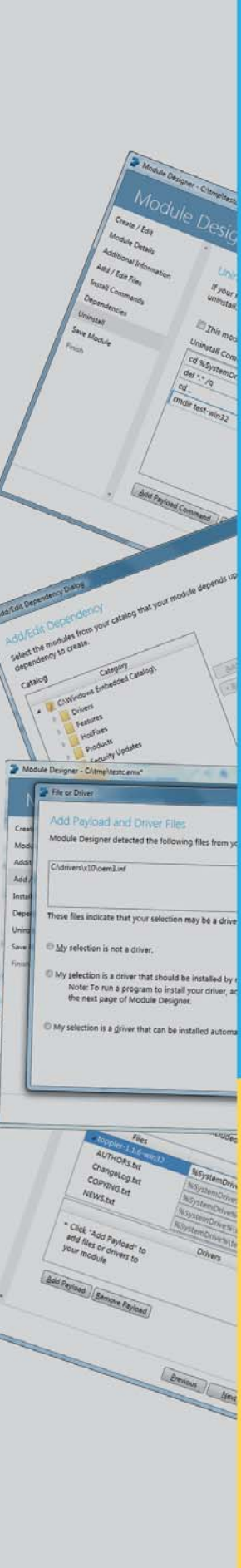
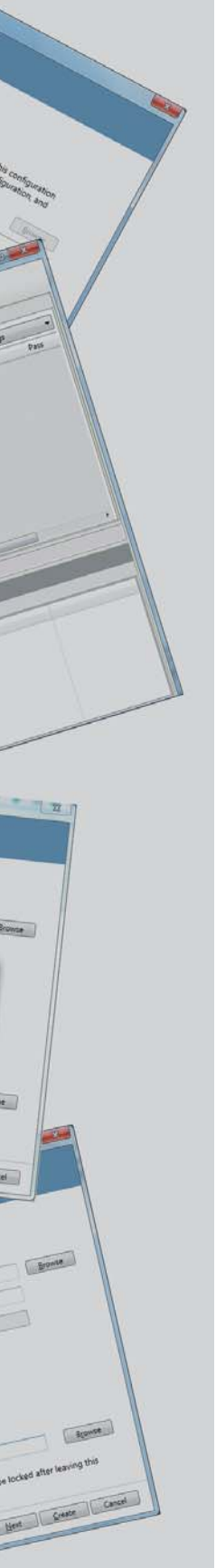


MODULE DESIGNER. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ МОДУЛЕЙ

СЕРГЕЙ ДАВЫДОВ
info@quarta.ru

Возможность создания пользовательских модулей предоставляет разработчикам большие преимущества при проектировании устройств специального назначения, так как существенно упрощает процесс массового создания и тиражирования разрабатываемых образов в случае использования предопределенного набора ПО и выполнения миграции образа на разные платформы. В статье рассматривается, что такое модуль, какие типы модулей существуют в Windows Embedded 8 Standard, в каких отношениях они могут находиться, а также процесс создания пользовательского модуля начиная от момента его разработки до импортирования в каталог Standard 8.



Подход, при котором операционная система для конкретной ЭВМ собирается из функциональных блоков для решения конкретной (четко определенной) задачи, существует со времен мэйнфреймов. Идея создания ОС для систем, построенных на базе микропроцессоров, которую можно было бы для получения требуемого функционала собрать из «кубиков» (компонентов-пакетов-модулей), представляющих собой функциональные элементы настольной ОС, была впервые реализована компанией Microsoft в Windows NT Embedded, о появлении которой было объявлено в 1999 г. Базируясь на Windows NT 4.0, эта ОС позволяет проектировщику создавать свой собственный образ ОС, используя для этого компоненты Windows NT 4.0. При этом была предусмотрена возможность разрабатывать и использовать в составе ОС свои собственные компоненты. Для этого в поставку включались как средства для разработки образа ОС, так и средства для создания своих собственных компонентов. При таком подходе компонент представляет собой некий минимальный (элементарный) функциональный элемент ОС, выполняющий конкретную функцию.

Следующим логичным шагом был выпуск компонентных версий Windows XP Professional: Windows XP Embedded (на базе Windows XP Professional SP1) и последовавшей за ней Windows Embedded Standard 2009 (Windows XP Professional SP3). Следует отметить, что начиная именно с этих версий компонентные ОС становятся популярными среди производителей специализированных устройств и находят широкое применение. В июле 2010 г. была выпущена Windows Embedded Standard 7, основанная на Windows 7 Ultimate. На данном этапе компоненты, которые теперь стали называться пакетами, несколько «укрупнились» и существенно упростился сам процесс сборки образа ОС в соответствии с предъявляемыми требованиями. Но исчезла возможность создания собственных компонентов и, как следствие, некоторая гибкость при проектировании образа.

И, наконец, в 2013 г. была выпущена Windows 8 Embedded Standard (или Standard 8) — новейшая операционная система линейки Standard, в основе которой лежит Windows 8 Professional.

БАЗОВЫЕ ПОНЯТИЯ. МОДУЛЬ, КАТАЛОГ

В Windows 8 Embedded Standard появилось новое название для «строительных блоков» ОС — модуль. Модуль — это наименьшая функциональная единица операционной системы, которая может быть добавлена в дизайн. Модули могут содержать пакеты и файлы с инструкциями для их установки в системе, необходимые, к примеру, для обеспечения функционирования какого-либо устройства или группы устройств (например, беспроводных сетей). Они могут функционировать самостоятельно или быть зависимыми от других модулей, то есть требовать для своего функционирования присутствия в образе других модулей, от которых они зависят (например, модуль, содержащий приложение, написанное на #C, будет зависим от модуля .NET). Microsoft поддерживает работу с «подписанными» модулями Standard 8 для обеспечения безопасности. Также, если это необходимо, можно подписывать модули сторонних разработчиков и самостоятельно разработанные.

Для создания и построения образа Standard 8 можно применять инструменты Image Builder Wizard (IBW) и Image Configuration Editor (ICE), которые, используя содержимое каталога компонентов Windows Embedded 8 Standard (модули — файлы .emd, пакеты компонентов — обычно файлы .cab или .msu), позволяют включать в образ только те функциональные возможности, которые требуются. Исключение из этого правила — наверное, только модуль Embedded Core, являющийся основой для любого образа ОС Standard 8 и поэтому присутствующий в каждом из них. Иных ограничений по наличию других модулей в разрабатываемом образе нет.

Добавление модулей возможно как на этапе проектирования и создания собственного образа ОС, так и после его развертывания, или в уже собранный образ с использованием утилиты Deployment Image Servicing and Management (DISM).

Остановимся более подробно на типах модулей, которые используются в Windows 8 Embedded Standard.

ТИПЫ МОДУЛЕЙ В STANDARD 8

В Standard 8 выделены пять основных типов модулей. Рассмотрим их.

Модуль ядра (Embedded Core Module)

Этот модуль является как бы «отправной точкой» для любого создаваемого образа ОС как для версий x86, так и для x64. Он содержит драйверы и минимальные функциональные возможности, необходимые для запуска целевого устройства, и обеспечивает базовую поддержку безопасности, обслуживания и сетей. Как следствие, он представляет собой, фактически, минимальный по размеру образ, возможный для создания и развертывания. При создании любого образа этот модуль автоматически добавляется Image Configuration Editor (ICE) в новый файл конфигурации. Удалить какую-либо часть данного модуля невозможно. Следует отметить, что Embedded Core не содержит никаких шрифтов и языковых ресурсов, поэтому на этапе разработки необходимо включить в образ хотя бы один языковой модуль.

Embedded Core включает в себя поддержку командной строки, но не включает ни полный графический интерфейс пользователя (GUI), ни какое-либо Windows-приложение, предполагающее его наличие.

Функциональные модули (Feature Modules)

Эти модули представляют собой группу технологического функционирования. К ним относятся, например, оболочки и поддержка беспроводных сетей, которые можно добавить к своему образу во время его разработки или после развертывания.

Модули драйверов (Driver Modules)

Эти модули включают в себя драйверы, необходимые для обеспечения функционирования аппаратного обеспечения системы и подключенных периферийных устройств или группы устройств под управлением Windows Embedded 8 Standard (Standard 8).

Языковые модули (Language Modules)

Языковые модули предоставляют шрифты и другие ресурсы, необходимые для представления всего пользовательского интерфейса (UI) операционной системы Standard 8 или его части на определенном языке. В разрабатываемом образе ОС необходимо использовать хотя бы один языковой модуль из списка основных и/или дополнительных поддерживаемых языков.

Пользовательские модули (Custom Modules)

В Windows Embedded 8 Standard была реинкарнирована возможность разработки собственных модулей. Они ведут себя подобно функциональным модулям, за исключением того, что содержат пользовательские файлы и приложения, с помощью которых можно производить изменения реестра и выполнять команды, помогающие устанавливать собственное программное обеспечение. Утилита Module Designer, входящая в состав инструментария Standard 8, предназначена для создания пользовательских модулей и последующего их импорта в каталог, откуда они могут быть добавлены в разрабатываемый образ.

ОТНОШЕНИЯ МОДУЛЕЙ

Модули, используемые при разработке образа ОС, могут функционировать как самостоятельно, так и быть зависимыми, то есть требовать присутствия в образе других модулей для обеспечения своего полноценного функционирования, или даже конфликтовать с другими модулями. Во многих случаях эти отношения управляются автоматически, но в некоторых случаях требуется вмешательство проектировщика. Рассмотрим эти взаимоотношения.

Зависимости (Dependencies) — это отношения, при которых один модуль зависит от функционирования другого. Неразрешение зависимости может привести к тому, что разрабатываемый образ будет функционировать неправильно. Зависимости могут применяться к конкретным модулям или к одному или нескольким модулям из группы. Например, необходимо включать в образ хотя бы один языковой модуль из группы всех доступных языковых модулей. Модули, требуемые для функционирования зависимого модуля, разделяются на:

- *Необходимые*, предоставляющие функциональные возможности, требуемые для обеспечения работоспособности зависимого модуля. Модулю может требоваться один или несколько модулей из группы, от которых он зависит.
- *Дополнительные*, обеспечивающие дополнительные функциональные возможности. Их присутствие в образе не является обязательным, но при их отсутствии зависимый модуль не будет обеспечивать всего предполагаемого функционала.

Конфликты (Conflicts) — это отношения, при которых функциональность одного модуля конфликтует с функциональностью другого. Конфликты не могут быть включены в образ и должны быть разрешены. Встречаются условные конфликты, при которых модуль конфликтует с другим только при определенных условиях.

ПРОЦЕСС СОЗДАНИЯ И ИСПОЛЬЗОВАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО МОДУЛЯ

Возможность создания пользовательских модулей, возрожденная в Standard 8, предоставляет разработчикам большую гибкость при создании образов ОС. Очевидно, что если устройство планируется к производству в единичном экземпляре, то нет особого смысла затрачивать силы и средства для создания какого-либо специального модуля. С другой стороны, если планируется массовое производство устройств с аналогичной архитектурой и/или единым набором программного обеспечения и драйверов, то для упрощения процесса создания образов целесообразно разработать специальные пользовательские модули. С их помощью можно добавлять файлы и программы и указывать, как они будут устанавливаться на устройстве, автоматизировать процесс развертывания пользовательского ПО и драйверов на целевом устройстве, запускать скрипты, включать элементы кастомизации и т. п. Например, для серии устройств требуется инсталлировать на каждое из них специфическую программу, требующую выполнения определенной последовательности команд при развертывании ОС и включение в образ экранной заставки с логотипом компании, позволяющей явно идентифицировать производителя решения. При этом подходе единовременные затраты на разработку такого модуля с лихвой будут компенсированы при разработке образов ОС с его использованием.

Весь процесс, начиная от создания модуля до его использования в ОС на целевом устройстве, можно разбить на три больших этапа (рис. 1):

- Создание модуля с использованием инструмента Module Designer (MOD). На этом этапе определяется назначение модуля, приложения и файлы, необходимые для его функционирования, процесс его

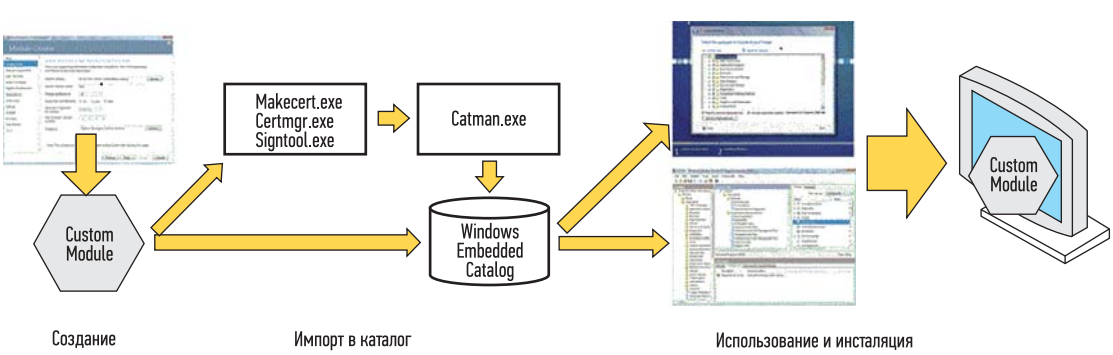


РИС. 1. ◄ Схема процесса создания модуля

инсталляции/деинсталляции, зависимости и т. п. Результатом выполнения данного этапа является модуль, который можно импортировать в каталог для дальнейшего использования. Более подробно этот этап будет описан ниже.

- Импортирование созданного модуля в каталог. Файл модуля (.emd), являющийся результатом работы Module Designer, может быть импортирован в каталог непосредственно с его помощью, что позволит в дальнейшем легко интегрировать его с другими модулями ОС. С другой стороны, как отмечалось, в Standard 8 обеспечивается поддержка «подписанных» модулей. Это правомерно и для разработанных пользователем модулей. Для этого необходимо сначала создать сертификат и подписать созданный модуль. И только после этого модуль должен быть импортирован в каталог для дальнейшего использования. Это можно сделать, используя утилиты командной строки makecert.exe, certmgr.exe, signtool.exe и catman.exe.
- Непосредственно использование модуля при создании образа ОС с последующей инсталляцией его на целевом устройстве, что может быть сделано с использованием Image Configuration Editor (ICE) или Image Builder Wizard (IBW).

Процесс разработки модуля в Module Designer представляет собой выполнение последовательности шагов, а фактически — последовательное прохождение мастера создания модуля.

ШАГ 1. СОЗДАНИЕ МОДУЛЯ

При запуске Module Designer экран **Create/Edit** предлагает выбор: создать новый модуль или внести изменения в уже существующий (рис. 2). После чего на странице **Module Details** производится ввод информации: в поле **Catalog** указывается местонахождение каталога Standard 8, в поле **Friendly name** — имя модуля, как он будет виден в ICE, а поле **Category** определяет, в какой группе он будет отображаться. Раскрывающийся список **Primary Architecture** позволяет выбрать архитектуру устройства, для которой изначально будет использоваться разрабатываемый модуль. Если целевое устройство поддерживает архитектуру x64 и требуется ее поддержка в дополнение к x86, необходимо указать это в установке **Supported architectures** (рис. 3). Ниспадающее меню **Minimum supported OS version** позволяет указать, начиная с какой версии операционной системы модуль будет поддерживаться.

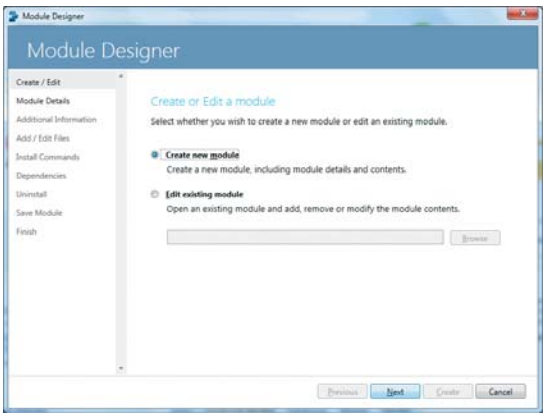


РИС. 2. Окно Create/Edit в Module Designer

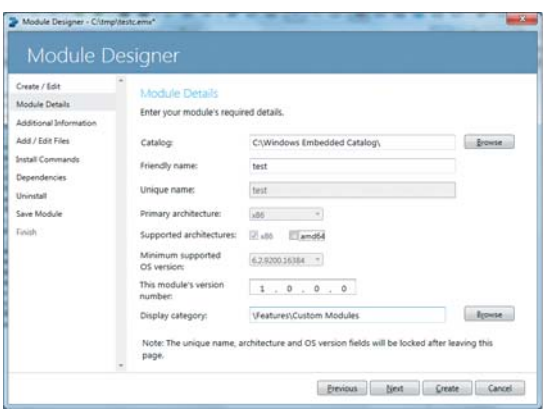
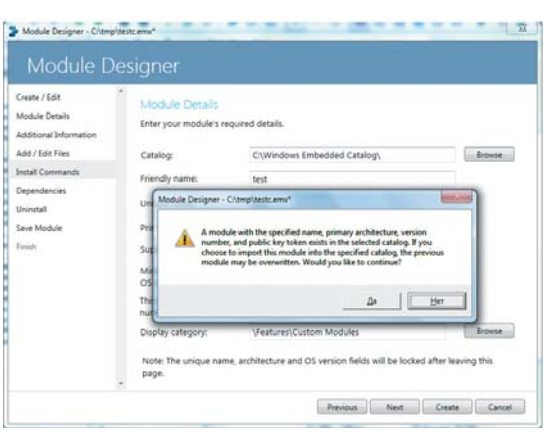


РИС. 3. Окно ввода информации о модуле

РИС. 4. Окно предупреждения о перезаписи информации



В случае если производится редактирование существующего модуля, разработчику будет выдано предупреждение о возможной перезаписи предварительно сохраненной информации (рис. 4). Следующая страница — **Additional Module Information** — позволяет ввести описание модуля, информацию об его авторе и ссылку на страницу поддержки.

ШАГ 2. ДОБАВЛЕНИЕ ФАЙЛОВ

Когда основная информация о модуле введена, можно определить файлы, которые необходимо скопировать на целевое устройство при установке операционной системы (страница **Add/Edit Files**). В модуль можно добавлять файлы любого типа, соответствующие требованиям устройства, например:

- инсталляторы программного обеспечения: *InstallApplication.msi;*
- папки приложений: *C:\ProgramFiles\CustomApp;*
- драйверы устройств: *VideoDriver.inf;*
- другие файлы: *CustomerBackground.bmp, Picture.png.*

Чтобы добавить файл или каталог в модуль, надо нажать кнопку **Add Payload**, выбрать соответствующий файл и нажать **Open** или **Folder Open** (рис. 5). Выбранные файлы будут отображены в панели **Files**. При этом в панели **Destination Path** будет указано, в какой каталог на целевой системе они будут помещены. При добавлении файла драйвера (.inf) Module Designer запросит, как его интерпретировать и выполнения каких действий он требует (рис. 6). Если какой-либо файл был ошибочно добавлен в модуль, его можно удалить, выбрав в панели **Files** и нажав кнопку **Remove Payload**.

ШАГ 3. СОЗДАНИЕ КОМАНД ИНСТАЛЛЯЦИИ

Если программному обеспечению целевого устройства требуется выполнение каких-либо действий, которые должны быть выполнены в дополнение к копированию файлов, они могут быть определены на этом шаге, например:

- запуск установщика приложения: *InstallApplication.msi;*

- запуск приложения с параметрами: *msiexec /i InstallApplication.msi /quiet;*
- запуск системной команды: *shutdown /r.*

Добавление команд производится нажатием кнопки **Add Custom Command**.

ШАГ 4. ДОБАВЛЕНИЕ ЗАВИСИМОСТЕЙ

Как отмечалось, зависимость — это способ описать, что модулю требуется другой модуль для его правильного функционирования. Использование зависимостей облегчает добавление модуля в конфигурацию без необходимости запоминать, что именно ему требуется для правильной работы. Все указанные зависимости будут автоматически включены в конфигурации ОС, когда модуль будет выбран в ICE или IBW. Например, для воспроизведения видеофайла на целевом устройстве требуется наличие Windows Media Player.

Определение зависимостей является достаточно кропотливым процессом, порой сопряженным со множеством проб и ошибок, требующим определенных знаний и навыков, а подчас и применения специального инструментария. Существенно облегчить и несколько автоматизировать его может утилита Dynamic Dependency Analyzer (DDA) — новый инструмент командной строки, входящий в Windows Embedded 8 Standard Toolkit. Он предназначен для анализа приложения во время его работы и выявления зависимостей. Сам процесс анализа сводится к эмуляции всех возможных действий пользователя, работающего с данным приложением, когда оно мониторится DDA. В результате создается файл журнала (.txt), в который DDA записывает список обнаруженных зависимостей и который может быть импортирован в Module Designer (MOD) при создании пользовательского модуля.

Следует обратить внимание что анализ приложения для выявлении зависимостей следует производить на компьютере, имеющем аппаратную конфигурацию как у целевого компьютера, а используемый модуль DDA должен соответствовать архитектуре используемого процессора

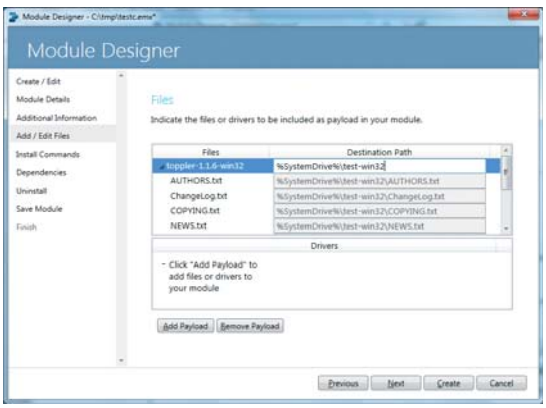


РИС. 5. ▲ Окно добавления файла или каталога в модуль

(DynamicAnalyzer-x86.exe для x86 и DynamicAnalyzer-x64.exe для x64 соответственно). Для получения наиболее точных результатов анализа

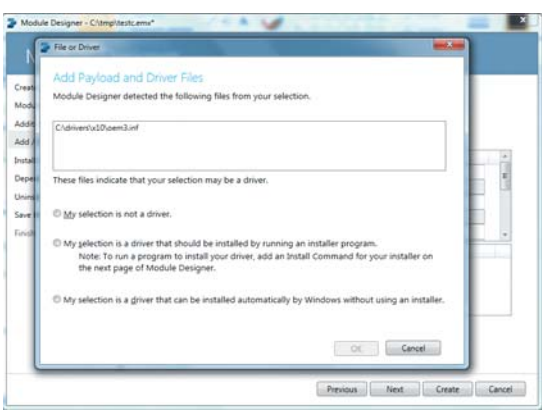
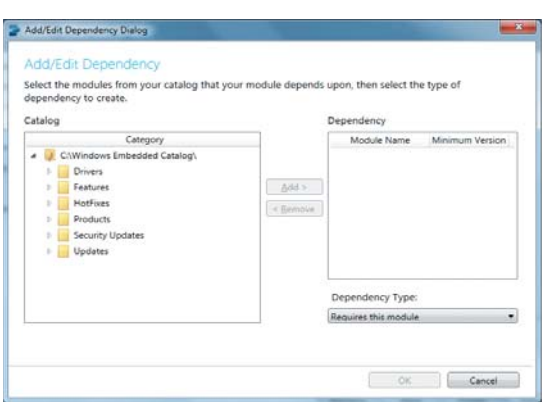


РИС. 6. ▲ Окно запроса при добавлении драйвера

РИС. 7. ▼ Окно добавления зависимостей «вручную»

рекомендуется запускать DDA только на Windows 8 или на полном образе Standard 8. Также следует учесть, что зависимости могут варьироваться от того, запущено ли приложение с правами администратора. Во многих случаях приложения ведут себя одинаково, независимо от того, с какими



привилегиями они были запущены. Но иногда список зависимостей может различаться при запуске приложения с большими привилегиями. Поэтому перед проведением анализа необходимо уточнить, с какими правами приложение будет запускаться на целевом устройстве.

Как только файл журнала создан, обнаруженные зависимости можно импортировать в Module Designer, нажав **Import Dependencies List** и выбрав соответствующий файл. В диалоговом окне **Dependency Scan Results** во вкладке **Mapped Dependencies** будут отображены все разрешенные зависимости. Чтобы отметить, какие модули будут использованы для разрешения зависимостей, надо установить флажок рядом с соответствующим модулем. Все неразрешенные зависимости отображаются на вкладке **Unmapped Dependencies**. Решение по ним принимает разработчик.

Также имеется возможность определения зависимостей «вручную» (рис. 7). Для добавления зависимого модуля надо нажать кнопку **Add**, в появившемся диалоговом окне **Add/Edit Dependency** указать, от каких модулей зависит создаваемый, и определить тип этих зависимостей.

ШАГ 5. СОЗДАНИЕ КОМАНД ДЕИНСТАЛЛЯЦИИ

По умолчанию при деинсталляции из хранилища удаляется только файл конфигурации модуля. Если программному обеспечению требуется, чтобы при удалении модуля выполнялись какие-либо иные действия, например запуск сценария деинсталляции для удаления файлов модуля с диска, удаление любых пользовательских файлов и записей в реестре и т. п., их можно добавить на этом шаге (рис. 8). Например:

- запуск приложения деинсталляции: *Program.exe /uninstall* или *msiexec /u mymsi.msi /quiet;*
- удалить файлы, которые больше не нужны: *del file.exe* или *rmdir userdirectory;*
- запуск системной команды: *shutdown /r.*

Если на шаге 2 в модуль были добавлены файлы, то можно ссылаться на них в определяемых командах. В случае если вы не хотите разрешать деинсталляцию моду-

ля, необходимо выбрать пункт **This module cannot be uninstalled**.

ШАГ 6. СОХРАНЕНИЕ МОДУЛЯ

Это завершающий шаг работы Module Designer, предоставляющий возможность сохранить созданный модуль и импортировать его в каталог, откуда он может быть выбран в ICE и IBW. Предлагаются три варианта, при этом возможен выбор любых их комбинаций (рис. 9):

- **Save module configuration file** — сохранение файла конфигурации модуля. На самом деле файл модуля не будет создан, а будет сохранен собственно весь процесс работы в Module Designer (файл .emx), чтобы его можно было продолжить впоследствии. Это удобно для быстрого сохранения модуля и последующего использования/модификации его в Module Designer.
- **Create and save module** — создание и сохранение модуля. В этом случае происходит создание модуля (файл .emd), который может быть установлен на целевом устройстве. Это полезно, когда требуется доустановить модуль на существующую операционную систему Standard 8 без ее полной переустановки. Также следует выбирать эту опцию, когда планируется подписать созданный модуль.
- **Create module and import it into the catalog** — создание модуля и импортирование его в каталог. Происходит создание модуля (файл .emd) и импорт его в каталог. Это делает модуль доступным в ICE. Он также будет доступен в IBW после создания нового диска IBW на основе обновленного каталога.

Осуществив выбор требуемых действий, следует нажать последовательно **Create** и **Finish**, что приведет к сохранению модуля и выходу из Module Designer.

Следует отметить, что только 1 и 6 шаги являются обязательными. Если нет необходимости добавления файлов и зависимостей и определения команд инсталляции/деинсталляции, то любые из шагов со второго по пятый можно пропустить.

В результате выполнения вышеописанных шагов был создан пользовательский модуль. Если в дальнейшем планируется использование его подписанной версии, то следует выполнить несколько дополнительных шагов, а именно создать сертификат и подписать им разработанный модуль:

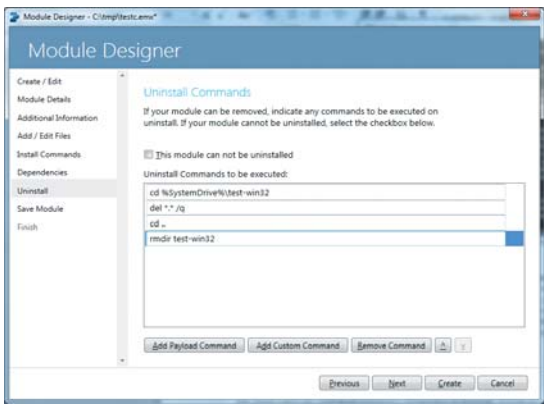


РИС. 8. ▲ Окно добавления команд деинсталляции

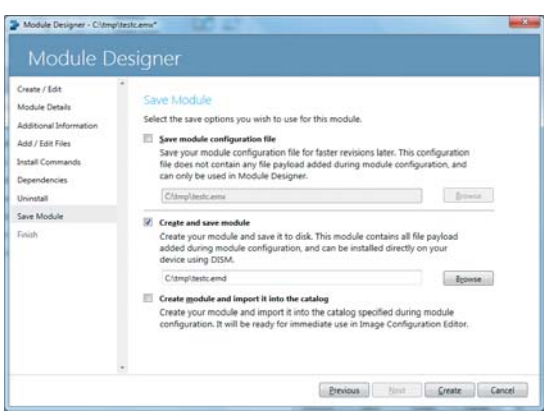
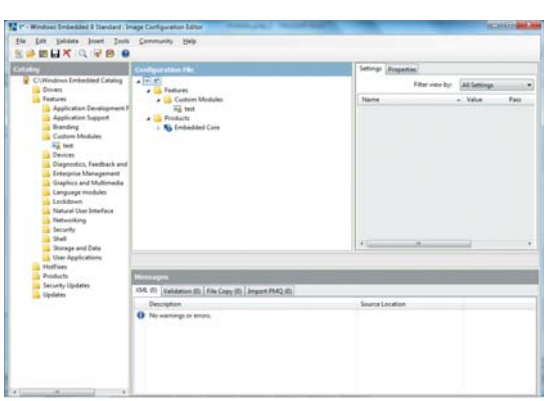


РИС. 9. ▲ Окно сохранения и импорта созданного модуля

РИС. 10. ▼ Подписанный модуль доступен для разработки образа ОС

- Создать новый сертификат с помощью утилиты создания сертификатов (Certificate Creation Tool) Makecert.exe.
- Добавить новый сертификат на компьютер разработчика с помощью утилиты управления сертификатами (Certificate Manager Tool) Certmgr.exe.
- Подписать пользовательский модуль с помощью утилиты подписания (Sign Tool) Signtool.exe.



Для этого используются утилиты командной строки, которые устанавливаются на компьютер разработчика автоматически вместе с Visual Studio и Windows SDK. Возможно определить и подписать любые неподписанные emd-файлы пользовательских модулей, которые уже были импортированы в каталог Standard 8. Однако после их подписания необходимо переиндексировать каталог, чтобы они отображались правильной иконкой в Image Builder Wizard (IBW) и Image Configuration Editor (ICE). После подписания пользовательского модуля, чтобы он стал виден в ICE и IBW, необходимо его импортировать в каталог Standard 8. Для этого удобно использовать менеджер каталогов (Catman.exe) — утилиту командной строки, которая идеально подходит для обслуживания каталога Standard 8. Catman.exe может быть запущен на Windows 7 с пакетом обновления 1 (Windows 7 SP1), Windows 8 или Windows Embedded 8 Standard (Standard 8) и позволяет импортировать wim-файлы, модули или CBS-пакеты в каталог, удалять модули из каталога и перестраивать индексный файл каталога. После импортирования подписанного пользовательского модуля в каталог он становится доступным для разработки образа ОС в Image Configuration Editor и Image Builder Wizard (рис. 10).

В дальнейшем мы продолжим рассмотрение возможностей Windows Embedded 8 Standard. Следующая статья будет посвящена приложениям Windows RT и вопросам их разработки и развертывания на устройстве. ⬢

ЛИТЕРАТУРА:

1. http://getwindowsembedded8.com/
2. http://msembedded.ru/os/windows-embedded-8-standard
3. www.microsoft.com/en-us/download/details.aspx?id=37019
4. http://msdn.microsoft.com/en-us/library/jj963237(v=winembedded.81).aspx
5. http://msdn.microsoft.com/en-us/library/jj963113(v=winembedded.81).aspx
6. www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx