

**LABCAR**

**-**

**AUTOMATION**

**4**

**.2.2**

ATCL (Automotive Test Class Library)

Getting Sta

r

ted for Test Case Developers

**Copyright**

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2012 – 2015 ETAS GmbH, Stuttgart

The names and designations used in this document are trademarks or brands belonging to the respective owners.

2

Contents

**Contents**

[1 Introduction 4](#_Toc33271)

[1.1 Preliminaries 4](#_Toc33272)

[1.2 Installation Paths 5](#_Toc33273)

[2 ATCL Overview 6](#_Toc33274)

[2.1 What is the Automotive Testing Class Library (ATCL)? 6](#_Toc33275)

[2.2 Ports and Signatures 6](#_Toc33276)

[2.3 Managers 7](#_Toc33277)

[3 Writing a simple test case 8](#_Toc33278)

[3.1 Setting up a new Visual Studio solution 8](#_Toc33279)

[3.1.1 Adding a Constructor 9](#_Toc33280)

[3.1.2 Connecting to LABCAR-AUTOMATION 9](#_Toc33281)

[3.1.3 New Files 10](#_Toc33282)

[4 Structure your Test Case 11](#_Toc33283)

[4.1 Test Case Initialization 11](#_Toc33284)

[4.1.1 Register Metadata 11](#_Toc33285)

[4.1.2 Register Ports 12](#_Toc33286)

[4.1.3 Register Parameters 12](#_Toc33287)

[4.1.4 Load Parameters 13](#_Toc33288)

[4.2 Test Case Structure 13](#_Toc33289)

[4.2.1 Initialization 14](#_Toc33290)

[4.2.2 Stimulation 14](#_Toc33291)

[4.2.3 Measurement 14](#_Toc33292)

[4.2.4 Evaluation 14](#_Toc33293)

[4.3 Verdict Handling 14](#_Toc33294)

[4.4 Configuring your test bench 14](#_Toc33295)

[4.4.1 Test Case Definition File 17](#_Toc33296)

[4.4.2 Using the TCD File Generator 19](#_Toc33297)

[4.4.3 Using the Command Line Interpreter 21](#_Toc33298)

[4.5 Setting up an LABCAR-AUTOMATION project 21](#_Toc33299)

[5 Report Design 22](#_Toc33300)

[5.1 Sections and Verdicts 22](#_Toc33301)

[5.2 Working with Tables 22](#_Toc33302)

[5.3 Working with Plots 23](#_Toc33303)

[6 Using own software components 25](#_Toc33304)

[6.1 ATCL Managers 25](#_Toc33305)

[6.2 Ports and Tool Adapters 25](#_Toc33306)

[6.3 Using other DLLs 26](#_Toc33307)

[7 ETAS Contact Addresses 27](#_Toc33308)

Introduction

# Introduction

В этом разделе «Начало работы» разработчику тестового сценария показано, как создать тестовый сценарий для LABCARAUTOMATION версии 4.0 и выше с помощью Test Design Connector .NET (ATCL). В качестве языка разработки тестов выбран C#, но примеры можно легко перенести на любой другой язык программирования, совместимый с Microsoft .NET, например VB.NET или Python.NET.

В учебнике подробно показано, как

 настроить новый тестовый пример,

 вызов функционала тестового стенда с использованием портов и подписей

 сгенерировать необходимое описание интерфейса для публикации тестового примера в библиотеке выпусков тестов

 использовать параметры для внешней параметризации в Менеджере тестирования

 ведение журнала

 управлять конфигурацией и инициализацией испытательного стенда.

Please have fun with your first steps in LABCAR-AUTOMATION with .NET.

Your LABCAR Product Team

1.1 Предварительные сведения

В руководстве предполагается, что у вас есть базовые знания о принципах и терминах, используемых в LABCAR-AUTOMATION V2, V3 или V4. В этом помогут презентация с информацией о продукте и руководство пользователя. Вы также должны иметь элементарные навыки программирования на C# или другом объектно-ориентированном языке.

Подготовка:

 Установка LABCAR-AUTOMATION V4.x с LABCAR-TBCNET или LABCARAUTOMATION Core V4.0 или выше. Ядро LABCAR-AUTOMATION является частью установки каждого пакета LABCAR-AUTOMATION, будь то Standard, Professional или Embeddable.

 Установка Microsoft Visual Studio 2005 или более поздней версии (или любой среды разработки, позволяющей создавать код C#). В любом случае в этом документе используются термины, используемые в Microsoft Visual Studio.

 Установка LABCAR-OPERATOR V4.0.x или выше и (опционально) INCA V5.4.1. или выше

 Совместимый LABCAR-RTPC V3.0.x для выполнения примеров.

Introduction

## Installation Paths

Все пути установки в этом документе относятся к системной среде Windows 7 (64-разрядная). Стандартные каталоги в этой среде:

**Program Files:**

C:\Program Files (x86)\ETAS\LABCAR-AUTOMATION 4.x

**Configuration Files:**

C:\ProgramData\ETAS\LABCAR-AUTOMATION 4.x

**Examples and Default Data:**

C:\Users\Public\Documents\ETAS\LABCAR-AUTOMATION 4.x

When working in a **Windows XP** environment, these directories are located at:

**Program Files:**

C:\Program Files\ETAS\LABCAR-AUTOMATION 4.x

**Configuration Files:**

C:\Documents and Settings\All Users\Application Data\

ETAS\LABCAR-AUTOMATION 4.x

**Examples and Default Data:**

C:\Documents and Settings\All Users\Documents\

ETAS\LABCAR-AUTOMATION 4.x

ATCL Overview

# ATCL Overview

2.1 Что такое библиотека классов автомобильного тестирования (ATCL)?

ATCL (библиотека классов автомобильного тестирования) — это инфраструктура, которая позволяет писать независимые тестовые примеры тестового стенда для использования в LABCAR-AUTOMATION. Он поставляется с надстройкой продукта LABCAR-TDCNET (Test Design Connector .NET) и находится в папке установки LABCARAUTOMATION («/TestTools/bin/Etas.Eas.Atcl.Interfaces.dll»).

ATCL предоставляет большой набор функций для доступа к испытательному стенду без использования специфичных для инструмента и собственных методов. Чтобы обеспечить независимость тестового примера от любого используемого инструмента, ATCL предоставляет список так называемых портов, которые группируют определенные функции (называемые «сигнатурами») разумным и родным образом.

Test Case

Port

Tool Adapter

Tool API

Abstract

Concrete

## Во время выполнения соединители тестового стенда LABCAR-AUTOMATION заботятся об интерпретации этих портов и подписей и сопоставляют их с вызовами инструментов и API (при этом тестировщику не приходится заниматься этим лично).

## Ports and Signatures

В настоящее время определены следующие порты (с их сигнатурами):

Table 1:

|  |  |  |
| --- | --- | --- |
| Port | Description | Name of port interface in the ATCL |
| Model Access | Используется для доступа к инструментам симуляции и моделирования Test Bench и любым механизмам моделирования (таким как LABCAR-OPERATOR или dSPACE ControlDesk).  Доступны функции для установки и считывания значений модели (измеряемых элементов и параметров), для регистрации данных и генерации сигналов и так далее. | IPortMA |
| ECU Access Measurement | Используется для доступа к внутренним данным ECU (элементы измерения, данные калибровки) и для прошивки ECU. | IPortEAM |
| ECU Access  Calibration | IPortEAC |
| ECU Access Flash | IPortEAF |
| Sync DL | Работает как микшер для синхронизации регистрации данных  между разными портами (например, для получения синхронных данных для INCA и LABCAR-OPERATOR | IPortSyncDL |
| File Access Composite | Порты, предоставляющие и абстрактные методы для доступа к табличным (например, \*.ini-или \*.csv) данным или составным (например, иерархическим данным, таким как \*.xml-файлы) для чтения данных в тестовые примеры без непосредственного открытия файлов из тестового примера. | IPortFACD |

ATCL Overview

|  |  |  |
| --- | --- | --- |
| Data |  |  |
|  |  |
| File Access Tabular Data |  | IPortFATD |
| Fault  Simulation | Предоставляет функциональные возможности для управления сценариями имитации отказов. | IPortFS |
| J1699 | Предоставляет функциональные возможности для выполнения тестов на соответствие OBD в соответствии со спецификацией J1699. | IPortJ1699 |
| Diagnostics | Предоставляет функциональные возможности для доступа к памяти диагностики (кода неисправности) ЭБУ. | IPortDiag |

For a detailed list of functions (“signatures”) available for each port, please refer to the “ATCL reference Manual” shipped electronically with LABCAR-AUTOMATION.

## Managers

Наряду с описанными портами для доступа к функциям испытательного стенда ATCL предоставляет ряд так называемых «менеджеров», работающих с внутренними данными тестового сценария, такими как вердикты, параметры, отчеты и метаданные.

В тестовом случае доступ к этим менеджерам осуществляется с помощью класса ATCL «Factory». Чтобы использовать функции менеджеров, вы должны получить к ним доступ, используя соответствующую функцию «Get…Manager» Factory, например

… Factory.GetMetaDataManager() …

… Factory.GetParameterManager() …

… Factory.GetVerdictMAnager() …

The use of this factory can be seen in more detail in the subsequent chapters.

LABCAR-AUTOMATION 4.2.2

**Написание простого тестового примера**

## Тестовый пример ATCL, который должен выполняться LABCAR-AUTOMATION, обычно представляет собой исполняемый файл .NET. Если тестовый пример используется в тестовом проекте (\*.lcaprj) и запускается обработчиком теста, он подключается к LABCAR-AUTOMATION и необходимым инструментам в зависимости от используемых сигнатур и списка доступных инструментов, описанных в конфигурация испытательного стенда.

Настройка нового решения Visual Studio

Вы можете написать свой тестовый пример ATCL на любом языке, который поддерживает коннектор .NET, и в любой среде разработки, которая вам нравится. В этом руководстве мы предполагаем, что вы используете Visual Studio 2005 и C#.

Для нового тестового случая вы создаете новое «Консольное приложение» с помощью мастера проектов Visual Studio. Мастер автоматически создает для вас класс и метод main..

namespace TC\_Demo

{

class Program

{

static void Main(string[]args)

{

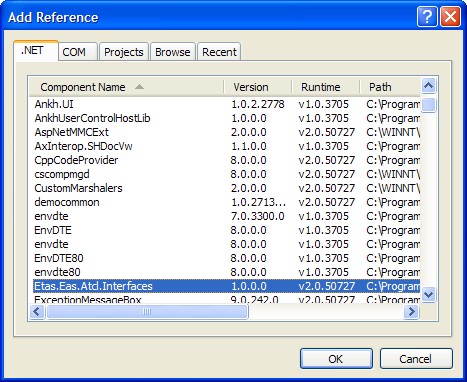
}

}

}

Чтобы получить доступ к инфраструктуре ATCL, вы должны добавить ссылку на

Etas.Eas.Atcl.Interfaces.dll в свой проект. В Visual Studio 2005 эта DLL автоматически интегрируется при установке LABCAR-AUTOMATION, и ее можно найти в диалоговом окне «Добавить ссылку».



Кроме того, DLL можно найти в каталоге «PublicAssemblies» в каталоге установки LABCARAUTOMATION.

Если вы используете какую-либо другую DLL, вы также должны указать ее в проекте.

### Имейте в виду, что обращение к .NET DLL или COM-компонентам непосредственно из тестового примера приводит к зависимому от тестового стенда поведению в тестовом примере.

### 

### После того, как вы добавили ссылку на интерфейсы ATCL, вы должны получить свой класс из базового класса «Etas.Eas.Atcl.TestCase». Это класс, определенный в структуре ATCL, из которого должны быть получены все тестовые примеры. Этот вывод преобразует стандартный исполняемый файл в абстрактный тестовый пример, готовый для использования в цепочке инструментов LABCAR-AUTOMATION. Если мы попытаемся скомпилировать наш тестовый пример сейчас, мы получим сообщение об ошибке, поскольку мы еще не реализовали конструктор класса тестового примера.

### Добавление конструктора

Базовый класс TestCase не имеет конструктора по умолчанию (как известно из других классов .NET), а имеет специальный конструктор, принимающий имя тестового примера в качестве первого аргумента. Поэтому нам нужно добавить конструктор, который предоставляет имя тестового примера базовому классу. Кроме того, мы переименовываем наш класс тестового примера и называем его «MyFirstTestCase». Теперь код должен выглядеть так:

namespace TC\_Demo

{

public class MyFirstTestCase : Etas.Eas.Atcl.Interfaces.TestCase

{

public MyFirstTestCase()

: base("MyFirstTestCase")

{

}

static void Main(string[] args)

{

}

}

}

### Connecting to LABCAR-AUTOMATION

Теперь мы можем скомпилировать и выполнить наш тестовый пример, но подключение к ЛАБКАР-АВТОМАТИЗАЦИЯ по-прежнему отсутствует. Это соединение устанавливается, когда мы создаем экземпляр нашего тестового класса.

static void Main(string[] args)

{

MyFirstTestCase tc = new MyFirstTestCase();

}

Когда мы создадим экземпляр нашего класса тестового примера, а затем запустим проект Visual Studio, мы увидим консоль для исполняемого файла тестового примера и через несколько секунд получим сообщение об ошибке: «[e] - Не удалось найти сервер. Повторите попытку через несколько секунд!» Это сообщение об ошибке говорит о том, что тестовый пример пытался установить соединение с LABCAR-AUTOMATION Engine, но соединение не может быть установлено, поскольку ядро не загружено. Это сообщение верно! Механизм обычно открывается приложением обработчика тестов, которое еще не запущено.

### New Files

Помимо отображения сообщения об ошибке, когда мы запускали тестовый пример, произошло еще кое-что. Если мы заглянем в каталог, из которого был выполнен тестовый пример (обычно это каталог bin\Debug в нашем проекте Visual Studio), то обнаружим еще несколько файлов.

Эти новые файлы:

** файл тестовых параметров (TPA)**

** файл описания тестовой иерархии (THD) и**

** файл описания тестовой архитектуры (TAD).**

Эти файлы требуются цепочке инструментов LABCAR-AUTOMATION и генерируются тестовым набором каждый раз при его выполнении.

Они содержат информацию о том, какие порты используются тестовым набором, какие параметры требуются этому тестовому набору для выполнения, а также некоторые метаданные, такие как версия тестового примера, разработчик тестового примера и т. д. Вся информация генерируется во время выполнения. Платформа ATCL записывает соответствующие вызовы платформы и автоматически создает соответствующие файлы.

# Structure your Test Case

Тестовый пример должен иметь раздел инициализации тестового примера, в котором выполняются все предварительные действия для основной последовательности тестов (например, генерация файлов тестовых случаев, объявление параметров и т. д.). Во время этого раздела инициализации тестового примера вы должны избегать создания исключений и прерывания процедуры.

Main

Test

Case

Initialization

Register

Metadata

Register Ports

Register Parameters

Load

Parameters

Initialization

Stimulation

Measurement

Evaluation

Finished

Variable

Declaration

## Базовая структура, описанная ниже, не является обязательной, но она достаточно хорошо описывает процесс.

## Test Case Initialization

Сам раздел «Инициализация тестового примера» более или менее разделен на четыре этапа.

### Register Metadata

Сначала мы регистрируем метаданные для этого теста. Метаданные — это любая проза, которая может быть полезной для конкретного теста. Можно использовать любые пары строк «ключ»/«значение» (первая описывает тип метаданных, а вторая содержит сами данные).

Тем не менее, минимально необходимые метаданные — это комментарий, цель и версия. Эти элементы метаданных генерируются автоматически, если разработчик тестового примера не указывает их явно. Таким образом, мы должны предоставить им достоверную информацию. Поскольку мы можем добавить все дополнительные метаданные, которые нам нужны, приведенный ниже пример также содержит название компании в качестве метаданных:

private void RegisterMetaData()

{

Factory.GetMetaDataManager().AddMetaData(

new TCMetaData("Comment", "My first demo test case"));

Factory.GetMetaDataManager().AddMetaData(

new TCMetaData("Purpose", "Demonstrating how to write a simple test case"));

Factory.GetMetaDataManager().AddMetaData(

new TCMetaData("Version",

GetType().Assembly.GetName().Version.ToString()));

Factory.GetMetaDataManager().AddMetaData( new TCMetaData("Company", "ETAS GmbH"));

}

Метаданные записываются в файл THD, и мы увидим эти метаданные в диспетчере тестов, когда добавим тестовый пример в проект LABCAR-AUTOMATION.

### Register Ports

Вторым шагом будет регистрация портов, которые мы используем в нашем тестовом примере. Этот шаг важен, так как он сообщает LABCAR-AUTOMATION, какие порты используются в тестовом примере. Во время инициализации тестового стенда (открытие инструментов тестового стенда и их настройка) «ЛАБКАР-АВТОМАТИЗАЦИЯ» просматривает эту регистрацию, чтобы узнать, какие порты каким инструментам будут сопоставлены.

Вы регистрируете порт «Доступ к модели» и порт «Измерение доступа к ECU», используя следующий код.

IPortMA m\_maPort = null;

IPortEAM m\_eaPort = null;

...

private void RegisterPorts()

{

m\_maPort = Factory.GetPortMA("P\_MA"); m\_eaPort = Factory.GetPortEAM("P\_EAM");

}

Строка, передаваемая соответствующему методу «Get\*», является желаемым именем экземпляра порта. Он снова появляется в файлах конфигурации испытательного стенда (\*.tbc), где его необходимо сопоставить с конкретными инструментами. (см. раздел «Конфигурация испытательного стенда» ниже).

### Register Parameters

Следующим шагом является создание интерфейса параметризации для обеспечения возможности адаптации теста путем параметризации в приложении LABCAR-AUTOMATION Test Manager.

Каждый тестовый пример может иметь произвольную структуру параметров (которые указаны в этом разделе здесь). В библиотеке выпусков тестов структура и спецификация параметров хранятся в файле «\*.tpa». По умолчанию тестовый пример не имеет параметров, поэтому автоматически сгенерированный файл TPA всегда пуст.

Параметризация состоит из двух шагов. Сначала мы должны объявить некоторые переменные в качестве параметров тестового примера, которые записываются в файл TPA и могут быть изменены в диспетчере параметров теста. Второй шаг — прочитать параметризованное значение для этой определенной переменной обратно в тестовый пример (это описано в следующей главе).

Публикация переменных в качестве параметров тестового примера чем-то похожа на регистрацию портов. Сначала необходимо объявить локальную переменную. Затем эти переменные необходимо зарегистрировать в экземпляре IParameterManager, вызвав метод «Register()». После этого параметры записываются в файл TPA и могут отображаться и изменяться с помощью диспетчера параметров тестирования.

TypeSutFloat operandA = new TypeSutFloat(

"Operand A", "operA", "First operand", 1.0, 0.0, 0.0, "");

TypeSutFloat operandB = new TypeSutFloat(

"Operand B", "operB", "Second operand", 1.0, 0.0, 0.0, "");

...

private void RegisterParameters()

{

Factory.GetParameterManager().CreateTpaFile();

Factory.GetParameterManager().Register(operandA);

Factory.GetParameterManager().Register(operandB);

Factory.GetParameterManager().Save();

}

В методе «RegisterParameters()» переменные регистрируются в ATCL и, таким образом, публикуются как параметры тестового примера. Они будут записаны в файл TPA и могут быть настроены доверенным платформенным модулем.

Имейте в виду, что это только первый шаг. Значения переменных по-прежнему являются значениями по умолчанию, определенными в конструкторе «TypeSUTFloat». Даже если доверенный платформенный модуль установил бы для параметров некоторые другие значения, это не повлияет на тестовый пример, пока вы явно не прочитаете параметризацию для этого тестового примера.

### Load Parameters

Прежде чем использовать переменную, которая была опубликована в качестве параметра тестового примера, вы должны прочитать параметризованное значение для этих переменных. Это делается с помощью метода «Parameterise()».

private void LoadParameters()

{ operandA =

Factory.GetParameterManager().Parameterise(operandA); operandB =

Factory.GetParameterManager().Parameterise(operandB);

}

После вызова методом «ЗагрузитьПараметр()» значения переменных «операндА» и

«operandB» были изменены со значений по умолчанию на значения, предоставленные диспетчером параметров тестирования. Следовательно, если вы не вызываете «Parameterise()» для переменной, вы всегда будете работать со значением переменной по умолчанию.

**IMPORTANT:**

Если вы изменяете или добавляете параметры в тестовый пример, вам нужно повторно запустить тестовый пример один раз, чтобы получить новые параметры в файле TPA. В противном случае файл TPA будет содержать старые параметры..

## Test Case Structure

После этого раздела инициализации тестового примера мы разделяем наш тестовый пример на четыре разных раздела: раздел инициализации, стимуляции, измерения и оценки.

ATCL (Automotive Test Class Library)

### Initialization

В разделе инициализации мы готовим наш тестовый стенд. Это включает в себя настройку всех портов, а также настройку регистратора данных или, например. задать определенные условия окружающей среды. Обычно измерение останавливается во время этой инициализации..

### Stimulation

В разделе стимуляции мы стимулируем ЭБУ и/или модель, чтобы получить желаемое поведение. Стимуляцию также можно проводить во время измерения.

### Measurement

В разделе измерений мы записываем значения от ЭБУ и/или модели.

### Evaluation

В разделе оценки мы сравниваем, например. записанные значения от модели с записанными значениями от ECU. Мы должны решить, в порядке ли записанные значения ECU.

## Verdict Handling

Вердикт имеет встроенный конечный автомат. Вердикты могут меняться с «нет» на «пройдено», с «пройдено» на «не пройдено», с «не пройдено» на «несоответствие» (неубедительно) и с «несогласно» на «ошибка». Вынесение вердикта осуществляется вызовом соответствующей функции вердикта (например, «Pass()» или «Fail()»). Если приговор, т.е. является «неудачным», вы никогда не сможете вернуть этот вердикт на «пройдено». Поэтому мы предлагаем создать новый вердикт для каждого внутреннего раздела тестового примера. Это позволяет нам иметь некоторые разделы с ошибкой, в то время как другие разделы могут пройти. Вердикт и разделы тесно связаны с дизайном отчета, поэтому более подробно об этом можно прочитать в главе 5.

## Configuring your test bench

Чтобы наш тестовый стенд заработал, его необходимо настроить. Настройка тестового стенда состоит из трех шагов:

1. Сопоставление портов с инструментами

В файлах конфигурации тестового стенда (\*.tbc, редактируется с помощью TBC

Editor) все необходимые экземпляры портов должны быть сопоставлены с инструментом (путем указания соответствующего адаптера инструмента). Для каждого инструмента предоставляется конфигурация инструмента, в которой перечислены доступные данные проекта для конкретного инструмента..

2. Сопоставление конфигураций инструментов

В файлах конфигурации инструментов (\*.tcf, отредактированных с помощью редактора TCF) каждый конкретный проект инструмента (например, проекты LABCAR-OPERATOR, базы данных INCA) абстрагируется с использованием наборов идентификаторов. Обычно существует более одного идентификатора, идентифицирующего конфигурацию инструмента, чтобы обеспечить зависимость конфигурации как от тестового примера, так и от проверяемого оборудования. Для каждого элемента конфигурации инструмента обычно имеется отображение сигналов (SUT Mapping), чтобы обеспечить максимально абстрактный доступ к меткам инструментов..

3. SUT Mapping

Каждая используемая метка элемента сигнала или меры сопоставляется с меткой конкретного инструмента в файлах SMF (редактируется с помощью редактора файлов сопоставления SUT). Записи сопоставления содержат спецификации единиц измерения и диапазонов значений.

Для полной настройки испытательного стенда каждый порт содержит одинаковые сигнатуры конфигурации.

* Port\_xy. Create()

Открывает инструмент, реализующий функции портов, указанные в файле TBC.

* Port xy.ConfigureTool()

Передает первую часть набора идентификаторов конфигурации инструмента в LABCAR-

AUTOMATION. Эти идентификаторы определяются для каждого UUT в среде UUT. Таким образом, эта сигнатура отображает зависимость UUT от конфигурации испытательного стенда..

* Port xy.Configure()

Передает вторую часть набора идентификаторов tcf в LABCAR-

AUTOMATION и, таким образом, завершает набор идентификаторов. Эта сигнатура обычно используется в самом тестовом примере, чтобы влиять на конфигурацию тестового стенда из тестового примера.

С этой последней сигнатурой конфигурация инструмента может быть разрешена, и соответствующий инструмент настроен. Для ЛАБКАР-ОПЕРАТОР это означает открытие \*.lca-файла, указанного в tcf-файле, загрузку кода на ЛАБКАР-RTPC и запуск целевой ОС.

В большинстве тестовых проектов вы будете использовать так называемую инициализацию тестового стенда («TBInit»), что означает, что сигнатуры «Create()» и «ConfigureTool()» всех необходимых портов вызываются обработчиком теста. инструмент перед запуском самого первого тестового примера. Следовательно, вам не нужно явно кодировать их в своем тестовом примере.

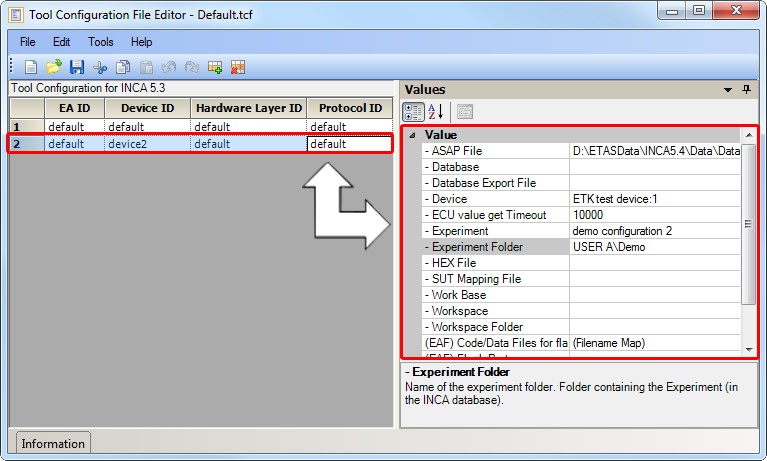
Если вы хотите выполнить тестовый пример непосредственно из Visual Studio (например, в целях отладки), вы должны сами вызвать сигнатуры «Create()» и «ConfigureTool()» в коде тестового примера. Примеры тестовых случаев показывают несколько возможностей, как справиться с этим во время инициализации.

Сигнатуры «ConfigureTool()» и «Configure()» зависят от порта и имеют аргументы, специфичные для порта. Это означает, что для каждого порта отдельно указано, сколько идентификаторов требуется для идентификации конфигурации и каким образом они разделены между «Configure()» и «ConfigureTool()». Конфигурация порта «Доступ к модели» сопоставляется с использованием трех идентификаторов, которые передаются сигнатурой «ConfigureTool()», в то время как конфигурация инструмента для порта «Доступ к ECU» сопоставляется с использованием четырех идентификаторов, из которых только один передается LABCAR-AUTOMATION с сигнатурой «ConfigureTool()», а остальные три передаются из тестового примера с помощью сигнатуры «Configure()».

Пример для порта EAM поможет проиллюстрировать это: порт EAM имеет четыре конфигурационных ключа «EA\_ID», «DeviceID», «HWLayerID» и «ProtocolID». Комбинация четырех клавиш выбирает ровно одну запись из файла TCF.

На снимке экрана ниже показаны две разные конфигурации порта EAM. Первая конфигурация в примере определяется ключами «по умолчанию», «по умолчанию», «по умолчанию» и «по умолчанию», а вторая конфигурация определяется ключами «по умолчанию», «устройство2», «по умолчанию» и «по умолчанию».

ATCL (Automotive Test Class Library)



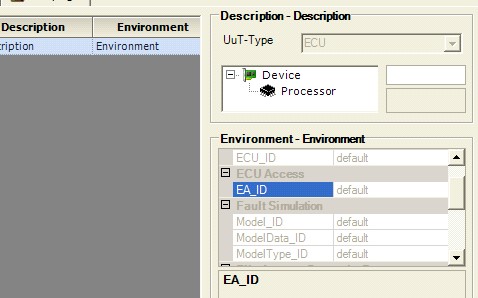
В порту EAM сигнатура «ConfigureTool()» принимает только один аргумент —

Ключ конфигурации EA\_ID. Сигнатура «Configure()» принимает три аргумента:

«DeviceID», «Идентификатор аппаратного уровня» и «Идентификатор протокола». Таким образом, как только тестовый пример вызывает подпись «Configure()», все необходимые ключи конфигурации фиксируются, и конфигурация для экземпляра порта может быть загружена адаптером инструмента.

Теперь разработчик тестового примера может принять решение о реализации «ConfigureTool («по умолчанию»)» и «Configure («устройство2», «по умолчанию», «по умолчанию»)» или просто вызвать «Configure()» и позволить обработчику тестов выполнить «Настроить инструмент()». В случае кампании «TBInit» TestHandler вызывает

«Настроить инструмент()». Значение для вызова «ConfigureTool()» определяется в среде проверяемого оборудования проверяемого оборудования.



На снимке экрана показана часть среды UuT. Для порта «Доступ к ECU» есть только ключ EA\_ID, который установлен по умолчанию. Значения из среды UUT считываются обработчиком тестов и передаются в сигнатуру «ConfigureTool()» соответствующих портов. Например, в случае порта «Доступ к модели» среда UUT содержит все три ключа, необходимые для выбора конфигурации инструмента. Таким образом, сигнатура «Configure()» порта «Model Access» не имеет аргументов, так как все необходимые ключи уже переданы при вызове «ConfigureTool()».

В общем случае мы говорим: Ключи из сигнатуры «ConfigureTool()» вместе с ключами из сигнатуры «Configure()» выбирают конфигурацию. Количество ключей «ConfigureTool()» и «Configure()» зависит от порта. Существуют порты с ключами на обеих подписях, такие как порт «EAM» или «EAC», есть ключи с исключительно ключами «ConfigureTool()», такие как порт «Model Access», и есть порты с исключительно ключами «Configure()». Порт «SyncDL».

Configure

Tool

Configure

EA\_ID

Device

HWLayer

Protocol

e.g. LCO Configuration

Model

Model

Data

Model

Type

ConfigureTool

Configure



NONE

e.g. INCA Configuration

ConfigureTool

Con

figure

Key

1

Key

2

Key

N

e.g. SyncDL Configuration



NONE

Кроме того, не для всех портов требуется файл конфигурации инструмента. Какой файл конфигурации инструмента (TCF) применяется к порту, определяется в файле конфигурации испытательного стенда (TBC). Файл TBC определяет список доступных портов и сопоставляет эти порты с инструментами и адаптерами инструментов. Один инструмент может иметь ноль или один файл TCF. Таким образом, если вы используете много экземпляров порта EAM в своем тестовом примере, все они работают с одним и тем же файлом TCF.

**Best Practice**

Если у вас нет особых требований к клавишам «Configure()», вы должны использовать «по умолчанию» в качестве стандартного значения. Это упрощает создание правильных записей TCF для вашего тестового примера.

Конечно, вы должны убедиться, что все порты, которые вы используете в своем тестовом примере, определены в файле TBC!

Дополнительные сведения о настройке подписей определенных портов см. в «Справочном руководстве ATCL».

### Test Case Definition File

Чтобы добавить наш новый тестовый пример (функциональность) в проект LABCAR-AUTOMATION, необходимы дополнительные файлы TPA (файл параметров теста), THD (файл описания иерархии теста), TAD (файл описания архитектуры теста) и TCD рядом с файлом. сама функциональность тестового примера. Файлы TPA, THD и TAD генерируются тестовым набором автоматически. Просто убедитесь, что вы выполнили свой тестовый пример хотя бы один раз, прежде чем пытаться добавить его в проект LABCAR-AUTOMATION.

Файл TCD (определение тестового примера) по-прежнему отсутствует. Его можно рассматривать как «испытательный случай». Он определяет, какие файлы принадлежат тестовому набору. Как было сказано ранее, должны быть как минимум файлы TPA, THD и TAD вместе с самим тестовым примером, который обычно является исполняемым файлом. Файл TCD может определять дополнительные зависимости для тестового примера, такие как сценарии Python или библиотеки .NET.

XML-схему (lca\_tcd\_v3.0.xsd) для файла TCD можно найти в LABCAR-

Каталог установки АВТОМАТИЗАЦИИ в папке ./TestTools/XMLSchema. Минимальный файл TCD для нашего простого тестового примера будет:

ATCL (Automotive Test Class Library)

<?xml version="1.0"?>

<file …="" TDL="StdExe" …="" >

<TestCase Name="MyFirstTestCase">

<EntryPoint>MyFirstTestCase.exe</EntryPoint>

</TestCase>

<MetaData>

<TPA>MyFirstTestCase.tpa</TPA>

<TAD>MyFirstTestCase.tad</TAD>

<THD>MyFirstTestCase.thd</THD>

</MetaData>

<Checksums />

</file>

В примере XML вы можете увидеть минимальный файл TDC для нашего тестового примера. Вы можете заметить, что элемент «Контрольные суммы» пуст. Контрольные суммы для каждого файла, принадлежащего этому тестовому набору, вычисляются при добавлении тестового набора в проект LABCAR-AUTOMATION. В любом случае элемент «Контрольные суммы» должен присутствовать. Необходимые файлы перечислены в элементе «Метаданные». Дополнительные файлы, необходимые для тестового примера, включаются с помощью элемента «Зависимости», параллельного элементу «EntryPoint»..

…

<TestCase Name="MyFirstTestCase">

<EntryPoint>MyFirstTestCase.exe</EntryPoint>

<Dependencies>

<Dependency>AdditionalFunctions.dll</Dependency>

</Dependencies>

</TestCase>

…

Файл TCD также определяет язык разработки тестов (TDL). Этот атрибут используется обработчиком тестов, чтобы различать, какой исполнитель тестового примера должен быть выбран для выполнения этого конкретного тестового примера. В этом случае выбирается исполнитель для стандартных исполняемых файлов. Атрибут «TDL» должен соответствовать одному исполнителю, определенному в файле TestCaseExecutorFactory.config, расположенном в каталоге TestTools/bin.

Установка ЛАБКАР-АВТОМАТИЗАЦИЯ 4.0 поддерживает:

 «Исполнитель TTCN», который может выполнять тестовые примеры TTCN-3 и выбирается значением атрибута «TDL» «TTCN». Разница между «TTCN Executor» и «StandardExe Executor» заключается в том, что первый предоставляет некоторые специфичные для TTCN аргументы командной строки, такие как уровень детализации TTCN и т. д.

 «Исполнитель StandardExe», который может выполнять файлы «\*.exe» и выбирается по значению атрибута TDL «StdExe».

 «Python Executor», который может выполнять сценарии Python и выбирается по значению атрибута TDL «PythonNET».

 «Исполнитель TestStand», который может выполнять тестовые последовательности TestStand и выбирается по значению атрибута TDL «TestStand».

**IMPORTANT**

Значение атрибута TDL чувствительно к регистру! Если у вас есть другие языки разработки тестов с соответствующими исполнителями, вы можете выбрать их, используя их значение TDL в файле TCD..

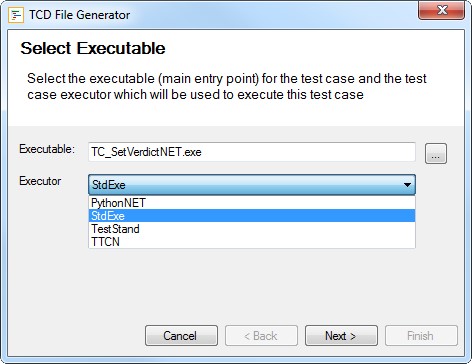
**IMPORTANT**

Все зависимости, включая файлы TPA, THD и TAD, должны находиться в каталоге тестового примера. Запрещено ссылаться на зависимости, расположенные за пределами каталога тестового примера, а также на подкаталоги под каталогами тестового набора.

### Using the TCD File Generator

Генератор файлов TCD (TCDCMD.Gui.exe) входит в состав пакета LABCAR-TDCNET и находится в меню «Пуск» в разделе «Программы»  ETAS  LABCAR-AUTOMATION  Test Design (ATCL)  TCD Generator. Этот инструмент представляет собой мастер, который автоматически создает для вас файл TCD.

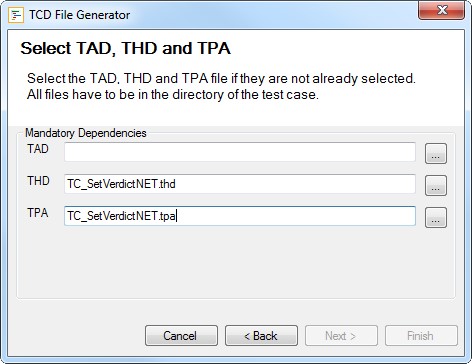
В первом диалоговом окне вы должны выбрать исполняемый файл тестового примера. Это файл, который считается «тестовым примером». Кроме того, вы должны выбрать исполнителя тестового примера, который используется для выполнения тестового примера. Для тестового примера ATCL C# .NET это будет стандартный исполнитель (StdExe).



Если помимо исполняемого файла тестового примера есть ровно один файл TPA, THD и TAD, эти файлы автоматически берутся в качестве зависимостей, и если вам не нужны никакие другие файлы для вашего тестового примера, просто нажмите «Готово».

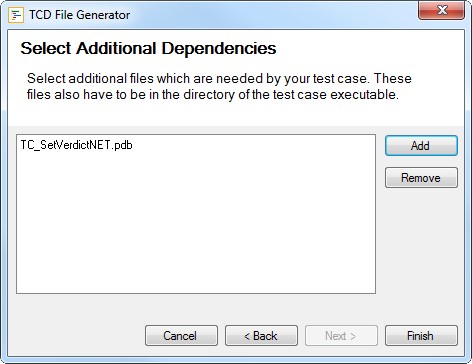
Если мастер не может обнаружить файл TAD, THD или TPA, вам необходимо выбрать их вручную. Для этого используйте кнопку обзора (…), чтобы открыть окно выбора файла, в котором можно выбрать файл. Помните, что все файлы должны находиться в каталоге тестового примера.

ATCL (Automotive Test Class Library)



Как только все необходимые зависимости определены, вы можете нажать «Готово», чтобы сгенерировать файл TCD, или вы можете перейти к следующему окну, где вы можете выбрать дополнительные зависимости для своего тестового примера.

В третьем окне вы можете выбрать дополнительные зависимости, нажав кнопку Добавить. Откроется диалоговое окно браузера файлов в каталоге тестового примера, где вы можете выбрать все типы файлов. Вы не должны выбирать какой-либо файл, который уже является зависимостью вашего тестового примера (например, файл TPA), а также вы не должны выбирать какие-либо файлы в подкаталогах или каталогах за пределами каталога тестового примера..



Сгенерированный файл TCD имеет то же имя, что и исполняемый файл тестового примера, но расширение файла изменено на «.tcd». Браузер функций Test Manager не сможет найти ваш тестовый пример, если вы не предоставите файл TCD! Но вам нужно сгенерировать файл TCD только один раз для тестового примера. Если вы измените свой тестовый пример, а затем перекомпилируете его, вы можете просто синхронизировать свой проект LABCAR-AUTOMATION, который автоматически обновит тестовый пример.

Но вам придется сгенерировать новый файл TCD, если вы добавите (или удалите) некоторые дополнительные ресурсы, такие как библиотеки DLL, в тестовый пример!

### Использование интерпретатора командной строки

Параллельно с TCDCMD.Gui.exe существует другое приложение TCDCMD.exe, которое представляет собой инструмент командной строки, генерирующий файл TCD без отображения мастера. Это можно использовать, например, в качестве шага после выполнения в вашей среде разработки Visual Studio.

Usage tcdcmd.exe <testlanguage> <entrypoint> <tpa> <thd> <tad> [<dependency1> [<more dependencies>]]

<testlanguage> Язык тестового примера. Возможные значения: «StdExe», «TTCN»,

«PythonNET» и «TestStand”

<entrypoint> Запуск файла тестового примера. Обычно исполняемый файл.

<tpa> Файл тестовых параметров тестового примера.

<thd> Определение тестовой иерархии тестового примера.

<tad> Определение тестовой архитектуры тестового примера.

<dependency> Дополнительный файл, необходимый тестовому набору для работы.

## Настройка LABCAR-AUTOMATION project

Чтобы действительно выполнить тестовый пример, лучше всего создать небольшой тестовый проект LABCAR AUTOMATION. Следующие шаги необходимы для создания минимального тестового проекта для одного тестового примера и одного UuT:

1. Используйте редактор UUT Editor, чтобы определить пример UUT Environment, содержащий все ключи для портов/инструментов. Возможно, вам потребуется создать новый список UUT.
2. Создайте тестовый проект в инструменте Test Manager и добавьте в него свой тестовый пример и UUT.
3. Используйте вкладку «Параметр», чтобы присвоить любое значение двум созданным вами параметрам поплавка.
4. Создайте тестовую кампанию, содержащую тестовый пример. Вы должны решить, использовать ли кампанию «TBInit» или нет. Это зависит от нашего тестового примера. Если тестовый пример выполняет «Создать()» и

«ConfigureTool()» сам по себе, мы используем кампанию, не относящуюся к TBInit. Но имейте в виду, что если тестовый пример не является тестовым набором TBInit, мы не можем изменить ключи TCF для этого тестового примера. Также можно создавать тестовые примеры, которые могут обрабатывать кампании как TBInit, так и не-TBInit, как указано выше.

ATCL (Automotive Test Class Library)

# Report Design

## Sections and Verdicts

Возможно, вы захотите разделить отчет на разные разделы, например. разделы, упомянутые в главе 4. Для каждого раздела вы можете создать новый вердикт, который может быть независимым от общего вердикта тестового примера. Это позволяет вам, чтобы раздел не прошел, а следующие разделы были пройдены. Из-за конечного автомата вердикта это невозможно с общим вердиктом тестового примера. Чтобы получить вердикт раздела, мы должны создать новый экземпляр вердикта.

Verdict sectionVerdict = new Verdict(VerdictCode.None);

try

{

Reporting.SectionBegin(MethodInfo.GetCurrentMethod().Name);

…

sectionVerdict.Pass();

Pass();//Overall test case verdict

}

catch (Exception ex)

{

sectionVerdict.Error(); throw ex;

}

finally

{

Reporting.SectionFinished(sectionVerdict.ActualVerdictCode.ToString, sectionVerdict);

}

Пример кода может быть применен к любому методу, который должен появиться в качестве раздела в отчете о тестовом примере. Имя раздела считывается методом «MethodInfo.GetCurrentMethod().Name» и возвращает имя текущего метода.

## Working with Tables

Чтобы отображать измеренные значения в отличие от допустимых диапазонов значений, вы можете использовать таблицы в своем отчете. Таблицы можно создавать с помощью менеджера отчетов. Таблицы создаются «внутри» службы отчетов, и доступ к ним осуществляется по имени. Созданная таблица не добавляется в отчет автоматически. Это необходимо сделать с помощью метода «AddTable», который добавляет таблицу с указанным именем в текущее место в отчете..

string tableID = "table\_ID";

Reporting.CreateTable(tableID, "Value Range Demo Table", 1, 0); Reporting.SetTableHeadline(tableID, "Measurement",0 , 1);

Reporting.SetTableHeadline(tableID, "Limits", 2, 4);

Reporting.SetTableHeadline(tableID, "Result", 5, 5);

Reporting.SetTableColDescription(tableID, new string[] { "Label", "Measure Value", "Tolerance", "Min",

"Max", "Verdict" });

Reporting.SetTableData(tableID, new string[] { "Angle A", "5.0", "7%", "4.65", "5.35", "pass"

});

Reporting.SetTableData(tableID, new string[] { "Angle B", "8.42", "1%", "8.32", "8.48", "pass"

});

Reporting.SetTableData(tableID, new string[] { "Angle C", "1.00", "5%", "3.00", "3.30", "fail"

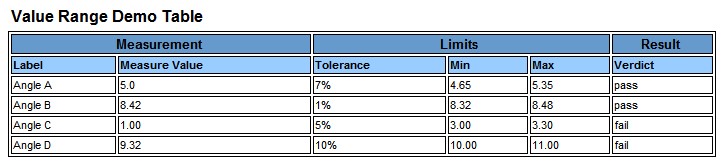
});

Reporting.SetTableData(tableID, new string[] { "Angle D", "9.32", "10%", "10.00", "11.00" ,

"fail"});

Reporting.AddTableToReport(tableID);

The resulting table will look like this:



## Working with Plots

Работа с графиками очень похожа на работу с таблицами. Вы также должны использовать диспетчер отчетов для создания нового графика. График также идентифицируется уникальным именем, но у нас дополнительно есть ссылка на объект графика в службе отчетов. Чтобы добавить данные на график, вы должны указать как минимум ось X и одну или несколько осей Y.

Если вы добавляете линию на график, вы должны выбрать ось Y, к которой относятся данные. Сам график добавляется в отчет, когда вы вызываете метод «AddPlot2Report» и добавляете график в текущее место в отчете; похоже на стол.

Обычно график используется для отображения данных, записанных регистратором данных.

TypeSut1DFloatTable [] table = ... ;

IPlot plot = Reporting.CreatePlot ( "Plot", "Engine Plot", 1.0 );

plot.AddYAxis ( "y",0.0, 1000.0,"rpm",100.0); plot.SetXAxis ( "x",0.0,11.0,"s",1.0); plot.YAxisCollection[0].AddLine(

"EngineSignal", table[0].ValueX, table[0].ValueY,

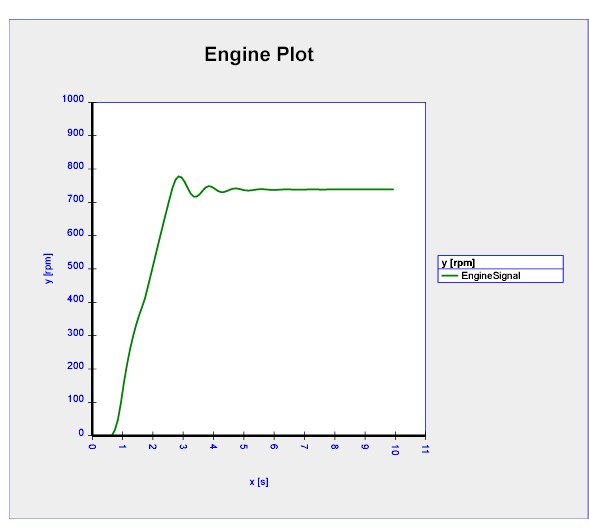
0.0,

0.0,

plot.CreateLineFormat("green", LineWeight.Thin, LineStyle.Stroke));

Reporting.AddPlot2Report (plot);

Отображаемые данные записывались регистратором данных и извлекались с помощью метода «GetLoggedSignals(...)». Полученный график будет выглядеть как:



Using own software components

# Using own software components

В ЛАБКАР-АВТОМАТИЗАЦИЯ и АТКЛ можно интегрировать собственные программные компоненты. Вы можете написать свой собственный порт и адаптеры для инструментов, чтобы интегрировать новые инструменты. Вы можете создавать свои собственные менеджеры и интегрировать их непосредственно в фабрику ATCL, а также использовать другие библиотеки DLL в своем тестовом примере..

## ATCL Managers

Фабрика ATCL по умолчанию предоставляет менеджерам информацию о параметризации, отчетах и ​​метаданных. Вы можете расширить список доступных менеджеров. Менеджеры

анонимные программные компоненты, реализующие

Интерфейс «Etas.Eas.Atcl.Interfaces.Managers.Imanager». Вы можете получить экземпляр менеджера из фабрики ATCL, а затем вам нужно привести этот экземпляр к интерфейсу, который реализует этот менеджер.

Например, вы можете использовать «Factory.GetReportManager()», чтобы получить экземпляр

«IreportingManager» или вы можете использовать общий

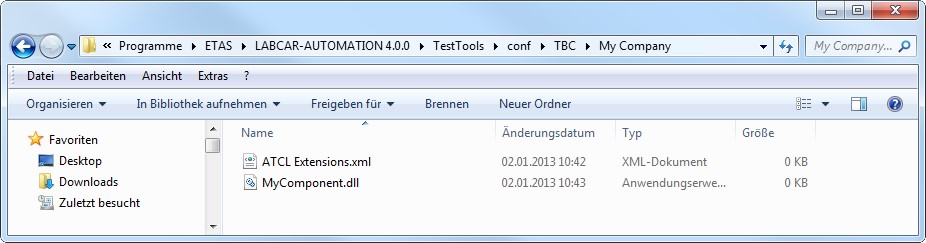
«Factory.GetManager(«IReportingManager»)», чтобы получить тот же экземпляр. Разница в том, что вы должны привести ссылку к интерфейсу «IreportingManager».

Менеджеры используются, когда вы хотите предоставить функциональность, которая не обязательно требует адаптера инструмента или порта. Например, если вы хотите создать программный компонент, который включает в себя генерацию таблиц отчетов. Эта функциональность должна быть инкапсулирована в отдельную сборку и доступна через концепцию менеджера.

Пользовательский менеджер можно вставить в ATCL, добавив файл ATCL Extension.xml где-нибудь в каталоге <Install Path>/TestTools/conf. В этом XML-файле вы можете объявить свой собственный менеджер, а затем вы можете получить доступ к этому менеджеру через метод «Factory.GetManager(«<уникальное имя менеджера»)».

Пример расширения ATCL можно найти в файле

TestTools/conf/TBC/ETAS/ATCL/ATCL Extensions.xml, который определяет порт FS. Сборка, включающая необходимый менеджер, также должна находиться где-то под каталогом conf. Мы предлагаем хранить ATCL Extension.xml вместе с вашей DLL в новом подкаталоге.



Поскольку менеджеры публикуются по конфигурации, они зависят от установки и компьютера. Таким образом, вы должны убедиться, что на машине, на которой выполняется ваш тестовый пример, настроены необходимые менеджеры.

## Ports and Tool Adapters

Всякий раз, когда вы хотите получить доступ к новому инструменту, вы должны не торопиться и уточнить, доступны ли функции, которые предоставляет инструмент, в существующем порте. Если это так, вы должны написать новый адаптер инструмента для этого инструмента, и все готово. Просто сопоставьте существующий порт с новым адаптером инструмента и используйте сигнатуры портов для управления новым инструментом.

Если существующего порта нет, вам следует подумать о создании нового порта и адаптера инструмента, чтобы получить доступ к вашему инструменту из тестового примера ATCL. Хотя предпринимаются попытки создать

новый порт и адаптер инструмента, это должно быть предпочтительным способом, потому что это гарантирует, что ваши тестовые примеры останутся независимыми от инструмента.

Новые порты определяются точно так же, как менеджеры в файле ATCL Extension.xml. Единственное отличие состоит в том, что порты запрашиваются у фабрики ATCL с помощью метода «GetPort (<идентификатор>, <имя экземпляра>)». «идентификатор» — это идентификатор объекта в ATCL Extension.xml.

<object id="IPortFS"

type="Etas.Eas.Atcl.Core.Port.FailureSimulation.FailureSimulationPor t, Etas.Eas.Atcl.Core" singleton="false" />

Для каждого порта требуется дополнительный XML-файл, который определяет доступные подписи, в каких состояниях разрешено вызывать эти подписи и некоторую дополнительную информацию. Файл XML полностью описывает новый порт. Поскольку создание этого XML занимает много времени и подвержено ошибкам, у нас (ETAS) есть генератор XML, который создает все необходимые файлы из интерфейса порта. Этот генератор НЕ является частью установки продукта, но мы создадим для вас файлы бесплатно, если вы предоставите реализацию интерфейса C# порта.

Создание портов не является повседневным занятием тестировщиков и дизайнеров тестовых наборов, поэтому здесь мы ограничимся кратким описанием. Пожалуйста, не стесняйтесь обращаться в ETAS для уточнения консультационных и инженерных услуг для поддержки вашего варианта использования здесь.

## Using other DLLs

В дополнение к концепции менеджера и порта вы также можете напрямую ссылаться на необходимую сборку, добавляя ссылку в проект Visual Studio. Если для вашего теста требуются дополнительные сборки ресурсов, вы должны указать их в файле TCD. В противном случае сборки, на которые есть ссылки, не считаются частью вашего тестового примера и, следовательно, не копируются в проект LABCAR-AUTOMATION при добавлении тестового примера. Преимущество прямой ссылки на сборку заключается в том, что вы можете выполнить этот тестовый пример на любом испытательном стенде LABCARAUTOMATION.

ETAS Contact Addresses

# ETAS Contact Addresses

ETAS HQ

|  |  |  |
| --- | --- | --- |
| ETAS GmbH  Borsigstraße 14 | Phone: | +49 711 89661-0 |
| 70469 Stuttgart | Fax: | +49 711 89661-106 |
| Germany  ETAS Subsidiaries and Technical | WWW:  Support | www.etas.com |

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

|  |  |  |
| --- | --- | --- |
| ETAS subsidiaries | WWW: | [www.etas.com/en/contact.php](http://www.etas.com/en/contact.php) |
| ETAS technical support | WWW: | [www.etas.com/en/hotlines.php](http://www.etas.com/en/hotlines.php) |

LABCAR-AUTOMATION 4.2.2

Getting Started for Test Case Developers