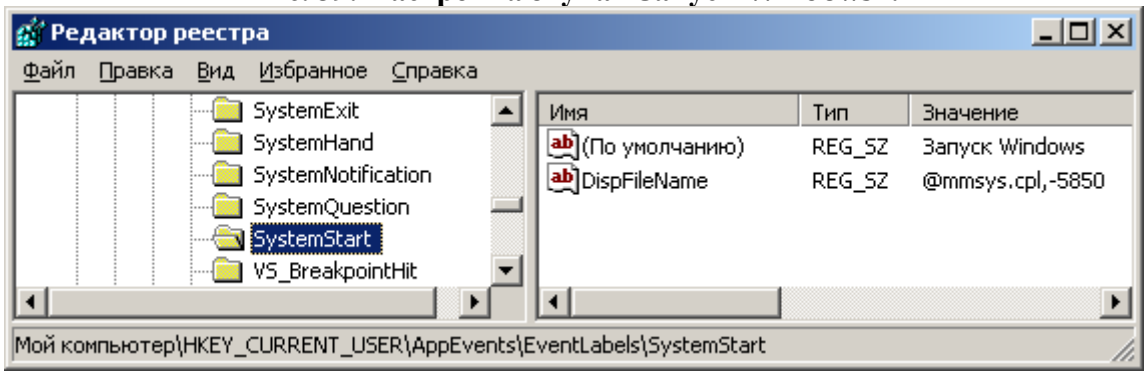

Рис. 40. В реестре ОС можно посмотреть соответствие идентификаторов событий с их наименованиями в аплете Звуки.

Тестовые примеры GWindows библиотеки OEM.

Тестовые примеры пакета GWindows библиотеки OEM размещены в каталоге \oemtools2010\ForTestGwindows. Тесты сгруппированы по функциональным характеристикам в соответствующих подкаталогах. Каждый тест пример имеет свой собственный проект для IDE GPS. При необходимости использования файла с графическими или другими дополнительными ресурсами в подкаталоге присутствует makefile. После изучения Главы 3 в целом тестовые примеры не должны вызвать сложностей в изучении. Изучение тестовых примеров может быть полезным, а их исходный текст послужить заготовками для реальных приложений с использованием пакета GWindows библиотеки OEM. В таблице 1 приведены сводные данные по тестовым примерам пакета GWindows библиотеки OEM.

Таблица 1

Проект	**Каталог и программы**	**Категория**	**Краткое описание**
control_test.gpr	\oemtools2010\ForTestGwindows\controls 1) control_test.exe 2) gtoolbar.exe 3) tab_test.exe 4) win_controls.exe	Управляющие элементы	Наиболее полный пример использования управляющих элементов и их взаимосвязей. Необходим файл ресурсов.
db_view.gpr	\oemtools2010\ForTestGwindows\databases 1) db_view.exe	Графическая форма работы с базой данных	Тот же обучающий пример tutorial19n выполненный в отдельном проекте
dlgtest.gpr	\oemtools2010\ForTestGwindows\dialogs 1) dialog_example.exe 2) dlgtest.exe 3) getdir.exe	Стандартные и созданные через файл описания ресурсов диалоговые окна	Раскрывает некоторые аспекты использования диалогов, в том числе созданных через файл ресурсов
call_gw_dll.gpr gw_in_a_dll.gpr	\oemtools2010\ForTestGwindows\dll 1) call_gw_dll.exe 2) gw_in_a_dll.dll	Вызов DLL и создание DLL с процедурами вывода графического окна	Полный комплект образец для создания собственной DLL и её тестирования. Требуется makefile.
scribble2.gpr	\oemtools2010\ForTestGwindows\drawing 1) scribble2.exe	Рисование в окне "мышью"	Рисование в окне без использования DROW PANEL, но с двойным буферированием.
scribble.gpr	\oemtools2010\ForTestGwindows\drawing	Рисование в	Рисование в

Проект	**Каталог и программы**	**Категория**	**Краткое описание**
	1) scribble.adb	окне "мышью"	окне с использованием DROW PANEL
gbmanager.gpr	\oemtools2010\ForTestGwindows\gbmanager 1) gbmanager.exe	Приложение, управляющие элементы, файл ресурсов для иконки приложения. Диалоговое окно выбора каталога.	Законченное приложение с графическим интерфейсом. Аналог утилит bindcom.exe и comscope.exe. Требуется makefile.
gnatreg.gpr	\oemtools2010\ForTestGwindows\gnatreg 1) gnatreg.exe	Приложение, управляющие элементы, файл ресурсов для иконки приложения, меню и клавиш ускорения выбора пунктов меню.	Законченное приложение утилита с графическим интерфейсом. Регистрирует библиотеки типа OEM и их ресурсы в операционной системе Windows.. Требуется makefile.
mdi_example.gpr	\oemtools2010\ForTestGwindows\mdi	Приложение с MDI окнами	Простой текстовый редактор с MDI интерфейсом. Требуется makefile.
cap_test.exe	\oemtools2010\ForTestGwindows\mouse 1) cap_test.exe	Обработка событий мыши и курсора в пределах и за пределами окна	При перемещении курсора "мыши" в пределах окна отображаются его координаты в окне. По нажатию левой клавиши изменяется курсор на перекрестие и отображаются координаты экрана монитора.
droptest.gpr	\oemtools2010\ForTestGwindows\mouse 1) droptest.exe	Обработка событий мыши и курсора в	Выбранное имя файла в стандартном окне Windows

Проект	**Каталог и программы**	**Категория**	**Краткое описание**
		пределах и за пределами окна	Проводника при нажатой левой кнопки мыши перетаскивается в окно программы. Отпустив кнопку мыши выводится сообщение с именем файла в Message_Box.
mscal_test.gpr	\oemtools2010\ForTestGwindows\mscaltest 1) mscal_test.exe C:\Program Files\Microsoft Office\OFFICE11\ 1) MSCAL.OCX 120192 03.05.07 16:52 Русская версия MS Office 2003	Вызов методов COM объектов OCX	Создание интерфейса объекта с помощью программы gbmanager.exe. Создание объекта в программе и вызов его методов двумя различными способами. Сам объект не отображается в окне приложения.
mybutton.gpr	\oemtools2010\ForTestGwindows\ownerdraw 1) mybutton.exe	Управляющий элемент Кнопка	Создание оригинального внешнего вида Кнопки.
print_hello.gpr	\oemtools2010\ForTestGwindows\printing 1) print_hello.exe	Вывод на принтер	Вывод на принтер по умолчанию по нажатию кнопки.
dock_test.gpr	\oemtools2010\ForTestGwindows\simple 1) dock_test.exe	Размещение управляющих элементов в окне	Пример размещения управляющих элементов в главном окне.
form_example.gpr	\oemtools2010\ForTestGwindows\simple 1) form_example.exe	Обработка событий в новом типе окна	Пример создания собственного типа окна в отдельном пакете и приложения использующего этот тип окна как основное

Проект	**Каталог и программы**	**Категория**	**Краткое описание**
			окно приложения в виде формы. Требуется makefile для меню и акселераторов клавиш.
hello_world.gpr	\oemtools2010\ForTestGwindows\simple 1) hello_world.exe	Минимальная программа с графическим интерфейсом	Обучающие примеры tutorial1 и tutoria2 в одной программе.
key_test.gpr	\oemtools2010\ForTestGwindows\simple 1) key_test.exe	Обработка событий нажатия клавиш	Код нажатой клавиши выводится на текстовую консоль.
menu_example.gpr	\oemtools2010\ForTestGwindows\simple 1) menu_example.exe	Обработка событий динамически созданного меню	Создание динамически иерархии меню приложения и обработка событий меню
point_test.gpr	\oemtools2010\ForTestGwindows\simple 1) point_test.exe	Обработка событий нажатия левой клавиши мыши	Координаты курсора мыши выводятся на текстовую консоль по нажатию левой клавиши мыши
task_dialogs.gpr	\oemtools2010\ForTestGwindows\tasks 1) task_dialogs.exe	Создание и запуск задач	По нажатию Кнопки создаются три модальных диалога, каждый из которых обслуживается своей задачей.
task_windows.gpr	\oemtools2010\ForTestGwindows\tasks 1) task_windows.exe	Обработка событий окна в задачах	Два вида реализации обработки событий окна в задачи. В первом каждая задача имеет свой явно прописанный индивидуальный цикл обработки

Проект	**Каталог и программы**	**Категория**	**Краткое описание**
			событий в окне. Во втором случае окна создаются отдельной задачей "ФАБРИКОЙ ОКОН" в которой реализовано динамическое созданных окон.
thread_test.gpr	\oemtools2010\ForTestGwindows\tasks 1) thread_test.exe	Управление контекстом окна из нескольких задач	Две задачи параллельно управляют контекстом одного окна. Первая – наименование, вторая выводит прямоугольники различного цвета и размера на полотно окна.

ЛИТЕРАТУРА

1. Гавва А. Е. “Адское” программирование. Ada-95. Компилятор GNAT: [Электрон. ресурс]. – http://www.ada-ru.org.
2. Рыбин С. И., Годунко В. GNAT Pro – Ада – технология для серьезных проектов. Семинар, выставка PTS-2009, Минск. [Электрон. ресурс]. – http://www.mediascan.by/index.files/GNAT-AdaCore.pdf .
3. Мищенко В. О., Гахов А.В. Обучение проектированию систем и основам параллельных вычислений на базе языка Ада в Харьковском национальном университете. Семинар, выставка PTS-2009, Минск. [Электрон. ресурс]. – http://www.mediascan.by/index.files/adaedu.pdf .
4. Рыбин С. И., Годунко В. Ада – перспективы использования в индустрии и образовании. Семинар, выставка PTS-2009, Минск. [Электрон. ресурс]. – http://www.mediascan.by/index.files/Ada-AdaCore.pdf .
5. Киркоров С. И., Киркорова Л. С. Параллельные алгоритмы математических моделей: исследование локальности и применение языка Ada. Ж-л: ВІСНИК Харківського національного університету імені В.Н. Каразіна, №863 Серія: Математичне моделювання. Інформаційні технології. Автоматизовані системи управління, Випуск 12, стр. 129-142, Харків, 2009.
6. The GNAT Academic Program [Электрон. ресурс]. – http://www.adacore.com/home/academia/
7. Библиотека OEM [Электрон. ресурс]. – http://www.mediascan.by/index.files/CD_Ada-2009_OEM.zip