ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
Corso di Laurea Magistrale in Ingegneria delle Telecomunicazioni

Tesi di Laurea Magistrale

SISTEMA INTERATTIVO DI COLLAUDO PER MODULI DI LETTURA DI CODICI OTTICI

Relatore:
Chiar.ma Prof.ssa
Carla Raffaelli

Candidato:
Guido Trentalancia

Correlatore:
Dott. Ing.
Marco Bozzoli

Sessione III
Anno Accademico 2013/2014

Dedicata ai miei genitori.

Indice

Introduzione

Questa tesi di laurea magistrale ha come oggetto la progettazione e l'implementazione di un sistema elettronico interattivo di collaudo per moduli di lettura di codici ottici, quali ad esempio i codici a barre.

Tali moduli sono di tipo fotografico, ovvero includono un sensore di immagine di tipo CMOS (complementary metal-oxide semiconductor). Attraverso il sensore di immagine, i moduli di lettura dei codici ottici acquisiscono una fotografia di un'area dove si suppone sia presente informazione codificata mediante un codice noto al sistema di decodifica posto a valle di tali moduli di lettura.

Per realizzare un tale sistema di collaudo è stato appositamente sviluppato un progetto software denominato *Scan Engine Test Program*, per calcolatori elettronici di tipo Personal Computer con sistema operativo Microsoft® Windows®, in grado di comunicare tramite interfacce standard con il modulo di lettura ottico esterno.

Per mezzo di una interfaccia grafica, appositamente sviluppata, il sistema interattivo può essere utilizzato con facilità da un operatore umano che diventa così in grado di progettare e condurre la sessione di collaudo, selezionando di volta in volta le prove da effettuare per validare il dispositivo collegato al sistema. Il sistema di prova è inoltre capace di produrre dei resoconti sull'esito delle prove effettuate.

Prima e durante la fase di sviluppo sono stati raccolti ed analizzati i requisiti di sistema in modo da ottenere delle specifiche riguardo alle funzionalità richieste. Dopo ogni nuovo requisito raccolto o

in seguito al cambiamento di requisiti precedentemente raccolti, ha avuto luogo la relativa fase di progettazione. Alla fine di ogni ciclo di progetto è seguita la fase di implementazione delle funzionalità richieste. Infine, ciascuna delle funzionalità implementate nel sistema software è stata validata. Tutte le problematiche emerse durante la validazione oppure in qualunque altra fase dello sviluppo, sono state opportunamente documentate e risolte intervenendo sulla fase di implementazione ed eventualmente anche sulla relativa fase di progetto.

Questa tesi ed il lavoro di progettazione ed implementazione in software del sistema di cui sopra sono stati svolti presso il reparto Ricerca e Sviluppo dell'azienda Datalogic S.p.A. di Lippo di Calderara di Reno (Bologna) che produce e commercializza sistemi elettronici destinati alla lettura automatica e semiautomatica di codici ottici.

Il sistema interattivo di validazione oggetto di questa tesi è destinato all'utilizzo, presso gli stabilimenti produttivi di Datalogic, durante la fase di collaudo dei moduli di lettura.

Capitolo 1

Descrizione del sistema

1.1 Brevi cenni sui codici ottici

Per *codici ottici* o *codici a lettura ottica* si intendono in questo contesto sistemi di codifica dell'informazione in forma grafica finalizzati ad agevolare la lettura automatica di tale informazione. Il primo tipo di codice ottico ad essere stato definito ed utilizzato è stato il codice a barre che è stato inventato nel 1949 da Norman Joseph Woodland e Bernard Silver. Le prime applicazioni commerciali videro la luce negli anni 1970 con l'avvento dei circuiti integrati e della tecnologia laser a basso costo.

Si parla di "simbologia" per fare riferimento al tipo di codice a barre, ovvero allo standard di rappresentazione dei dati; la simbologia definisce implicitamente il tipo di dati che possono essere memorizzati. Ogni simbologia usa un proprio numero di elementi differenti (barre di larghezza differente e spazi di ampiezza diversa).

Le simbologie più comuni per i codici a barre lineari sono del tipo UPC (Universal Product Code) ed EAN (European Article Number). Per tali simbologie l'informazione non codificata è ripetuta in forma testuale sotto al codice stesso.

UPC è un insieme di simbologie utilizzato in particolare negli Stati Uniti e nel Canada. Con le varianti UPC-A (la più comunemente usata) ed UPC-E si possono rappresentare solo cifre numeri-

Figura 1.1: Simbologie UPC-A (sinistra) ed UPC-E (destra).

Figura 1.2: Simbologie EAN-13 (sinistra) ed EAN-8 (destra).

che: UPC-A consente di rappresentare 11 cifre più una di controllo; UPC-E consente di rappresentare 6 cifre, senza codice di controllo. Gli elementi utilizzati nella simbologia UPC sono otto: quattro tipi di barre nere e quattro tipi di spazi.

L'insieme di simbologie EAN è invece di origine europea ma utilizzato anche in altri paesi ed è nato come estensione delle simbologie UPC.

Le varianti EAN-13 (la più comunemente usata) ed EAN-8 permettono di rappresentare anch'esse solo cifre numeriche: EAN-13 consente di rappresentare 12 cifre più una di controllo, mentre EAN-8 consente di rappresentare 7 cifre numeriche più una di controllo.

Esempi di codifica nelle quattro simbologie sopra descritte sono mostrati rispettivamente in Figura 1.1 (UPC) ed in Figura 1.2 (EAN).

L'applicazione di più vasto utilizzo dei codici a barre consiste nell'identificazione univoca delle unità commerciali (prodotti e ser-

Prefisso aziendale	Codice prodotto	Cifra di controllo
9 cifre	3 cifre	1 cifra

Tabella 1.1: Struttura tipica di GTIN per identificazione di prodotti commerciali.

vizi). A tale scopo vengono utilizzate le sequenze numeriche denominate GTIN (Global Trade Item Number).

Al di fuori degli Stati Uniti, le sequenze numeriche GTIN vengono di solito codificate a barre mediante la simbologia EAN-13 e pertanto a ciascun prodotto viene in questo caso assegnato un codice con una struttura simile a quella riportata in Tabella 1.1.

Il codice è assegnato dall'azienda a ciascun prodotto che deve essere identificato e sono disponibili 1000 codici, da assegnare in ordine progressivo. Nel caso in cui si esaurisca la banda numerica disponibile, è possibile noleggiare un prefisso aziendale supplementare.

Negli Stati Uniti invece, dove è in uso la codifica UPC, si utilizzano per i GTIN prefissi aziendali di lunghezza variabile da 6 a 10 cifre e dunque anche il numero di prodotti catalogabili sarà variabile, da 100000 a 10.

I prefissi aziendali iniziano comunque sempre con l'identificativo della nazione (di registrazione e non necessariamente di origine del prodotto) e sono assegnati dall'organizzazione internazionale GS1. Un identificativo iniziale speciale è riservato per i libri (sistema di codifica ISBN) ed i periodici (sistema di codifica ISSN).

Altre applicazioni dei codici ottici sono in ambito logistico, nella gestione magazzini, nelle operazioni di carico/scarico/smistamento delle merci, nella gestione dell'inventario e dei listini, nello smistamento dei bagagli, nelle ricerche di mercato e nel controllo degli accessi, in ambito medico e sanitario per l'identificazione dei pazienti, dei campioni di materiale biologico e dei farmaci.

Altri tipi di codici ottici, quali ad esempio i codici a matrice o bidimensionali, sono stati definiti successivamente anche se ad oggi i più diffusi rimangono ancora i codici a barre o monodimensionali.

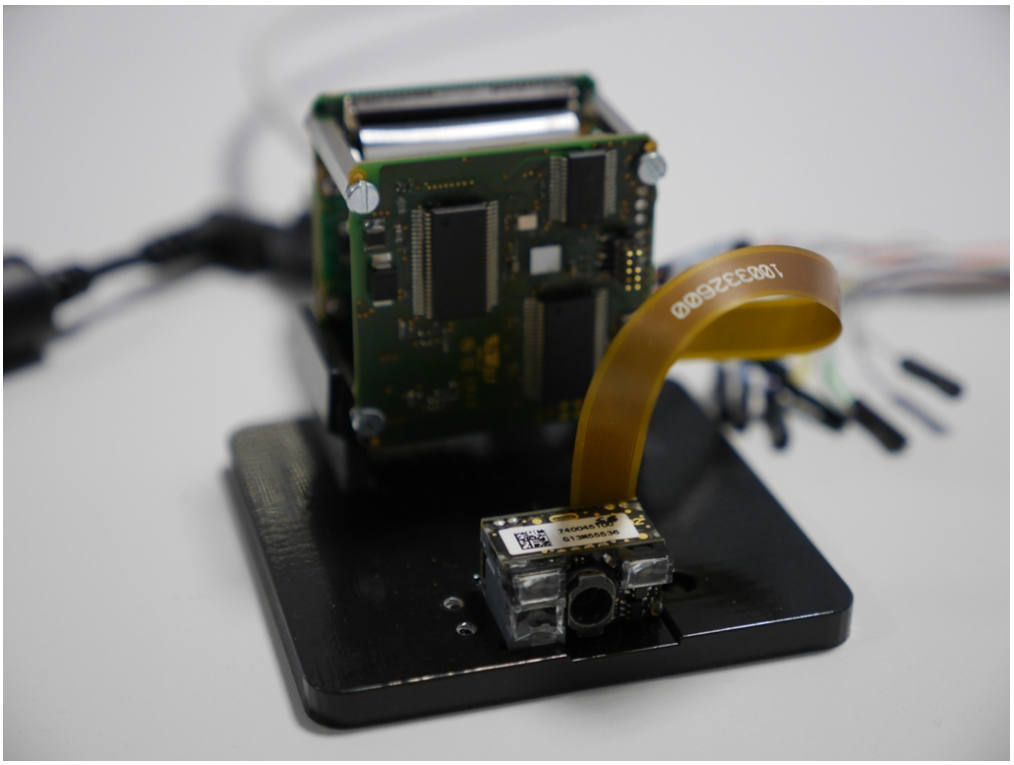

Figura 1.3: Vista frontale del dispositivo da collaudare (in basso).

1.2 Il modulo di lettura da collaudare

Il modulo di lettura da collaudare è un dispositivo elettronico che include una telecamera, un sistema di illuminazione ed un sistema di puntamento laser. Esso è il dispositivo più piccolo visibile in basso nella Figura 1.3.

Come già accennato nella Introduzione, il sensore di immagine, da cui dipende la qualità dell'immagine acquisita, è di tipo CMOS (complementary metal-oxide semiconductor).

Allo stato attuale della tecnologia, i sensori di immagine comunemente utilizzati nelle varie applicazioni elettroniche possono essere di due tipi: CMOS e CCD (charge coupled device). Il principio di funzionamento di entrambi i tipi di sensori è basato sulla generazione di un segnale elettrico a partire dalla luce incidente, tramite l'effetto fotoelettrico. Entrambi i tipi di sensori sono stati inventati alla

fine degli anni 1960 e negli anni 1970 per sostituire le telecamere a valvole.

I sensori di tipo CMOS sono apparsi sul mercato nei primi anni 1990 andando così ad affiancare i già presenti sensori di tipo CCD. Rispetto a questi ultimi, essi offrono vantaggi quali [1, 2, 3]:

- una più bassa dissipazione di potenza;
- un più basso costo di produzione;
- una alta integrabilità su un singolo circuito integrato di sistemi completi che includono spesso anche amplificatori, riduttori di rumore, circuiti di digitalizzazione, circuiti per la generazione dei segnali di temporizzazione e circuiti di interfacciamento.

La possibilità di integrare tutte le funzioni circuitali su di un singolo circuito integrato non solo permette di ridurre le dimensioni ed aumentare la velocità, ma soprattutto rende i sensori basati sulla tecnologia CMOS più robusti e più adatti a lavorare in condizioni ambientali più difficili, minimizzando ad esempio il rischio di rottura dei punti di saldatura [4]. Un tipico schema di sensore CMOS è riportato in Figura 1.4.

Nei sensori di tipo CMOS, in ogni pixel (o punto che compone l'immagine) vi è la conversione da carica elettrica a voltaggio. Nei sensori di tipo CCD invece, la carica di ciascun pixel viene trasferita sequenzialmente ad una unica struttura di uscita comune per essere poi convertita in voltaggio, immagazzinata in una memoria e infine inviata fuori dal circuito integrato.

La massima risoluzione dei sensori presenti nei moduli di scansione è al momento di 752x480 pixel (Wide VGA) ed essi riescono a catturare fino a 60 fotogrammi per secondo [5]. Nuovi moduli in fase di sviluppo e che saranno disponibili a brevissimo termine raggiungeranno risoluzione dell'ordine di 1 megapixel.

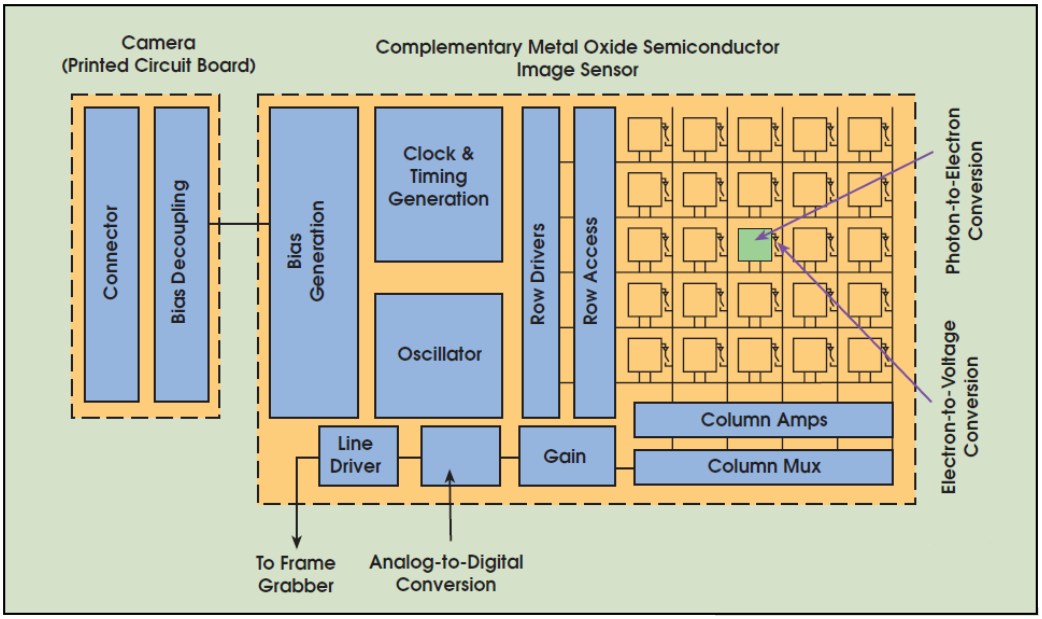

Figura 1.4: Schema a blocchi di un sensore di immagine CMOS (ristampato con permesso di Teledyne DALSA da [4]).

Il sistema di illuminazione presente nel modulo di scansione è costituito da due LED bianchi in grado di assicurare il corretto funzionamento del modulo anche nella totale oscurità.

Il sistema di puntamento laser invece è basato su un diodo laser a 650 nm di classe 2 (massima potenza 1 mW) che proietta quattro vertici di un rettangolo rappresentante il campo di vista ed una croce centrale. Esso assiste l'operatore nel dirigere il lettore di codici a barre verso l'obiettivo.

Un microcontrollore Atmel® della famiglia AVR® XMEGA® pilota il modulo e gestisce la comunicazione con gli altri dispositivi esterni, per mezzo di un protocollo denominato I^2C (Inter Integrated Circuit) e descritto nel paragrafo 1.3. Come vedremo meglio successivamente, tramite tale protocollo avviene appunto la comunicazione con il sistema di validazione.

AVR® è un tipo di CPU (unità centrale di elaborazione) basato su una architettura di tipo Harvard modificata e sulla tecnologia RISC (reduced instruction set computer).

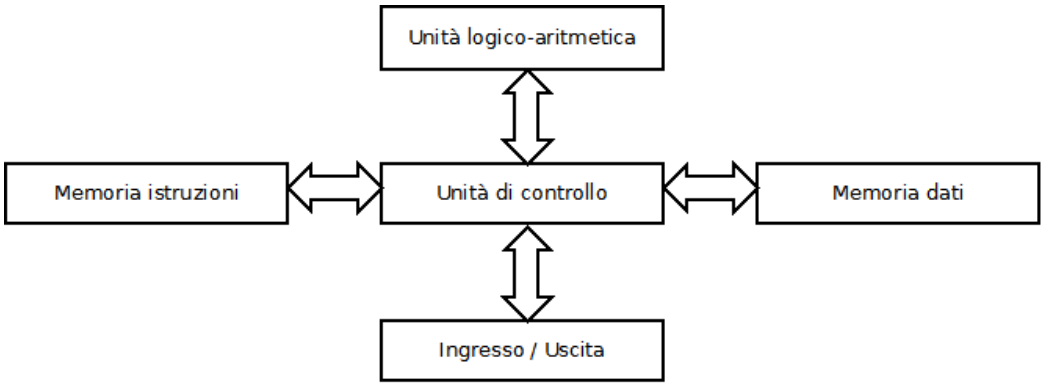

Figura 1.5: Architettura Harvard dei calcolatori elettronici.

L'architettura Harvard[1] è un tipo di architettura dei calcolatori elettronici caratterizzata da spazi di memorizzazione e percorsi separati dei relativi segnali per istruzioni e dati (Figura 1.5).

Il fatto che la memoria per istruzioni e dati sono separate implica in primo luogo che istruzioni e dati possono avere diversa dimensione. In una tale architettura è inoltre possibile eseguire parallelamente le operazioni di lettura e scrittura dei due tipi di memoria (ad esempio si può leggere la successiva istruzione mentre vengono letti o scritti dati).

Il microcontrollore utilizza una architettura Harvard "modificata" nel senso che si può accedere alla memoria di programma anche come memoria dati. Esso costituisce un sistema completo che raggruppa in un unico circuito integrato il processore, la memoria permanente, la memoria volatile, i canali di ingresso e uscita ed altri blocchi specializzati.

La caratteristica di essere basato sulla tecnologia RISC implica che esso adotta un insieme di istruzioni ridotto, le quali per questo motivo riescono ad essere eseguite in tempi più brevi rispetto a quanto avviene nella tecnologia contrapposta e denominata CISC (complex instruction set computer). Il principale svantaggio dell'approccio RISC è la maggiore occupazione di memoria da parte

[1] Il nome deriva dal calcolatore elettronico a relè denominato Harvard Mark I.

del codice eseguibile (poiché l'insieme di istruzioni a disposizione essendo ridotto è meno "espressivo").

L'interfaccia I^2C del microcontrollore può lavorare a due specifiche frequenze: 100kHz e 400kHz. La funzionalità di riconoscimento della trasmissione del proprio indirizzo sull'interfaccia I^2C è implementata in hardware e permette anche di "risvegliare" il dispositivo dallo stato di dormienza.

Il modulo di scansione dispone inoltre di una porta video parallela composta da:

- 8 bit per pixel;
- segnali di sincronismo verticale ed orizzontale;
- impulso di temporizzazione del pixel (*pixel clock*).

La tensione di alimentazione del modulo di scansione (ed il tipico livello alto per i segnali di ingresso ed uscita) è di 3.3 volt.

Sono due i principali tipi di applicazioni in cui vengono utilizzati i moduli di scansione considerati: *mobile* e *hand-held*. Il sistema di validazione per moduli di scansione sviluppato in questo progetto di tesi potrà essere configurato per funzionare con tutti i differenti tipi di moduli disponibili e andrà a beneficiare tutte le diverse applicazioni possibili. La Figura 1.6 mostra il sistema hardware completo per la validazione dei moduli di scansione: a sinistra è visibile il modulo di scansione (dispositivo da collaudare); a destra è visibile l'interfaccia di controllo (descritta nel paragrafo 1.3); dietro il dispositivo è posizionata l'interfaccia di acquisizione immagini (descritta nel paragrafo 1.4).

1.3 Interfaccia di controllo (I^2C)

Il sistema di collaudo del modulo di scansione è costituito da un calcolatore elettronico di tipo Personal Computer, collegato al modulo di scansione, su cui viene eseguita una applicazione di vali-

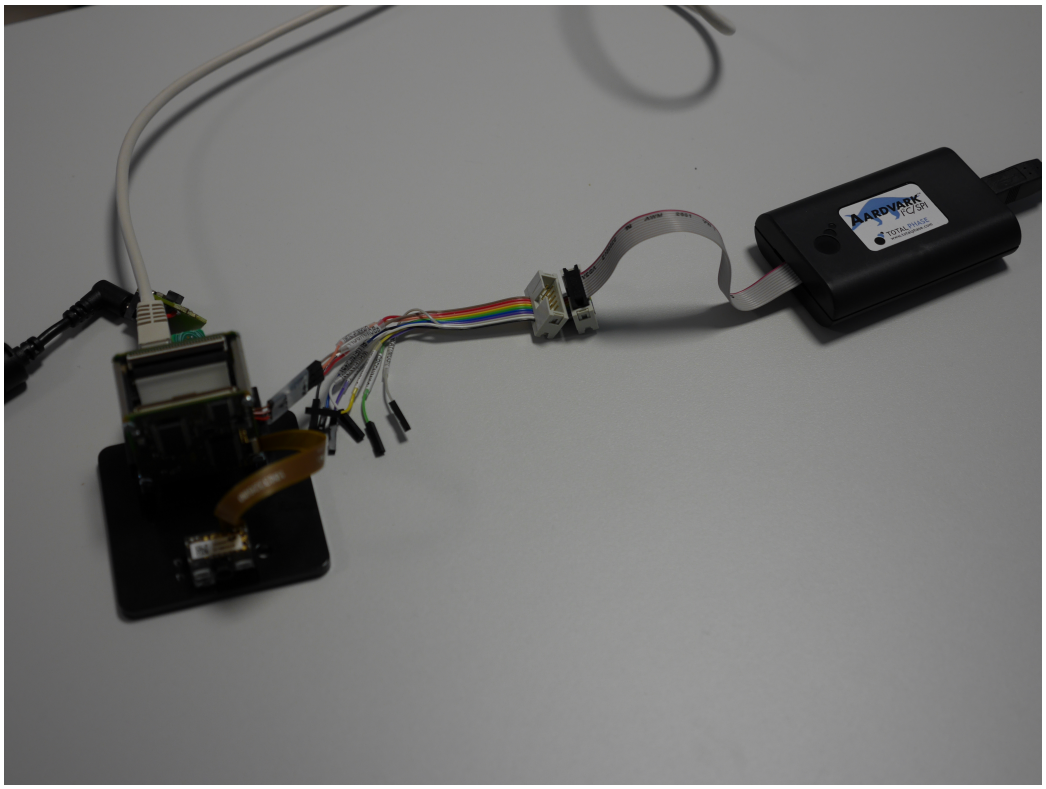

Figura 1.6: Il dispositivo da collaudare (in basso a sinistra) nel sistema di collaudo.

dazione dotata di interfaccia grafica con l'utente (*Scan Engine Test Program*). Lo schema a blocchi di un tale sistema è riportato in Figura 1.7.

Il protocollo di comunicazione di più basso livello adottato per tale interfaccia di controllo è I^2C (Inter Integrated Circuit). Esso è un protocollo di tipo seriale half-duplex introdotto da Philips Semiconductors (ora NXP Semiconductors) nel 1982.

Più interfacce basate sul protocollo I^2C possono eventualmente essere presenti sul calcolatore elettronico sul quale viene eseguito il sistema software di validazione, tuttavia almeno una di esse deve essere dedicata al controllo del dispositivo in prova.

Un bus I^2C è formato da due linee bidirezionali SCL (Serial CLock) e SDA (Serial DAta) che trasportano rispettivamente la tempistica di sincronizzazione ed i dati. Ogni dispositivo collegato a

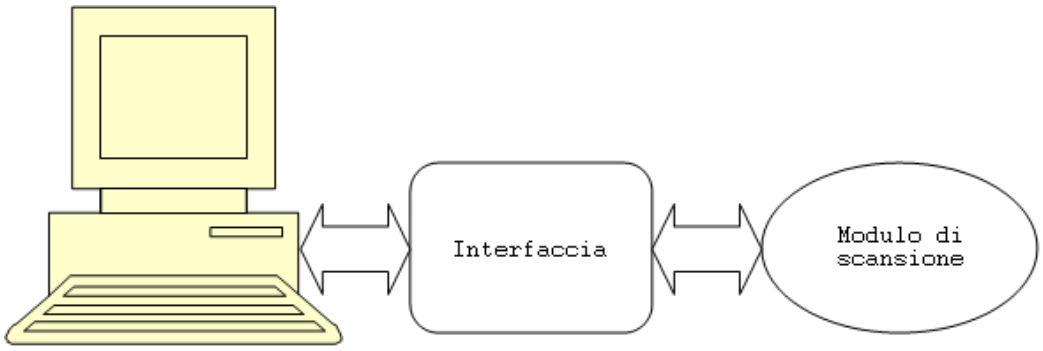

Figura 1.7: Schema a blocchi del sistema complessivo.

queste linee ha un indirizzo univoco di 7 o 10 bit e può agire secondo uno dei due ruoli seguenti: "*master*" (emissione del segnale di clock ed iniziazione della comunicazione) o "*slave*" (ricezione delle richieste e trasmissione delle relative risposte).

Il modulo di scansione si comporta sempre come dispositivo *slave* e non assume mai il ruolo di dispositivo *master*. Tale ruolo viene invece assunto dal dispositivo che funge da interfaccia di controllo.

La trasmissione inizia con la generazione di un bit di avvio (start bit), immesso sulla linea dal dispositivo *master*, dopo un controllo sull'occupazione del canale (o "bus"). Tale segnale di avvio consiste in una transizione dallo stato alto allo stato basso sulla linea SDA mentre la linea SCL è nello stato alto (Figura 1.8).

Dopo la generazione del bit di avvio segue la trasmissione dell'indirizzo del dispositivo *slave* e successivamente segue l'invio di un bit che indica se il dispositivo *master* intende trasmettere allo *slave* (valore basso) oppure intende ricevere dallo *slave* (valore alto).

L'ordine di trasmissione dei bit (di indirizzo e di dati) è quello dal più significativo al meno significativo.

Se il dispositivo *slave* indirizzato esiste, prende il controllo della linea dati sul successivo impulso alto del SCL e la forza bassa (segnale di conferma, ACK). A questo punto il dispositivo *master* è a conoscenza del fatto che il dispositivo *slave* selezionato ha ricevuto la richiesta ed è in attesa di rispondere. Se il segnale di confer-

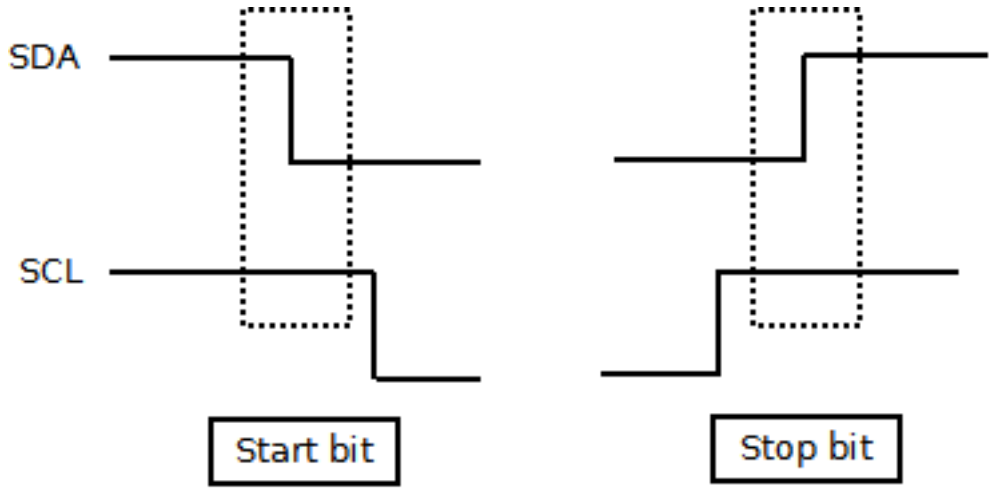

Figura 1.8: Condizioni di avvio e fine trasmissione (protocollo I^2C).

ma (ACK bit) non venisse generato, il dispositivo che ha iniziato la trasmissione può interromperla, utilizzando la condizione di bit di arresto (stop bit).

Il segnale di bit di arresto consiste nella generazione di una transizione dallo stato basso allo stato alto sulla linea SDA mentre la linea SCL è nello stato alto (Figura 1.8).

A seguito della ricezione del primo segnale di conferma (ACK bit) il dispositivo *master* continua in modalità trasmissione dati o ricezione dati (a seconda di come era configurato il bit corrispondente) ed il dispositivo *slave* continua nella modalità duale (ricezione o trasmissione, rispettivamente). In altre parole, seguono i dati veri e propri (byte da 8 bit) che saranno a loro volta seguiti da un ulteriore bit di conferma (ACK bit).

Alla fine della trasmissione/ricezione dei dati e della ricezione/-trasmissione del relativo bit di conferma, vi è la trasmissione da parte del dispositivo *master* del segnale di arresto (stop bit).

Si possono verificare due tipi di problematiche di mancata temporizzazione: sul byte di indirizzo (ad esempio, il dispositivo *slave* è occupato e non risponde entro il prefissato tempo limite) oppure sulla intera sequenza trasmessa fino al bit di arresto.

Bit di avvio	Indirizzo	Comando	Parametri	...	Somma di controllo	Bit di arresto

Tabella 1.2: Formato delle sequenze di comando da inviare al modulo di scansione.

Bit di avvio	Indirizzo	Comando	Stato	Dati risposta	...	Bit di arresto

Tabella 1.3: Formato delle risposte provenienti dal modulo di scansione.

Nella comunicazione tra sistema di validazione e modulo di scansione vengono utilizzati due possibili tipi di sequenze di dati:

- sequenze di comando (generate dal sistema di validazione);
- sequenze di risposta (generate dal modulo di scansione).

I formati relativi ad i due tipi di sequenze sono riportati rispettivamente in Tabella 1.2 ed in Tabella 1.3 [5].

Nelle due tabelle, il campo *Indirizzo* include l'indirizzo I^2C vero e proprio seguito rispettivamente dalla opzione di scrittura (byte 0x00) nel primo caso e di lettura (byte 0x01) nel secondo caso. Il campo *Stato* della tabella relativa alle risposte è un byte codificato come indicato nella Tabella 1.4 [5].

Il dispositivo di interfacciamento fisico utilizzato in questo progetto per l'adattamento I^2C è prodotto dalla azienda Total Phase Incorporated (USA) ed è denominato *Aardvark I^2C/SPI Host Adapter* (vedi Figura 1.9). Esso utilizza una porta USB (Universal Serial Bus) per il collegamento con il calcolatore elettronico.

Per poter interfacciare a livello software l'applicazione con la libreria del dispositivo viene utilizzata una libreria di incapsulamento

Stato	**Valore**	**Significato**
ACK	0x80	Il comando ha avuto esito positivo
NACK	0x82	Il comando ha avuto esito negativo (comando o parametro invalido)
CKSM_ERR	0x84	Esito negativo della verifica della somma di controllo

Tabella 1.4: Possibili valori di stato delle risposte provenienti dal modulo di scansione.

Figura 1.9: Il dispositivo di interfacciamento I^2C denominato *Aardvark* (sulla destra è visibile la porta USB).

sviluppata in C++ all'interno di questo progetto ed utilizzabile grazie ai meccanismi di *interoperabilità* forniti da C# e dall'ambiente .NET Framework (come descritto dettagliatamente nel paragrafo 2.5).

1.4 Interfaccia di acquisizione immagini

Un secondo tipo di interfaccia, quando eventualmente presente, può svolgere invece funzioni di acquisizione video. Le immagini acquisite potranno poi essere analizzate con vari algoritmi secondo le specifiche di collaudo. L'applicazione sviluppata in questo progetto fornisce entrambe queste due funzionalità: l'acquisizione immagini e la loro analisi.

In questo progetto il dispositivo di interfacciamento fisico utilizzato per l'acquisizione video è prodotto da Pleora Technologies Incorporated (Canada) ed è del tipo *iPORT^{TM} NTx-Mini*. Esso è visibile in Figura 1.10, collegato al modulo di scansione sulla sua destra e al calcolatore elettronico sulla sua sinistra.

Il trasferimento delle immagini dalla scheda di acquisizione al calcolatore elettronico avviene tramite interfaccia Ethernet (IEEE 802.3) con User Datagram Protocol (UDP) su protocollo Internet (IP) versione 4 alla velocità massima di 1 Gb/s in conformità allo standard GigE Vision®. La scheda *iPORT^{TM}* utilizzata dispone di una memoria tampone del tipo DDR2 e della capacità di 32 MB per immagazzinare i fotogrammi.

Vengono accettati in ingresso, dalla porta video parallela del modulo di scansione, segnali a basso voltaggio del tipo LVCMOS o LVTTL alla frequenza di 90 MHz e con profondità fino a 24 bit per pixel, in bianco e nero o a colori.

Il flusso di fotogrammi viene pertanto trasformato per la trasmissione al calcolatore elettronico in un flusso di datagrammi UDP ed un protocollo di livello superiore denominato GVCP (GigE Vision® Control Protocol) garantisce il trasferimento affidabile di tutti i fotogrammi.

Oltre a garantire il trasferimento affidabile dei datagrammi tramite IP, il protocollo GVCP è utilizzato anche per configurare i dispositivi compatibili con lo standard GigE Vision® che vengono individuati nella rete; il parametro più importante della configurazione, soprattutto dal punto di vista della connettività, è l'indirizzo IP del dispositivo. Un'ulteriore funzione svolta da tale protocollo è quella di permettere la creazione di uno o più canali di controllo, primari o secondari, da potersi utilizzare rispettivamente per leggere e scrivere o per leggere solamente i registri del dispositivo.

Il protocollo GVCP è affiancato dal protocollo GVSP (GigE Vision® Streaming Protocol) basato anch'esso su UDP a livello inferiore. GVSP serve a permettere all'applicazione di ricevere i foto-

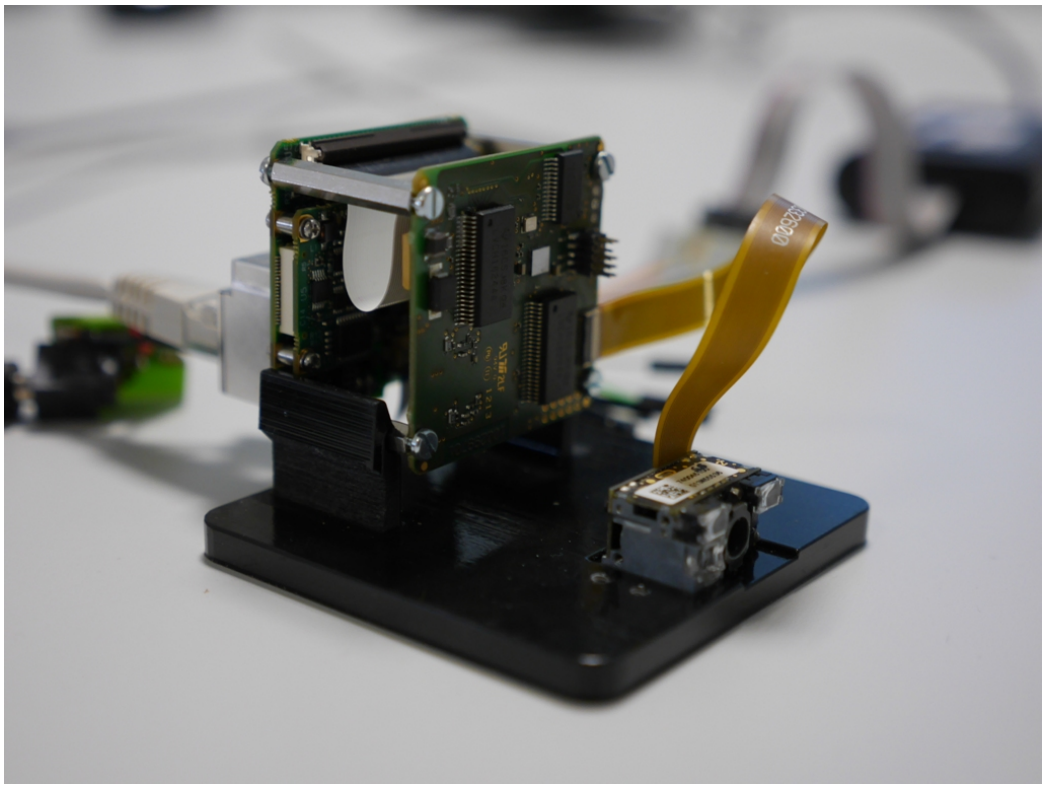

Figura 1.10: Interfaccia di acquisizione immagini (a sinistra) e modulo di scansione (a destra).

grammi, le informazioni relative ai fotogrammi (metadati) e marche temporali sui fotogrammi.

Sia GVCP che GVSP sono protocolli dello strato di applicazione secondo il modello di riferimento Open Systems Interconnection (OSI). La Figura 1.11 rappresenta tali due protocolli in riferimento a quelli di livello inferiore su cui si appoggiano (rispettivamente di livello OSI 4, livello OSI 3 e livelli OSI 2 e 1).

Il tipo di cavo raccomandato per l'utilizzo nel collegamento Ethernet è di categoria 6, a doppia schermatura con 4 coppie bilanciate (S/STP). Rispetto ad un cavo di categoria 5, esso offre infatti minore attenuazione, minore interferenza da canale adiacente, maggiore perdita di ritorno, larghezza di banda raddoppiata fino a 200 MHz e rapporto segnale-rumore migliorato di oltre 12dB.

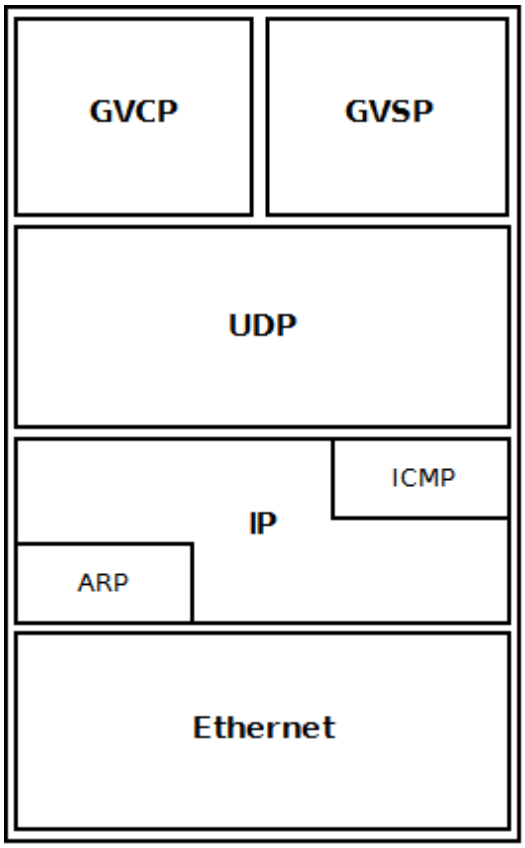

Figura 1.11: Pila comprendente i due protocolli GVCP e GVSP dello standard GigE Vision®.

L'azienda costruttrice della scheda di acquisizione immagini fornisce al momento due diverse versioni di software Application Programming Interface (API): versione 3 e versione 4. Per fornire la funzionalità di acquisizione immagini alla applicazione *Scan Engine Test Program* è stata sviluppata durante questo progetto un'apposita libreria di incapsulamento in C# la quale supporta entrambe le versioni di software API.

Si possono distinguere le seguenti fasi operative:

- ricerca dei dispositivi *iPORT*^{TM} *NTx-Mini* disponibili in rete e selezione del dispositivo da utilizzare;
- connessione al dispositivo *iPORT*^{TM} *NTx-Mini* prescelto;

- configurazione del dispositivo a cui si è connessi;
- ricezione dei fotogrammi acquisiti.

Per la fase di ricerca e selezione della scheda di acquisizione immagini la libreria di incapsulamento utilizza una classe fornita dalla API del produttore la quale permette di creare una apposita finestra ausiliaria tramite cui è possibile selezionare uno dei dispositivi trovati e configurarne il relativo indirizzo IP. La selezione è richiesta normalmente una sola volta al primo utilizzo della funzionalità di acquisizione immagini. Dopo che la scheda di acquisizione è stata individuata e selezionata, il sistema può effettuare la connessione tramite la rete IP. La connessione include anche una fase in cui viene aperto un flusso di dati tra l'indirizzo IP del calcolatore elettronico e l'indirizzo IP corrispondente alla scheda di acquisizione. Normalmente la connessione viene eseguita una sola volta dopo che è stato selezionato il dispositivo e non ad ogni esecuzione di un piano di collaudo. Una eventuale disconnessione reinnesca la procedura di ricerca e selezione del dispositivo da utilizzare. Tramite altre funzioni specifiche fornite dalla API di Pleora Technologies vengono quindi impostati i seguenti parametri: la risoluzione (selezionabile dall'utente tramite l'interfaccia grafica) ed il formato dell'immagine (bianco e nero con 256 tonalità di grigio). A questo punto deve essere avviata la fase di ricezione dei fotogrammi: la scheda *iPORT*^{TM} memorizza i fotogrammi acquisiti nella memoria tampone di capacità configurabile ma limitata, l'applicazione preleva uno o più fotogrammi a seconda delle necessità ed infine libera tempestivamente le locazioni di memoria tampone corrispondenti ad i nuovi fotogrammi che sono stati prelevati. In totale sono quattro gli oggetti appartenenti a classi fornite dalla API e coinvolti, tramite la libreria di incapsulamento, in questa sequenza di operazioni:

- PvDeviceFinderWnd: rappresenta la finestra ausiliaria di selezione del dispositivo;

- PvDevice: rappresenta la scheda di acquisizione e viene usata per le fasi di connessione e configurazione;
- PvStream: rappresenta il flusso di dati dalla scheda di acquisizione al calcolatore elettronico e viene usata per avviare tale flusso di dati prima che inizi l'acquisizione e per chiuderlo quando è terminata l'acquisizione;
- PvPipeline: permette il prelievo dei fotogrammi dalla memoria tampone ovvero l'acquisizione vera e propria.

1.5 Sistema software da sviluppare

Al fine di controllare un sistema quale quello raffigurato in Figura 1.7 è necessario che sia sviluppato un apposito sistema software, di tipo cosiddetto "applicativo", per il calcolatore elettronico da utilizzare nelle sessioni di collaudo. La progettazione, l'implementazione e la validazione di un tale sistema software sono stati gli obiettivi primari di questo progetto e pertanto verranno discussi dettagliatamente nei successivi due capitoli. Tuttavia, dopo che è stato descritto il contesto in cui si inserisce un tale sottosistema, è importante premettere quali sono i suoi requisiti principali.

Il sistema software deve anzitutto includere una interfaccia grafica di facile utilizzo, basata su meccanismi di controllo intuitivi. Tramite tale interfaccia grafica l'utente deve poter svolgere le seguenti funzioni principali:

1. Progettare e gestire prove (costituite da passi di prova);
2. Progettare e gestire piani di prova (costituiti da singole prove);
3. Eseguire prove secondo un determinato piano di prova;
4. Iterare l'esecuzione di determinati blocchi di prove all'interno di un piano di prova (tipo prove sotto "sforzo");

5. Generare resoconti sui risultati delle esecuzioni dei piani di prova.

Per permettere la progettazione di prove è necessario fornire all'utente, tramite l'interfaccia grafica, un apposito strumento di configurazione e composizione di uno o più singoli passi di prova di tipo predefinito. Una volta che una prova è stata composta ed i relativi passi costituenti sono stati tutti appropriatamente configurati, deve essere possibile salvare tale prova su disco rigido o altro tipo di memoria di massa. Deve altresì essere possibile in qualunque momento caricare prove precedentemente salvate su memorie di massa.

Deve poi essere possibile esplorare facilmente tramite interfaccia grafica una determinata prova e visualizzarne i dettagli, ossia visualizzare i passi di prova di cui è composta e la loro configurazione.

Una volta definite le prove elementari, l'utente deve essere in grado di poter progettare dei piani di prova ovvero delle entità costituite da sequenze di prove e/o iterazioni di sequenze di prove. Come per le prove, anche i piani di prova devono poter essere salvati su memoria di massa oppure caricati da memoria di massa e devono inoltre poter essere esplorati tramite interfaccia grafica.

L'interazione con l'utente deve essere sempre "fluida" e veloce, senza che sia introdotta alcuna latenza nelle varie operazioni che l'utente esegue tramite l'interfaccia grafica.

Una volta che sulla memoria di massa sono stati archiviati i file delle singole prove e che nella applicazione è stato creato o caricato un piano di prova, l'utente deve poter eseguire tale piano di prova, ovvero tramite appositi bottoni presenti nell'interfaccia grafica l'utente deve poter avviare e controllare l'esecuzione parallela e concorrente di un sottoprocesso che si occupi di leggere le prove costituenti il piano di prova ed attuarle. Dopo che il processo della applicazione è stato suddiviso per creare un tale sottoprocesso concorrente, sempre tramite la semplice pressione di un bottone, l'utente deve poter fermare temporaneamente o definitivamente l'esecuzione di tale sottoprocesso (senza però fermare l'esecuzio-

ne dell'applicazione ovvero del sottoprocesso "padre" che ha creato tale sottoprocesso "figlio").

Da ultimo, durante l'esecuzione del piano di prova, il programma deve poter generare un apposito documento contenente un resoconto sul risultato dell'esecuzione del piano di prova e di ciascuna prova di cui esso è costituito, riportando eventualmente la causa del fallimento per ciascuno dei passi di prova non completati con successo e riportando anche altre informazioni ausiliarie quali la data e l'ora di esecuzione del piano di prova, il nome dell'operatore e la durata totale di esecuzione.

I messaggi di errore, di avvertimento e i messaggi relativi allo stato di avanzamento delle varie operazioni devono essere visualizzati in una apposita area dell'interfaccia grafica (detta "console") e deve essere limitata il più possibile la creazione di finestre ausiliarie dedicate alla visualizzazione di tali messaggi.

Il programma deve comprendere anche un manuale di utilizzo (o guida d'utente) che sia visualizzabile direttamente dalla interfaccia grafica. Particolare cura dovrà essere riposta durante ogni fase dello sviluppo nel rispettare i seguenti vincoli:

- correttezza (ovvero rispetto dei requisiti);
- robustezza;
- usabilità;
- modularità;
- mantenibilità.

Dovrà inoltre essere rispettato il vincolo prestabilito di tempo ovvero che la durata complessiva del progetto sia di 4 mesi.

Sarà necessario un sistema di gestione delle diverse versioni del software che saranno prodotte nel corso del tempo. In particolare, dovrà essere tenuta traccia dei cambiamenti intercorsi tra versioni successive del software stesso.

Prima della creazione di ogni nuova versione dovranno essere provate in modo quanto più accurato possibile tutte le nuove funzionalità introdotte con tale versione e dovranno essere condotte quante più possibili prove di regressione che purtroppo, nel caso di una applicazione basata su interfaccia grafica, non sarà possibile automatizzare. Le prove di regressione saranno utili per assicurare che i nuovi cambiamenti non producano un decadimento o degradamento di funzionalità precedentemente sviluppate e/o favoriscano il riemergere di problemi precedentemente risolti. In particolare, sarà necessario verificare tramite suddette prove di regressione che i più recenti cambiamenti in determinate parti del programma non vadano ad influenzare negativamente il comportamento, precedentemente accertato come corretto, di altre parti del programma.

Di grande interesse durante la fase di prova del programma è la eventuale possibilità di sollecitare il programma stesso con "condizioni limite" al fine di poter controllare, prima del rilascio di una nuova versione, la qualità del nuovo codice sorgente.

Dal punto di vista delle specifiche di sistema è bene sottolineare come di frequente esse possano cambiare anche perché spesso la visione che il committente ha del sistema evolve man mano che il sistema stesso prende forma.

Anche per quest'ultimo motivo, il modello di sviluppo adottato è di tipo evolutivo. In un modello di tipo evolutivo, un prototipo iniziale evolve verso il prodotto finito attraverso un certo numero di iterazioni di sviluppo. Per evitare che il progetto possa fallire sarà necessario prestare la massima attenzione nel rispettare i seguenti vincoli:

- comprendere appieno le necessità dell'azienda (che assume in questo caso un ruolo simile a quello di un committente);

- promuovere ed assecondare la stabilizzazione da parte dell'azienda di requisiti eventualmente instabili;

- fare in modo che ci sia un buon livello di cooperazione con l'azienda;
- rigettare eventuali attese non realistiche da parte dell'azienda;
- favorire il massimo beneficio per l'azienda derivante dall'utilizzo del sistema;
- evidenziare tempestivamente l'eventuale fornitura insufficiente di risorse da parte dell'azienda.

Capitolo 2

Descrizione del progetto realizzato

2.1 Il linguaggio C# e l'ambiente .NET Framework

L'applicazione di validazione del modulo di scansione con la relativa interfaccia grafica è stata sviluppata per scelta aziendale in linguaggio C#, in ambiente Microsoft® .NET Framework versione 4.

C# è un linguaggio di programmazione orientato agli oggetti sviluppato alla Microsoft® verso la fine degli anni 1990 principalmente da Anders Hejlsberg, Scott Wiltamuth e Peter Golde e definito nelle specifiche ECMA-334 [6]. Microsoft® .NET Framework è l'implementazione adottata della *Common Language Infrastructure* (CLI) utilizzata nello sviluppo e conforme alle specifiche ECMA-335 [7]. Entrambe le specifiche sono state approvate anche dall'Organizzazione Internazionale per la Normazione (ISO) con i rispettivi documenti ISO/IEC 23270 e ISO/IEC 23271.

Il linguaggio C# unisce funzioni ben consolidate nel tempo con innovazioni di massimo livello per fornire all'ambiente di calcolo delle aziende moderne un modo efficiente e di facile utilizzo per scrivere programmi [8]. Esso interagisce con l'ambiente di esecuzione .NET Framework per creare un ambiente di programmazione altamente raffinato.

Il linguaggio C# discende direttamente da due linguaggi di pro-

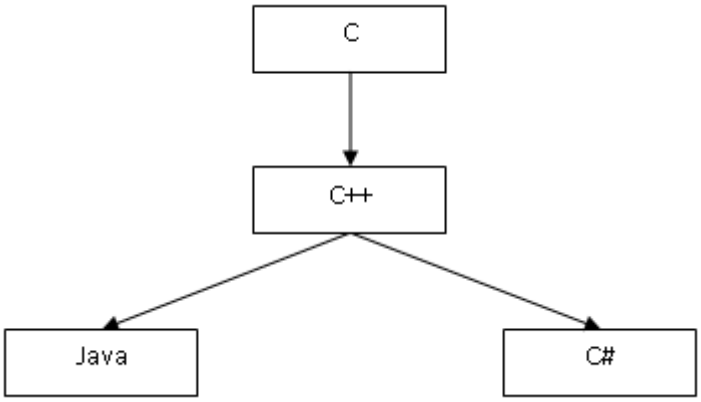

Figura 2.1: Le origini del linguaggio C#.

grammazione di maggior successo: C e C++. Inoltre esso è strettamente correlato ad un terzo linguaggio di programmazione: Java (Figura 2.1).

Tra le funzionalità più importanti introdotte da C# rispetto ai suoi predecessori vi è l'*interoperabilità* tra linguaggi diversi, ovvero la possibilità per il codice prodotto da un linguaggio di poter facilmente lavorare insieme al codice prodotto da un altro linguaggio.

Un'altra funzionalità carente ad esempio in Java ma presente in C# è la piena integrazione con il sistema operativo Windows®.

I dati sono fortemente tipizzati e questa caratteristica del linguaggio rende impossibile leggere da variabili non inizializzate, eseguire conversioni di tipo non valide o accedere ad indici inesistenti di vettori.

Essendo un linguaggio orientato agli oggetti, C# supporta i concetti di incapsulamento, ereditarietà e polimorfismo. Una classe può ereditare direttamente da un'unica classe padre, ma può implementare un numero qualsiasi di interfacce. Le strutture sono un tipo simile ad una classe "leggera", sono trattate per valore invece che per riferimento e possono implementare interfacce ma non supportano l'ereditarietà.

C# supporta anche i puntatori e il concetto di codice "non sicuro"

per i casi nei quali l'accesso diretto alla memoria è assolutamente critico.

Il processo di compilazione di C# è più semplice rispetto a quello di C e C++ ed è più flessibile rispetto a quello di Java. Non sono previsti file di intestazione separati e non è necessario che i metodi ed i tipi vengano dichiarati in un ordine specifico. In un file di origine C# è possibile definire un numero qualsiasi di classi, strutture, interfacce ed eventi.

L'ambiente .NET Framework può essere considerato come suddiviso in *Common Language Runtime* (CLR)[1] e *Framework Class Library* (FCL). Il CLR è la macchina virtuale che gestisce l'esecuzione dei programmi .NET e che fornisce servizi aggiuntivi quali la gestione della memoria, la gestione dei thread, la gestione delle eccezioni, la prevenzione degli errori nei tipi di dato e la remotizzazione. La FCL invece è una collezione di tipi riutilizzabili che si integrano strettamente con il CLR e che a loro volta permettono a nuovo codice di integrarsi. Tale libreria fornisce soluzioni a problematiche comuni quali la gestione delle stringhe, l'accesso ai file, l'internazionalizzazione, la raccolta di dati ed il collegamento a basi di dati.

Quando viene compilato un programma scritto in C#, il risultato non è codice macchina eseguibile direttamente. Infatti, quello che il compilatore genera è un file che contiene uno speciale tipo di pseudocodice chiamato Microsoft® Intermediate Language (MSIL). MSIL definisce un linguaggio di tipo assembly portabile. Il compito del CLR è di tradurre il codice intermedio in codice eseguibile quando il programma viene avviato. Pertanto, ogni programma compilato in MSIL può essere eseguito in qualunque ambiente per il quale è stato implementato il CLR [8].

Il Microsoft Intermediate Language è trasformato in codice eseguibile tramite un compilatore JIT (Just-In-Time). Infatti, quando un

[1] L'ambiente di esecuzione gestito, ovvero l'implementazione del Virtual Execution System (VES), dell'ambiente Microsoft® .NET.

programma viene avviato, il CLR attiva il compilatore JIT. Il compilatore JIT converte su richiesta l'assemblato MSIL in codice nativo mentre ciascuna parte del programma viene eseguita. Per questo motivo il programma in C# viene eseguito di fatto come codice nativo anche se inizialmente esso era stato compilato in MSIL. Ciò significa che il programma viene eseguito ad una velocità simile a quella che si otterrebbe se esso fosse stato compilato direttamente in codice nativo, ma allo stesso tempo esso beneficia della portabilità offerta da MSIL. Inoltre, durante la compilazione, ha luogo la verifica del codice per assicurare la corretta definizione dei tipi di dati ed evitare conseguenti accessi inappropriati alle locazioni di memoria (*type-safety*) [8].

Durante la compilazione, oltre al MSIL vengono generati anche i metadati che sono informazioni binarie le quali descrivono un programma e vengono memorizzate in un file eseguibile portabile (PE, Portable Executable). Ogni tipo e membro definito a cui si fa riferimento è descritto nei metadati. Quando il codice viene eseguito, il CLR carica i metadati in memoria e vi fa riferimento per ottenere informazioni sulle classi utilizzate nel codice, sui membri e sull'ereditarietà, permettendo l'interazione con altro codice, anche se prodotto da linguaggi diversi (come nel caso illustrato nel paragrafo 2.5).

2.2 Descrizione del sistema in Unified Modeling Language

In questo paragrafo verrà fornita una descrizione del sistema software mediante un linguaggio di modellazione visuale noto come Unified Modeling Language (UML).

Un linguaggio di modellazione risolve un problema di comunicazione tra progettisti e tra progettisti e committente dovuto al fatto che il linguaggio naturale è troppo impreciso mentre il codice sorgente pur essendo preciso è troppo dettagliato.

Area	Diagramma
Classificazione strutturale	Diagramma di classe
	Diagramma dei pacchetti
	Diagramma degli oggetti
	Diagramma dei componenti
	Diagramma di dispiegamento
	Diagramma delle strutture composite
Comportamento dinamico	Diagramma dei casi d'uso
	Diagramma di sequenza
	Diagramma di comunicazione
	Diagramma della macchina a stati
	Diagramma di attività
	Diagramma di interazione generale
	Diagramma di temporizzazione

Tabella 2.1: Tipologie di diagrammi definiti da UML versione 2 suddivisi per aree.

Un linguaggio di modellazione deve avere le seguenti caratteristiche:

- essere sufficientemente preciso;
- essere flessibile dal punto di vista descrittivo in modo da poter arrivare a qualunque livello di dettaglio;
- costituire un modello di riferimento (standard).

Tramite UML è possibile visualizzare informazioni riguardo alla struttura statica ed al comportamento dinamico di un sistema il quale viene modellato come un insieme di oggetti discreti che interagiscono per eseguire un lavoro di cui alla fine beneficia un utente esterno al sistema [9].

La versione 2 di UML definisce 13 diagrammi (vedi Tabella 2.1) ed essi sono suddivisi in due aree principali secondo il tipo di descrizione che forniscono.

Il primo gruppo di diagrammi fornisce una classificazione strutturale ovvero descrive le entità presenti nel sistema e le loro relazioni con altre entità. Il secondo gruppo fornisce invece una descrizione

del comportamento dinamico del sistema o delle entità presenti in esso [9, 10].

Il primo diagramma riportato in questo lavoro (Figura 2.2) è un diagramma dei casi d'uso. Esso serve a carpire il comportamento del sistema software per come apparirebbe ad un utente dall'esterno.

Un diagramma dei casi d'uso partiziona la funzionalità del sistema in transazioni significative per gli attori ovvero i soggetti, esterni al sistema, che interagiscono con esso. Il termine *attore* include esseri umani, calcolatori elettronici o processi in esecuzione su questi ultimi.

Un *caso d'uso* costituisce, pertanto, una descrizione logica di una unità di funzionalità, è disegnato con una ellisse, può essere associato con gli attori (linee rosse continue) e può partecipare a varie altre relazioni tra cui l'estensione (linea blu tratteggiata con freccia aperta). Un caso d'uso può anche essere specializzato in uno o più casi d'uso "figli" (generalizzazione). In questo modo, ogni caso d'uso "figlio" può venire utilizzato in qualunque situazione in cui può venire utilizzato il caso d'uso "genitore". La generalizzazione di un caso d'uso è disegnata con una linea continua dal "figlio" al "genitore" ed una grande freccia triangolare nella estremità corrispondente al "genitore". Quando un caso d'uso viene implementato esso è realizzato tramite collaborazioni tra classi del sistema.

Lo scopo principale di un diagramma dei casi d'uso è quello di aiutare gli sviluppatori a visualizzare i requisiti funzionali di un sistema.

Il diagramma dei casi d'uso creato per descrivere il sistema software *Scan Engine Test Program* è riportato in Figura 2.2 e mostra le funzioni di alto livello di tale sistema ed il suo scopo.

Sulla sinistra del diagramma sono visibili i due attori primari: il progettista ed il collaudatore. Tali due attori primari possono corrispondere ad utenti diversi o a due ruoli diversi svolti da uno stesso utente.

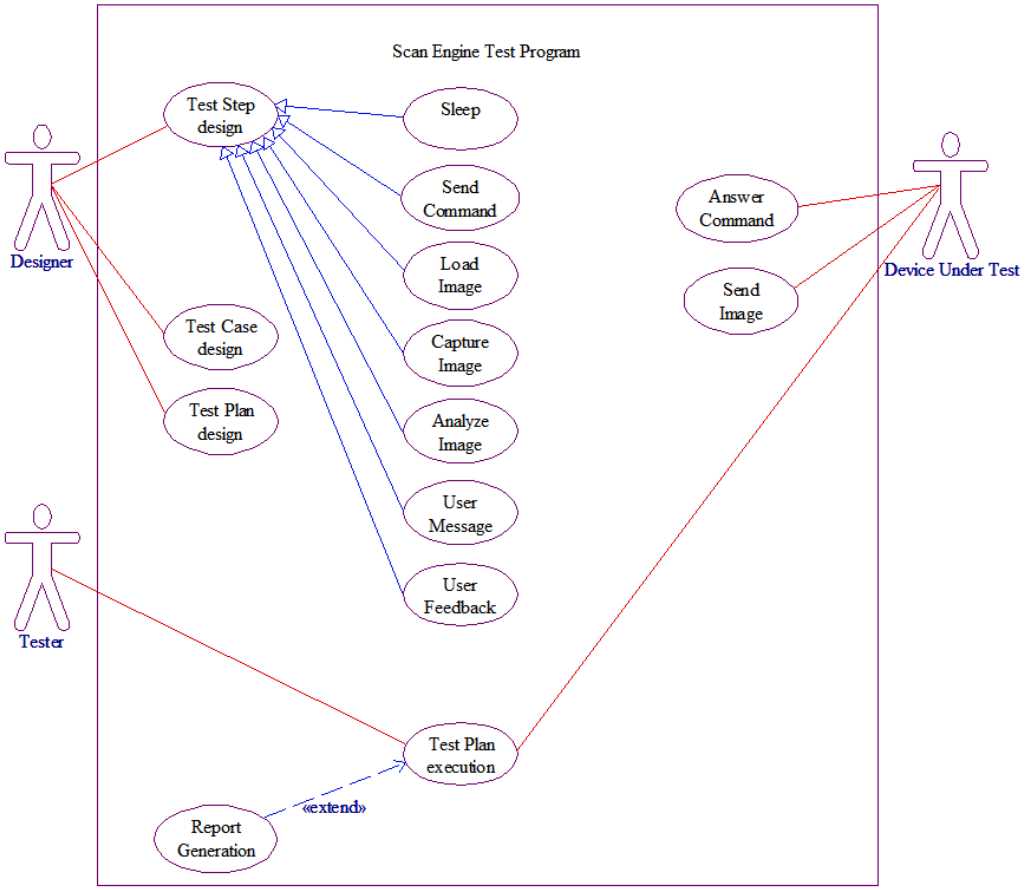

Figura 2.2: Diagramma dei casi d'uso per il sistema software *Scan Engine Test Program*.

Come in parte già accennato nel paragrafo 1.5, il sistema permette al progettista (a sinistra in alto) di definire tre elementi principali:

- i passi di prova (*test steps*);
- le prove (*test cases*);
- i piani di prova (*test plans*).

I passi di prova sono l'elemento atomico della rappresentazione dei dati di collaudo e possono essere di 7 tipi diversi:

- *Sleep* (pausa);

- *Send Command* (invio di un comando al modulo di scansione sul bus I^2C);
- *Load Image* (caricamento di una immagine da memoria di massa);
- *Capture Image* (cattura di una o più immagini consecutive dal modulo di scansione tramite la scheda di acquisizione);
- *Analyze Image* (analisi di una, due o tre regioni in una o due immagini);
- *User Message* (visualizzazione di un messaggio testuale e/o grafico all'utente);
- *User Feedback* (presentazione di una domanda all'utente con risposta del tipo "SI" o "NO").

Il secondo attore primario, cioè il collaudatore (a sinistra in basso), può interagire con il sistema per eseguire piani di prova ed eventualmente generare resoconti riguardanti l'esecuzione.

Nell'esecuzione di un piano di prova è in genere coinvolto anche il dispositivo da collaudare (modulo di scansione) il quale può rispondere ad un comando inviato dall'applicazione (passo di prova *Send Command*) e può inviare una o più immagini su richiesta dell'applicazione (passo di prova *Capture Image*).

Il secondo diagramma riportato in questo lavoro è un diagramma dei componenti (Figura 2.3). Esso mostra le varie unità software a partire dalle quali è costruita l'applicazione e le dipendenze tra tali componenti. In alto è visibile il componente costituito dalla applicazione *Scan Engine Test Program* (SETP). Essa dipende in modo diretto da due librerie di incapsulamento (nella fascia intermedia): una relativa alla interfaccia di controllo su bus I^2C (mostrata a sinistra nel diagramma e citata nel paragrafo 1.3) ed una relativa alla interfaccia di acquisizione immagini (mostrata a destra nel diagramma ed introdotta nel paragrafo 1.4). Entrambe le librerie di incapsu-

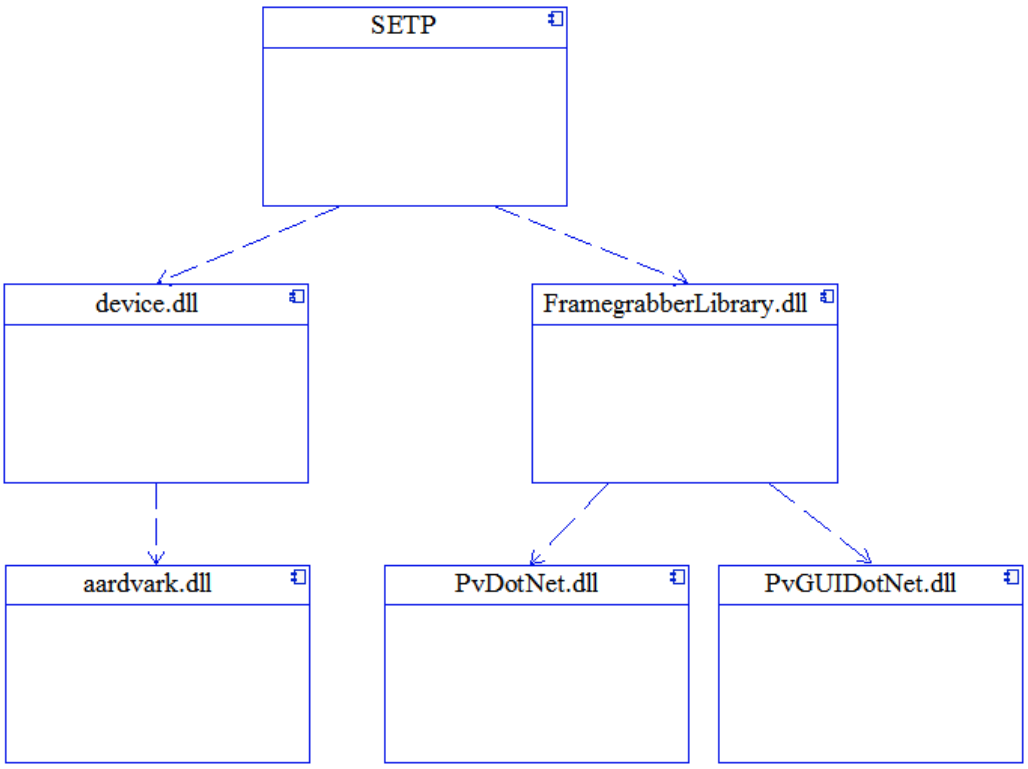

Figura 2.3: Diagramma dei componenti del sistema software *Scan Engine Test Program*.

lamento sono state sviluppate insieme alla applicazione nel corso di questo progetto di tesi.

I componenti alle estremità inferiori rappresentano specifici prodotti commerciali i quali, grazie al progetto modulare del sistema ed all'utilizzo delle due librerie di incapsulamento, possono essere con facilità sostituiti all'occorrenza con altri prodotti equivalenti che offrano la stessa funzionalità ed una interfaccia simile. Le librerie di incapsulamento inoltre, possono essere riutilizzate in altri sistemi software in quanto generiche.

Per implementare l'elemento atomico costituito dal passo di prova, è stata creata una classe base (o genitrice) da cui discendono classi specializzate nei 7 tipi di passi di prova precedentemente elencati. Ciò è raffigurato nel diagramma di classe riportato in Figura 2.4.

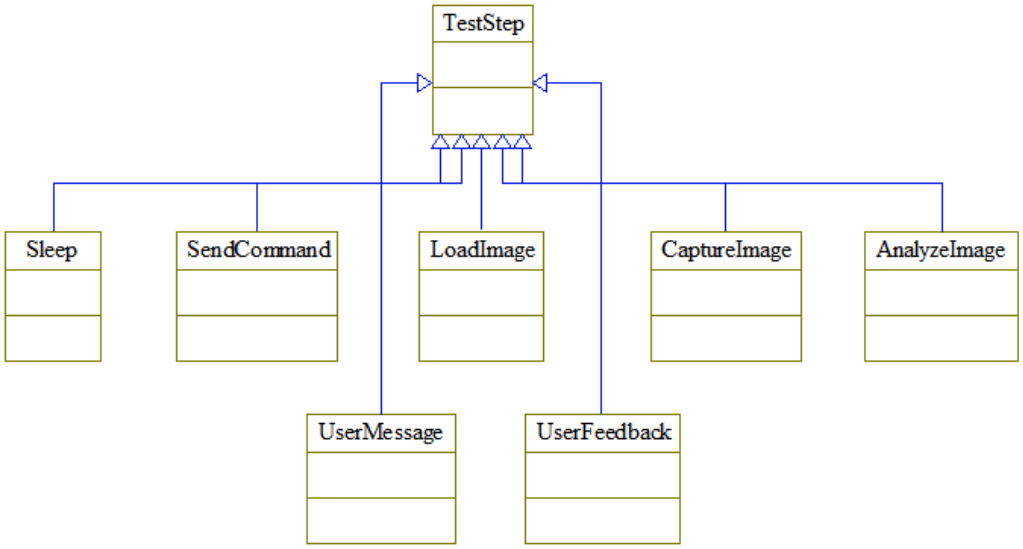

Figura 2.4: Diagramma della classe TestStep.

La rappresentazione dei dati di collaudo è basata oltre che sulla classe TestStep anche sulle due ulteriori classi TestCase e TestPlan. Le tre classi TestStep, TestCase e TestPlan sono utilizzate per contenere i dati relativi alle sessioni di collaudo. Le relazioni che intercorrono tra queste tre classi sono rappresentate dal diagramma riportato in Figura 2.5.

Sono visibili tre diversi tipi di relazioni tra le classi rappresentate: generalizzazione, aggregazione e composizione.

La prima relazione (generalizzazione) lega la classe ArrayList alle due classi TestCase e TestPlan ovvero queste ultime sono classi derivate dalla prima. La classe Arraylist è fornita dalla Framework Class Library e fornisce vettori di dimensione dinamica di oggetti. Un oggetto ArrayList viene creato con una dimensione iniziale e quando questa dimensione non è più sufficiente viene automaticamente ingrandito. Si è scelto questo tipo di classe base poichè sia le prove che i piani di prova sono liste, rispettivamente di passi di prova e prove.

La seconda relazione (aggregazione) lega la classe TestPlan alla

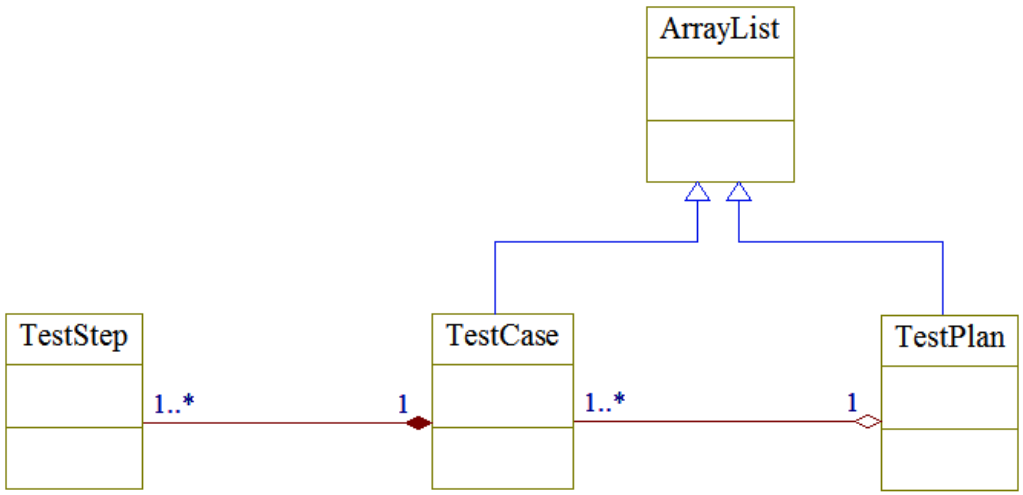

Figura 2.5: Diagramma delle classi usate per la rappresentazione dei dati di collaudo.

classe TestCase e più specificatamente indica che un oggetto di tipo TestPlan è composto da uno o più oggetti di tipo TestCase. Questo tipo di relazione è stato scelto invece della relazione di composizione poichè oggetti di tipo TestCase possono esistere anche in assenza di un oggetto di tipo TestPlan che li contenga.

La terza relazione (composizione), infine, lega la classe TestCase alla classe TestStep, specificando che un oggetto di tipo TestCase è composto da uno o più oggetti di tipo TestStep e che oggetti di tipo TestStep non possono esistere separatamente da oggetti di tipo TestCase che li contengano.

Il quinto diagramma (riportato in Figura 2.6) visualizza la struttura statica dell'intero sistema software *Scan Engine Test Program*. La finestra principale dell'interfaccia grafica con l'utente è rappresentata da un'istanza della classe MainForm derivata dalla classe base Form messa a disposizione dalla Framework Class Library proprio allo scopo di permettere la progettazione di finestre utilizzate come interfaccia con l'utente.

Come prima cosa l'oggetto MainForm crea un'istanza degli oggetti TestCase (la prova visualizzata) e TestPlan (il piano di prova

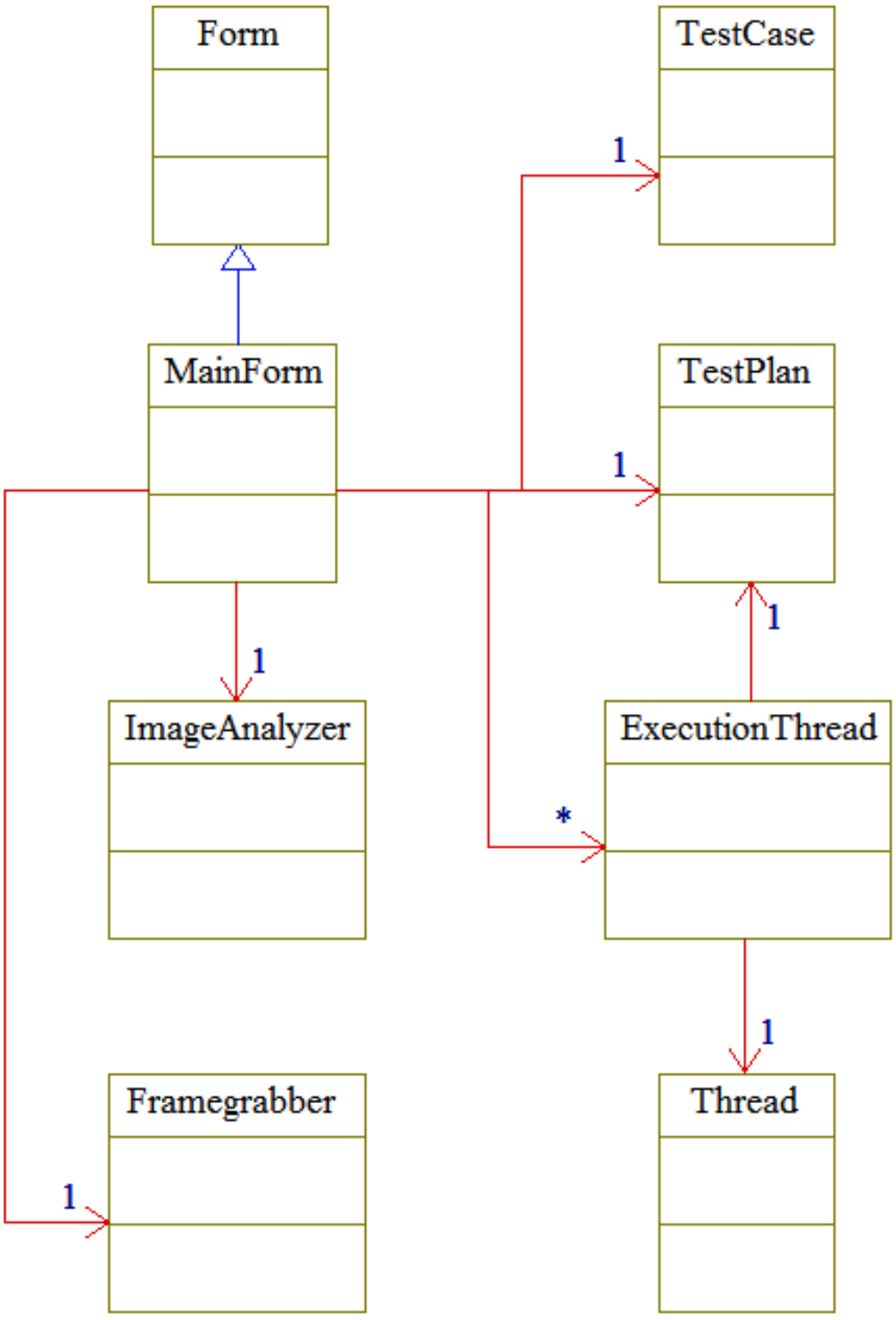

Figura 2.6: Struttura statica del sistema software *Scan Engine Test Program*.

visualizzato): essi servono a memorizzare i dati di collaudo sui quali l'utente sta lavorando. L'oggetto MainForm crea anche un'istanza di ciascuna delle classi Framegrabber ed ImageAnalyzer: la prima è utilizzata per catturare immagini dal modulo di scansione tramite l'interfaccia di acquisizione immagini (descritta nel paragrafo 1.4) mentre la seconda è utilizzata per eseguire l'analisi delle immagini secondo uno di vari algoritmi disponibili. Infine l'oggetto MainForm può anche eventualmente creare uno o più oggetti della classe ExecutionThread corrispondenti alla esecuzione di un piano di prova ed i quali a loro volta generano un oggetto della classe Thread. Ricordando che, in un diagramma di classe, la molteplicità di una associazione indica quante istanze di una classe possono essere correlate ad un'istanza dell'altra classe, si può notare come vi possono in generale essere infinite istanze della classe ExecutionThread poichè ad una singola esecuzione della applicazione possono corrispondere più esecuzioni di piani di prova.

2.3 Sviluppo dell'interfaccia grafica con l'utente

Un'interfaccia grafica con l'utente è costituita di finestre reattive agli eventi generati dall'utente. Essa è incentrata sull'esecuzione di determinati metodi (chiamati gestori d'evento) in risposta alle azioni dell'utente. La computazione avviene solo quando gli eventi innescano l'esecuzione dei gestori d'evento. In molti casi i gestori d'evento possono, al loro interno, innescarne altri producendo così una cascata di eventi. Normalmente non si dovrebbe gravare di un carico computazionale eccessivo i gestori d'evento altrimenti l'interazione dell'interfaccia grafica con l'utente non risulterebbe più fluida.

Gli eventi che determinano il flusso del programma *Scan Engine Test Program* sono dei seguenti tipi:

- selezione di menù o di menù a tendina;
- pressione di pulsanti presenti nell'interfaccia grafica;

- selezione di un elemento in una lista tramite pressione del pulsante sinistro del dispositivo di puntamento o tramite tasti di spostamento;
- attivazione di menù contestuali tramite pressione del pulsante destro del dispositivo di puntamento;
- inserimento di dati in una casella di testo;
- cambiamento di un valore numerico tramite barra a scorrimento;
- pressione del pulsante sinistro del dispositivo di puntamento su una etichetta;
- selezione di un elemento in una casella di selezione;
- selezione o deselezione di una casella di spunta;
- cambiamento dell'immagine di sfondo nel riquadro di visualizzazione;
- ridisegno del riquadro di visualizzazione immagini;
- movimento e pressione del pulsante sinistro del dispositivo di puntamento (per la selezione delle aree dell'immagine);
- ridimensionamento dell'area utilizzata da un controllo grafico;
- chiusura della finestra principale o selezione dell'opzione *Exit* dal menù principale o dall'icona dell'applicazione nell'area di notifica;
- cambiamento di un oggetto (innescato quando una nuova immagine viene acquisita oppure quando l'esito di un passo di prova o di una prova diventa disponibile a seguito della sua esecuzione o viene cancellato all'inizio di una nuova esecuzione).

L'interfaccia grafica sviluppata in questo progetto genera due tipi di finestre: la finestra principale che viene visualizzata all'avvio della applicazione e le finestre ausiliarie che possono venire visualizzate successivamente. Queste ultime possono essere dei seguenti tipi:

- finestre di selezione di documenti su memoria di massa dai quali caricare o su cui salvare i dati di collaudo;
- finestre contenenti messaggi di errore (causati ad esempio dall'inserimento da parte dell'utente di dati non corretti come parametri di configurazione);
- finestre visualizzate durante l'esecuzione di un piano di prova per inviare messaggi all'utente (passi di prova *User Message*);
- finestre visualizzate durante l'esecuzione di un piano di prova per richiedere all'utente risposta a determinate domande (passi di prova *User Feedback*).

L'applicazione *Scan Engine Test Program* sviluppata in questo progetto di tesi è una applicazione a thread multipli o più semplicemente *multithread*. Un thread di esecuzione definisce un percorso separato di esecuzione, pertanto un programma multithread contiene due o più parti (thread) che vengono eseguite concorrentemente.

Affinchè la finestra principale MainForm venga utilizzata come classe iniziale nell'applicazione deve essere creata e visualizzata da un metodo denominato Main() e marcato con l'attributo STAThread[2]. La chiusura di tale finestra iniziale determina la chiusura dell'applicazione. Ne consegue che il thread principale dell'applicazione, e solo esso, si deve occupare di creare, inizializzare e gestire

[2]L'attributo *STAThread* serve ad indicare che il modello di threading COM per l'applicazione è apartment a thread singolo, in quanto il modello apartment multithreading non è supportato da Windows Form. Il modello STA è utilizzato per oggetti che non gestiscono la loro stessa sincronizzazione (oggetti che non sono *thread-safe*).

l'interfaccia grafica con l'utente, occupandosi anche di rilevare e gestire i relativi eventi.

La progettazione di questa parte di applicazione è basata sull'utilizzo dei controlli grafici messi a disposizione dalla Framework Class Library ed un elenco esaustivo di tutti i controlli utilizzati è fornito in Appendice A.

Poichè gli attori primari del sistema sono due, il progettista di prove ed il collaudatore (Figura 2.2), è stato deciso che la finestra principale dell'interfaccia grafica fosse dotata di due modalità di funzionamento distinte: una modalità progettuale (anche detta "vista di progetto") ed una modalità di esecuzione del collaudo (indicata anche con il nome di "vista di collaudo"). Il passaggio tra le due modalità d'uso è stato implementato tramite l'utilizzo di un controllo grafico del tipo ToolStripComboBox (menù a tendina) come elemento della barra principale dei menù (controllo grafico MenuStrip). Un evento è generato dal controllo ToolStripComboBox ogni qualvolta l'utente cambia la modalità selezionata; a questo punto un metodo prestabilito riceve e gestisce tale evento eseguendo opportune operazioni per passare da una modalità all'altra.

Entrambe le due modalità di funzionamento della finestra principale prevedono che essa sia suddivisa verticalmente in due pannelli destinati allo svolgimento di azioni diverse. Nel caso della modalità di progettazione, il pannello a sinistra è destinato alla progettazione delle prove ed il pannello a destra è destinato alla progettazione dei piani di prova ed alla visualizzazione di messaggi di servizio per l'utente (funzionalità di "console"). Nel caso della modalità di esecuzione del collaudo, invece, il pannello a sinistra è utilizzato per le attività di visualizzazione del piano di prova, delle prove e dei relativi esiti ed il pannello a destra è utilizzato per il controllo dell'esecuzione del collaudo, per la visualizzazione delle immagini da analizzare, per la selezione delle aree dell'immagine su cui effettuare l'analisi e per la visualizzazione di messaggi di servizio per l'utente.

Per realizzare tale suddivisione verticale della finestra principale in due pannelli diversi, a seconda delle due modalità, si è scelta la seguente implementazione: lo spazio disponibile nell'intera finestra escludendo la barra dei menù è stato suddiviso in tre tramite due controlli grafici di tipo Splitter ed in ciascuno dei tre spazi ricavati è stato inserito un diverso controllo di tipo Panel (dal nome rispettivamente di *panelDesign*, *panelPlan* e *panelExecute*). I due Splitter (da sinistra verso destra) sono stati configurati con ancoraggio a sinistra e a destra ed i tre pannelli sono stati configurati rispettivamente con ancoraggio a sinistra, con ancoraggio a riempimento dello spazio disponibile e con ancoraggio a destra. Al passaggio tra le due modalità uno dei tre pannelli (*panelDesign* oppure *panelExecute*) viene reso invisibile ed il pannello *panelPlan* viene riconfigurato abilitando nella parte bassa la visualizzazione dei passi di prova oppure dei messaggi di servizio.

L'aspetto tipico della finestra principale dell'applicazione nelle due modalità di utilizzo è mostrato nella Figura 2.7 e nella Figura 2.8.

Come risulta evidente dalle figure, oltre al già citato controllo ToolStripComboBox per cambiare modalità d'utilizzo, la barra principale dei menù ospita anche il menù *Config* (per la configurazione dei parametri del dispositivo di interfacciamento I^2C, quali indirizzo e velocità di comunicazione), il menù *Help* (per la visualizzazione di informazioni sulla versione in uso dell'applicazione e per la consultazione della guida d'utente) ed infine il pulsante *Exit* per terminare l'applicazione.

Per la visualizzazione dei dati di collaudo, cioè passi di prova, prove e piani di prova, l'interfaccia grafica sviluppata in questo progetto utilizza istanze di un controllo grafico denominato ListView. La prima istanza di ListView è utilizzata nella modalità progettuale (Figura 2.7) in *panelDesign* per visualizzare i passi di prova di cui è composta la prova che l'utente sta progettando o la prova selezionata nel piano di prova visualizzato a destra nel *panelPlan*. Una secon-

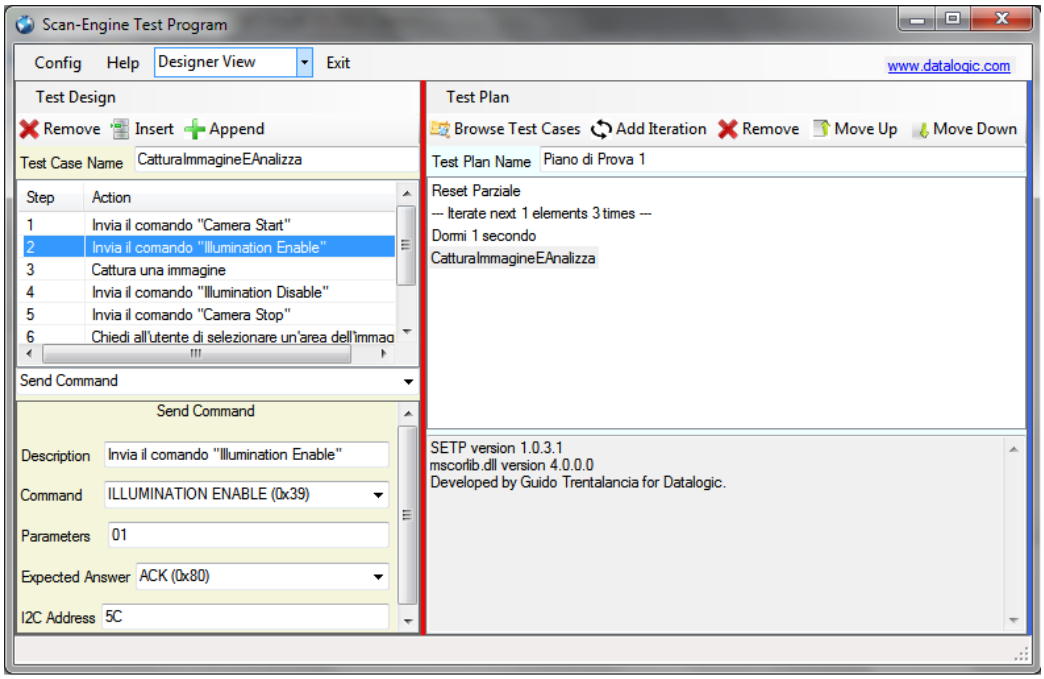

Figura 2.7: Tipico aspetto dell'applicazione in modalità di progettazione delle prove.

da istanza di ListView è utilizzata sia in modalità progettuale che in modalità di esecuzione del collaudo per visualizzare nella parte superiore di *panelPlan* le prove che costituiscono il piano di prova che è stato caricato da memoria di massa o che è in fase di progetto. Una terza ed ultima istanza di ListView viene utilizzata nella sola modalità di esecuzione del collaudo (Figura 2.8), nella parte inferiore del controllo *panelPlan*, con lo scopo di permettere la visualizzazione dei passi di prova che costituiscono la prova eventualmente selezionata nella parte superiore del pannello.

Per permettere l'ulteriore suddivisione dei tre pannelli principali *panelDesign*, *panelPlan* e *panelExecute* sono stati utilizzati controlli del tipo SplitContainer, mediante i quali per ciascun pannello sono stati creati due sottopannelli.

Ciascuno dei tre pannelli principali è dotato di una propria barra dei menù (controllo MenuStrip), di un proprio menù (control-

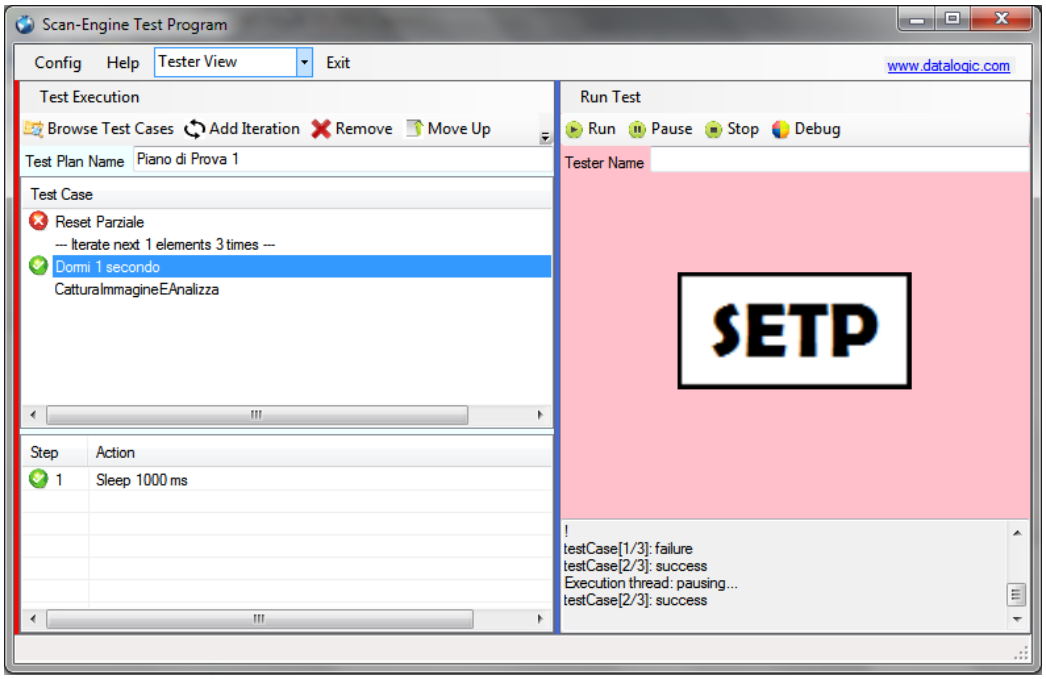

Figura 2.8: Tipico aspetto dell'applicazione in modalità di esecuzione del collaudo.

lo ToolStripMenuItem), di una barra dedicata a pulsanti (controllo ToolStrip), di pulsanti (controllo ToolStripButton) e di un menù di contesto attivabile con la pressione del tasto destro del dispositivo di puntamento. In ciascuno di tali pannelli, infine, è presente sotto la barra dei menù e sotto la barra dei pulsanti una casella di testo (controllo TextBox) dedicata all'assegnazione di una descrizione globale della prova (*panelDesign*), all'assegnazione di una descrizione globale del piano di prova (*panelPlan*) ed alla configurazione del nome del collaudatore (*panelExecute*).

Il menù del *panelDesign* è stato concepito per offrire le seguenti opzioni: creazione di una nuova prova (*New*), caricamento di una prova da memoria di massa (*Load*) e salvataggio di una prova su memoria di massa (*Save* e *Save as*). Lo stesso pannello ospita anche tre pulsanti: per la rimozione del passo di prova selezionato (*Remove*), per l'inserimento di un nuovo passo nella posizione selezionata

della prova (*Insert*) e per l'aggiunta di un nuovo passo alla fine della prova (*Append*). Il menù di contesto disponibile in *panelDesign* offre le stesse opzioni offerte da tali tre pulsanti.

Per consentire la configurazione dei passi di cui è composta una determinata prova, viene utilizzato in modalità progettuale, il sottopannello inferiore di *panelDesign* all'interno del quale sono stati inseriti, per ciascuna delle 7 tipologie di passi di prova, vari controlli grafici per la scelta delle opzioni di configurazione: Label (etichetta), TextBox (casella di testo), ComboBox (menù a tendina), TrackBar (barra a scorrimento), Button (bottone), NumericUpDown (casella numerica di selezione) e RadioButton (casella di selezione mutualmente esclusiva). Per ciascuna tipologia di passo di prova è stato creato un diverso pannello di configurazione con parte dei controlli appena citati ed in fase di utilizzo dell'applicazione, quando viene scelta una determinata tipologia, viene reso visibile il solo pannello corrispondente.

Il diagramma UML di attività riportato in Figura 2.9 mostra il ciclo di azioni necessarie per la progettazione di una prova. Le varie attività sono divise in due regioni distinte e separate da linee, chiamate *partizioni*. Questa suddivisione serve a separare le attività gestite da entità diverse (nel caso specifico l'attore progettista e l'oggetto TestCase utilizzato per la rappresentazione dei dati relativi ad una prova).

Il menù del *panelPlan* è stato progettato per offrire le seguenti opzioni: creazione di un nuovo piano di prova (*New*), caricamento di un piano di prova da memoria di massa (*Load*), aggiornamento dell'attuale piano di prova (*Reload*) e salvataggio di un piano di prova su memoria di massa (*Save* e *Save as*). In tale pannello sono anche presenti cinque pulsanti che offrono le seguenti funzioni: caricamento da memoria di massa di una prova nella posizione selezionata del piano di prova (*Browse Test Cases*), inserimento di una direttiva di iterazione nella posizione selezionata del piano di prova (*Add Iteration*), rimozione della prova selezionata (*Remove*), sposta-

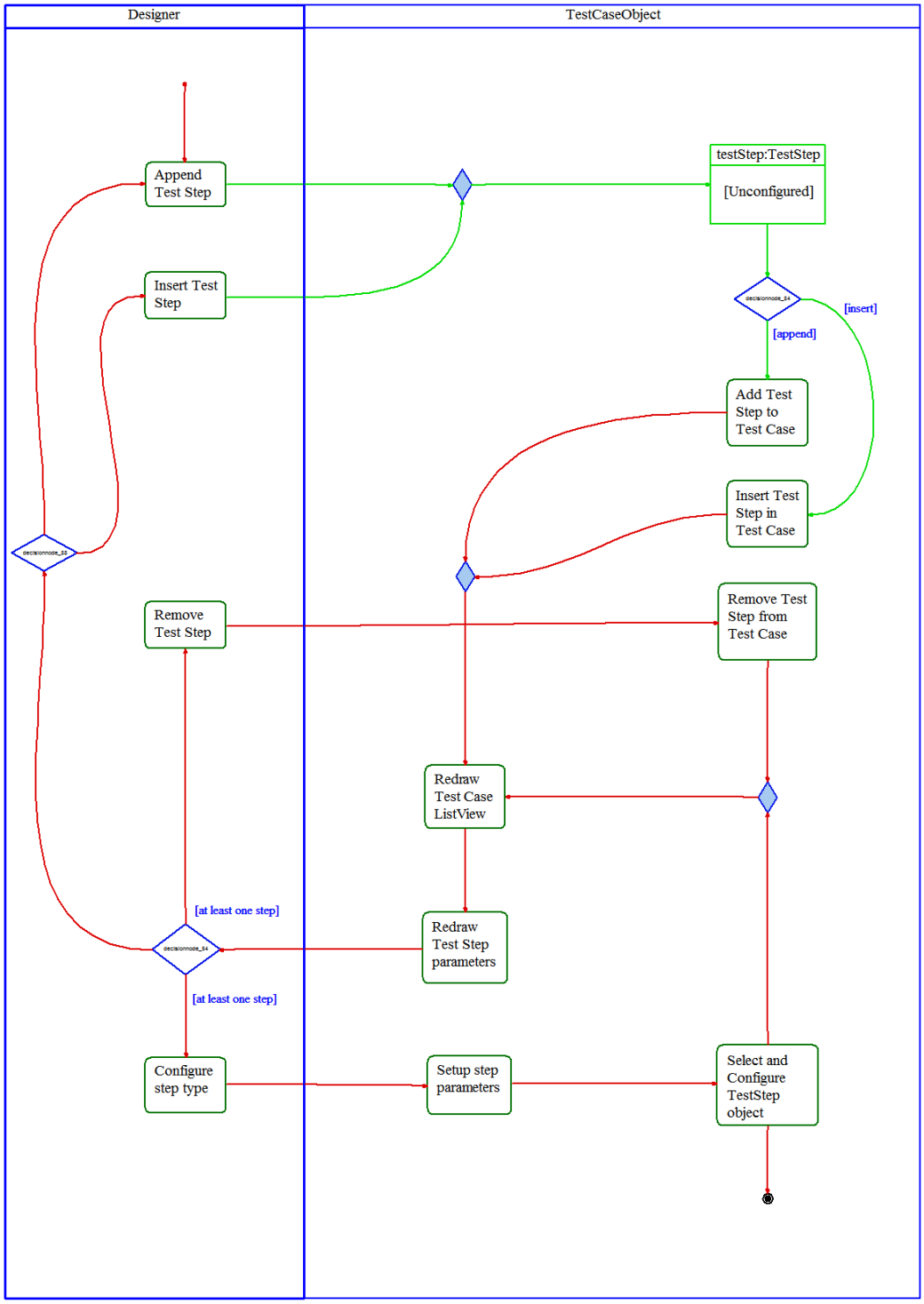

Figura 2.9: Ciclo di azioni necessarie per la progettazione di una prova.

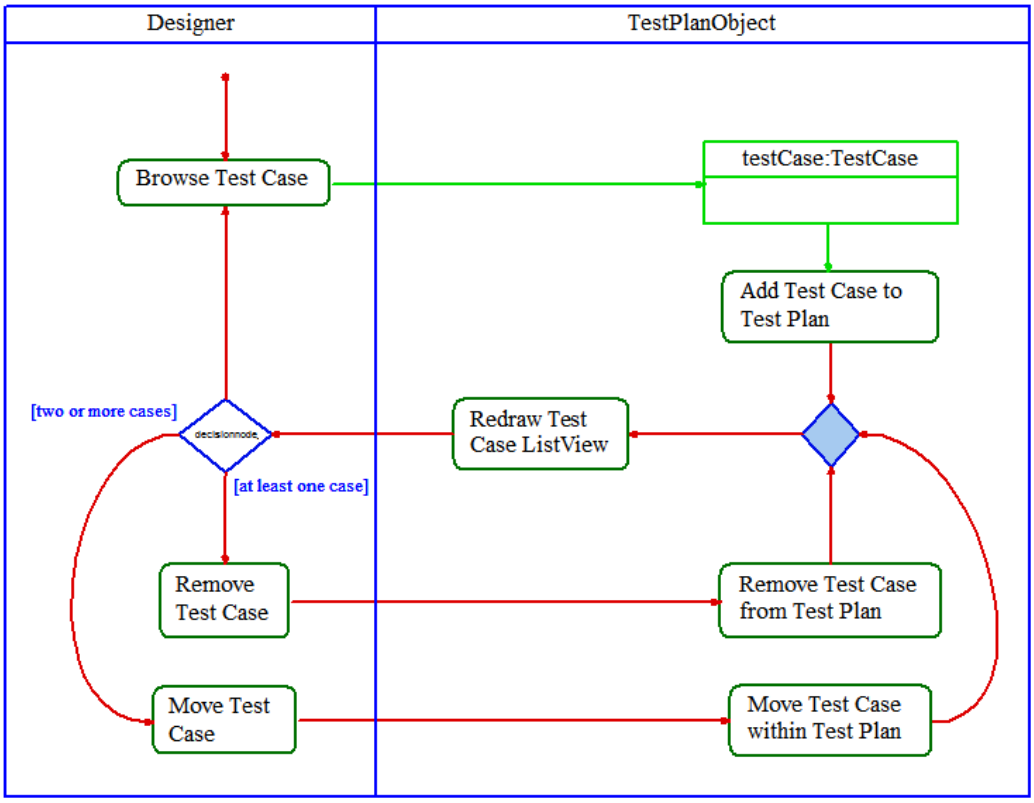

Figura 2.10: Ciclo di azioni necessarie per la progettazione di un piano di prova.

mento della prova selezionata più in alto o più in basso nel piano di prova (*Move Up* e *Move Down*). Il menù di contesto offre in questo caso le seguenti funzioni: rimozione della prova selezionata (*Remove*), spostamento della prova selezionata più in alto o più in basso (*Move Up* e *Move Down*) e spostamento della prova selezionata in cima o in fondo (*Move to Top* e *Move to Bottom*).

Il diagramma UML di attività riportato in Figura 2.10 mostra il ciclo di azioni necessario per la progettazione di un piano di prova. Anche in questo caso il diagramma di attività è diviso in due partizioni corrispondenti all'attore progettista (parte sinistra) ed all'oggetto TestPlan utilizzato per la rappresentazione dei dati relativi ad un piano di prova (parte destra).

Il menù del *panelExecute* offre le seguenti opzioni: esecuzione di

un piano di prova in modalità passo-passo (*Debug Test Plan*), esecuzione di un piano di prova in modalità normale (*Run Test Plan*), arresto dell'esecuzione (*Stop Execution*), abilitazione e disabilitazione della generazione del resoconto di esecuzione (*Generate Report*) e configurazione del comportamento in caso di fallimento di un passo di prova (*On Fail ...*). Sono presenti in questo caso i seguenti quattro pulsanti: per avviare l'esecuzione (*Run*), per mettere in pausa l'esecuzione (*Pause*), per fermare l'esecuzione (*Stop*) e per eseguire il piano di prova in modalità passo-passo (*Debug*). L'unica funzione offerta dal menù di contesto relativo a questo pannello è quella di rimuovere un'area selezionata nell'immagine caricata da memoria di massa (passo di prova *Load Image*) oppure acquisita dal modulo di scansione (passo di prova *Capture Image*).

Come già accennato, infine, è stata implementata per entrambe le modalità di utilizzo dell'interfaccia grafica un'area dedicata alla visualizzazione di messaggi di servizio per l'utente. Tale area si trova in basso a destra (in *panelPlan* nella modalità progettuale e in *panelExecute* nella modalità di esecuzione del collaudo) ed è stata realizzata tramite un controllo grafico di tipo TextBox (casella di testo) configurato in modalità multilinea.

Per rendere più facile l'utilizzo sono state programmate in molti controlli grafici indicazioni d'uso per l'utente le quali vengono visualizzate al passaggio del dispositivo di puntamento sul controllo.

Data la particolare importanza del ruolo rivestito dagli eventi nel funzionamento dell'interfaccia grafica, viene fornito un loro elenco esaustivo nell'Appendice B.

2.4 Sviluppo di un modello dei dati

Un modello dei dati è necessario per lo sviluppo del sistema software in quanto serve a fornire la definizione ed il formato dei dati; esso può servire anche a permettere lo scambio dei dati tra appli-

cazioni diverse. Il suo scopo è quello di catturare e descrivere un sottoinsieme delle informazioni reali significativo per l'applicazione.

Il modello dei dati per questa applicazione consiste di due gruppi di strutture dati: un primo gruppo necessario per permettere un'agile presentazione e manipolazione dei dati ed un secondo gruppo necessario per ottenere la persistenza di tali dati ovvero per garantire la loro durevolezza nel tempo tramite l'ausilio di memorie di massa.

Mediante il primo gruppo di strutture dati l'utente può, tramite l'interfaccia grafica, accedere visivamente ai dati di collaudo, apportarvi modifiche ed utilizzarli per controllare l'esecuzione del collaudo. Mediante il secondo gruppo di strutture dati, invece, l'utente può leggere i dati dalla memoria di massa e trasferirli nella memoria centrale per l'elaborazione oppure scrivere sulla memoria di massa i dati presenti nella memoria centrale per poi recuperarli in un'altra sessione.

Il primo gruppo di strutture dati si ricava dall'analisi del dominio della applicazione ed individua:

- Classi
- Attributi (ovvero informazioni contenute negli oggetti della classe);
- Operazioni (ovvero servizi offerti dagli oggetti della classe);
- Relazioni tra le classi (ad esempio ereditarietà, associazione, dipendenza, etc.).

Nel caso in esame il dominio dell'applicazione è il collaudo di un prodotto industriale e pertanto gli elementi fondamentali sono le prove da effettuare su tale prodotto. Un collaudo completo è definito come l'esecuzione di un insieme di prove sul prodotto e può essere descritto tramite un piano di collaudo o piano di prova il quale costituisce pertanto il secondo elemento fondamentale del dominio. Un

terzo ed ultimo elemento fondamentale si trova osservando che le prove sono in realtà costituite da uno o più passi i quali, nel caso in oggetto, sono di uno dei 7 tipi già descritti nel paragrafo 2.2 e raffigurati nel diagramma della Figura 2.4 dal quale risulta evidente che la classe TestStep viene utilizzata come classe base da cui far derivare le 7 classi corrispondenti alle diverse tipologie di passi di prova. Gli elementi fondamentali così trovati definiscono le tre classi del modello dei dati che sono state già riportate in Figura 2.5: TestStep (passo di prova), TestCase (prova) e TestPlan (piano di prova).

Per ottenere la persistenza di tali dati si ricorre ad una conversione delle strutture dati non atomiche del primo gruppo in un'altra struttura dati denominata DataSet e progettata come astrazione di una base di dati relazionale. Una prima istanza di una struttura del tipo DataSet è utilizzata per contenere i dati relativi ad una determinata prova ed una seconda istanza di struttura del tipo DataSet è utilizzata per contenere i dati relativi ad un determinato piano di prova. La Figura 2.11 mostra il modello dei dati fin qui descritto.

La classe DataSet è fornita dal componente ADO.NET della Framework Class Library: le classi ADO.NET sono contenute nella libreria *System.Data.dll* e sono integrate con le classi XML (sigla di *eXtensible Markup Language*) contenute nella libreria *System.Xml.dll*. ADO.NET fornisce difatti una stretta integrazione con XML: da un DataSet può venire scritto un documento XML su memoria di massa e da un documento XML residente su memoria di massa può venire caricato un DataSet. Per riempire un DataSet con i dati provenienti da una sorgente XML viene utilizzato il metodo ReadXml, mentre per creare documenti XML si invoca il metodo WriteXml da un oggetto Dataset. Il Dataset è basato su una struttura relazionale dei dati, mentre il documento XML è basato su una struttura gerarchica degli stessi. Un oggetto DataSet rappresenta una *cache* in memoria dei dati recuperati da un'origine dati ed è costituito da un insieme di oggetti DataTable che contengono i dati. Oltre alla classe DataTable per le tabelle, sono disponibili anche le clas-

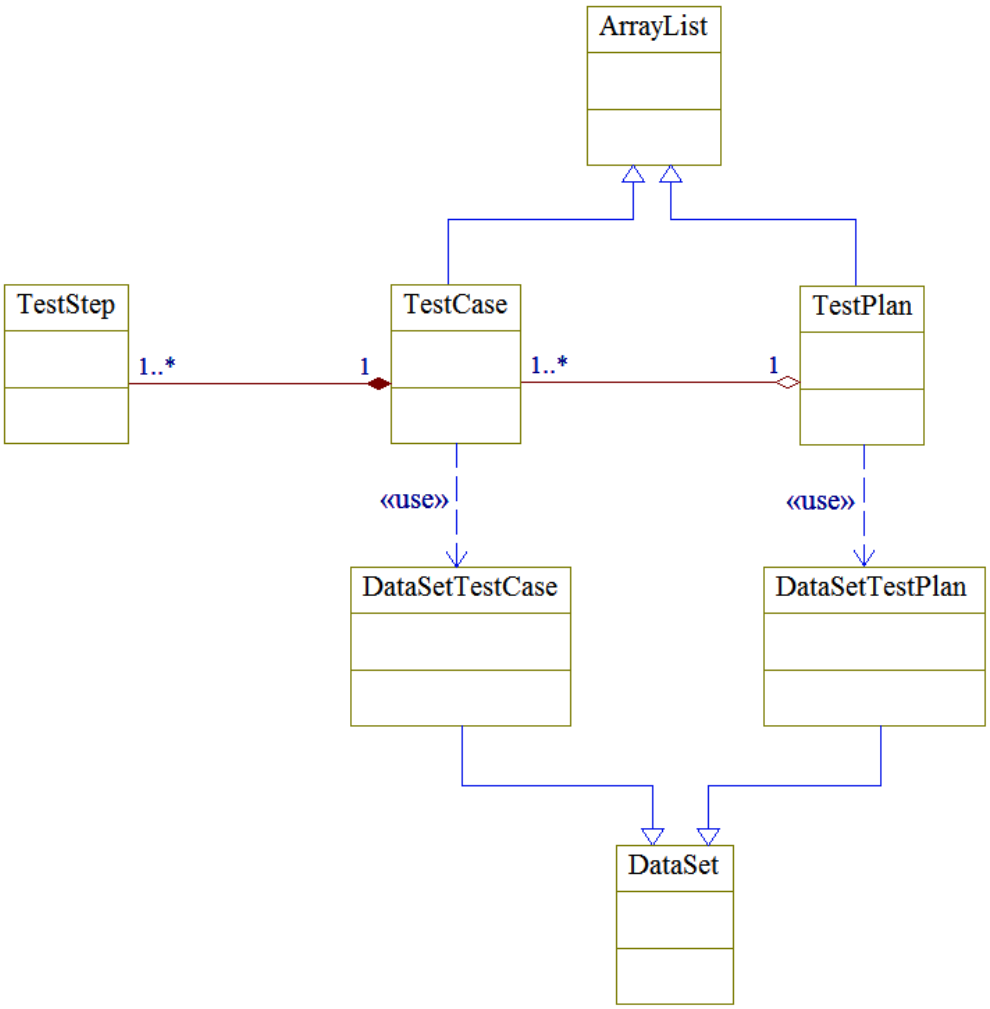

Figura 2.11: Diagramma delle classi per il modello dei dati.

si DataRow per le righe di tali tabelle e DataColumn per le colonne. Per gli scopi di questo progetto non è stato necessario utilizzare relazioni tra tabelle.

I documenti XML prodotti avranno tre livelli:

- livello 0: elemento radice;
- livello 1: elementi figli dell'elemento radice; sono di tipo complesso e rappresentano le tabelle nella visione relazionale; non contengono né nodi testo né nodi attributi ma solamente al-

tri elementi figli; il numero di elementi con un certo nome è uguale al numero di righe della tabella che ha quel nome;

- livello 2: elementi figli degli elementi di livello precedente; rappresentano le colonne della tabella e contengono i dati veri e propri delle tabelle.

2.5 Sviluppo del motore di esecuzione delle prove

Il motore di esecuzione deve effettuare le prove nell'ordine in cui queste appaiono nel piano di prova, rispettando i seguenti vincoli:

- durante l'esecuzione l'interfaccia con l'utente (paragrafo 2.3) deve continuare ad essere responsiva e deve poter eseguire altre operazioni;
- deve essere possibile in ogni momento controllare l'esecuzione (messa in pausa, riavvio, passaggio alla modalità passo-passo, arresto);
- deve essere mostrato in tempo reale il risultato dell'esecuzione dei singoli passi di prova e delle singole prove.

Per soddisfare questi vincoli è stata scelta come soluzione lo sviluppo del motore di esecuzione in un thread secondario che all'occorrenza può comunicare in modo asincrono con il thread principale ovvero con il thread che gestisce l'interfaccia grafica con l'utente.

Il flusso di comunicazione dal thread secondario al thread principale è necessario per controllare l'aggiornamento dell'interfaccia grafica (scrittura di messaggi di servizio per l'utente nell'area dedicata e/o visualizzazione in tempo reale dell'esito dell'esecuzione di ciascun passo di prova e di ciascuna prova nei controlli ListView corrispondenti). L'accesso ai controlli grafici di Windows Form in applicazioni con thread multipli presenta la seguente problematica: se più thread gestiscono lo stato di un controllo grafico è possibile

che questo venga forzato in uno stato incoerente; inoltre possono verificarsi altri problemi quali condizioni di corsa critica e condizioni di stallo. Per questo motivo, è importante accertarsi che i controlli grafici Windows Form vengano chiamati solamente dal thread che li ha creati. Del resto .NET Framework è predisposto per rilevare accessi non sicuri ai controlli grafici ed a generare in tal caso eccezioni del tipo InvalidOperationException. Per eseguire chiamate sicure ad un controllo Windows Form è necessario seguire i seguenti passi in ordine:

1. leggere la proprietà InvokeRequired del controllo a cui si intende accedere;

2. se InvokeRequired restituisce *true*, chiamare il metodo Invoke o il metodo BeginInvoke per far eseguire la chiamata effettiva al controllo tramite un delegato;

3. se InvokeRequired restituisce *false*, chiamare direttamente il controllo.

Si noti come sia assolutamente necessario, per evitare condizioni di stallo, utilizzare il metodo BeginInvoke che realizza una chiamata asincrona invece che il metodo Invoke che realizza una chiamata sincrona. Una chiamata sincrona infatti blocca il thread chiamante finchè non è stata completata, mentre una chiamata asincrona non si comporta in modo bloccante.

In Figura 2.12 è mostrato il diagramma di stato per il motore di esecuzione. Gli eventi il cui nome inizia con il prefisso "button" sono gli eventi associati alla pressione dei quattro pulsanti Run, Pause, Stop e Debug che controllano l'esecuzione del collaudo. L'evento denominato ExecutionThread_Done invece indica che sono state eseguite tutte le prove previste dal piano di collaudo.

Un secondo diagramma di stato, riportato in Figura 2.13, mostra il meccanismo mediante il quale viene aggiornato l'esito di ciascun

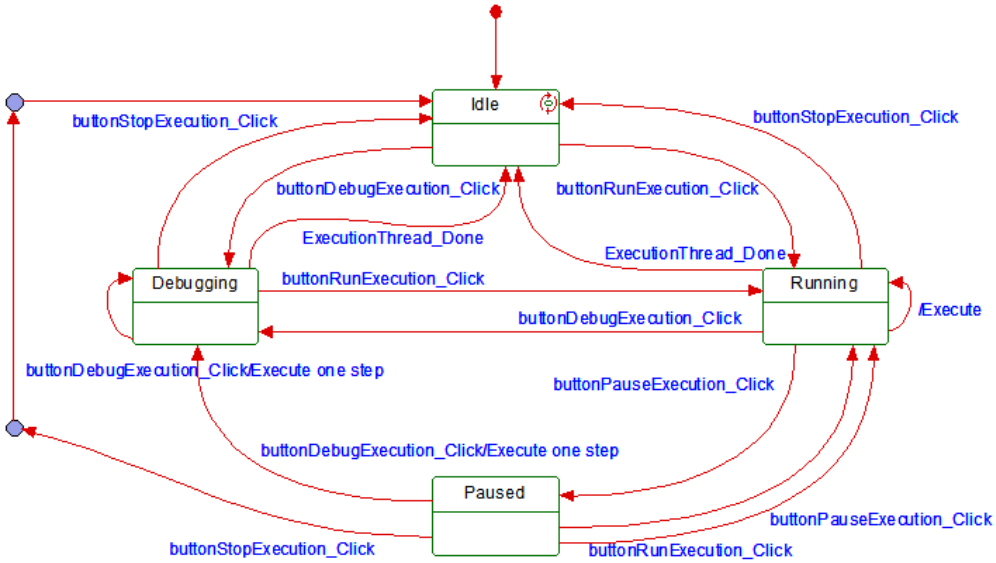

Figura 2.12: Diagramma di stato per il motore di esecuzione.

passo di prova e di ciascuna prova nell'interfaccia grafica con l'utente. Ogni volta che viene caricata da memoria di massa una prova, viene anche registrato per ciascun oggetto TestStep il gestore d'evento relativo al cambiamento dell'esito del corrispondente passo di prova. Similmente, ogni volta che viene caricato da memoria di massa un piano di collaudo e ogni volta che viene aggiunta una prova al piano di collaudo, viene anche registrato per ciascun oggetto TestCase un gestore d'evento relativo al cambiamento dell'esito della prova corrispondente. In questo modo, non appena un passo di prova oppure una prova vengono completati il gestore d'evento relativo provvede ad aggiornare i corrispondenti controlli grafici ListView presenti nell'interfaccia grafica in modo da mostrare l'icona colorata corrispondente all'esito.

Quando l'utente avvia l'esecuzione, il thread principale crea un oggetto della classe ExecutionThread (vedasi Figura 2.6) passando come argomenti al costruttore un riferimento all'oggetto MainForm, il piano di prova da eseguire ed altri argomenti relativi ai parametri di configurazione. A sua volta l'oggetto ExecutionThread crea

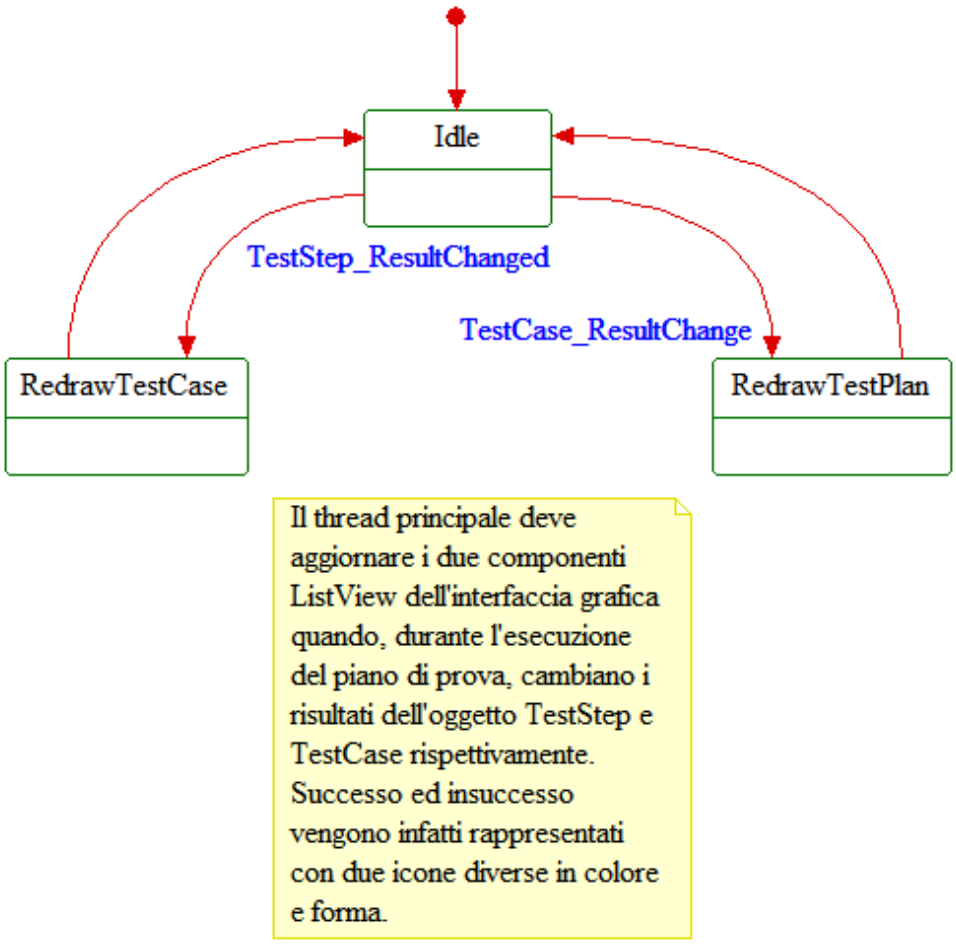

Figura 2.13: Diagramma di stato per l'aggiornamento dell'esito dei passi di prova e delle prove.

un oggetto del tipo Thread e lo avvia. Il thread esegue le seguenti operazioni nell'ordine:

1. avvia il cronometro;
2. azzera gli esiti di tutte le prove e di tutti i passi di prova;
3. inizializza le variabili locali utilizzate per la generazione delle statistiche di esecuzione;
4. calcola il numero totale di prove e di passi di prova considerando eventuali iterazioni;

5. calcola il numero di prove senza considerare eventuali iterazioni (per indicare la posizione relativa di una prova nel piano di prova);
6. apre una porta di comunicazione con l'adattatore I^2C e lo configura con la velocità di comunicazione e l'indirizzo impostati nel menù Config;
7. avvia la scrittura del resoconto se abilitato;
8. legge e visualizza il numero seriale, la versione del firmware e la tipologia di dispositivo connesso;
9. determina il passo di prova da eseguire tenendo conto di eventuali direttive di iterazione e lo esegue, visualizzando l'esito e l'eventuale causa di esito negativo ed inoltre, se previsto dalla configurazione, chiede all'utente dopo ogni passo di prova con esito negativo se intende proseguire con l'esecuzione oppure terminarla; questo viene ripetuto fino a quando non sono finite le prove previste dal piano di prova oppure fino a quando l'utente non arresta l'esecuzione;
10. visualizza le statistiche sull'esecuzione;
11. visualizza l'esito del piano di prova;
12. ferma il cronometro, determina la durata totale dell'esecuzione e la visualizza.

I passi di prova del tipo *Sleep*, *Send Command* e *User Feedback* vengono eseguiti tramite metodi della classe ExecutionThread. I passi di prova *Load Image*, *Capture Image*, *Analyze Image* e *User Message* invece, poichè richiedono l'accesso a variabili globali private della classe MainForm ed in alcuni casi l'accesso ai controlli grafici (per la gestione delle immagini), chiamano metodi della classe MainForm tramite il riferimento che l'oggetto ExecutionThread ha ricevuto nel suo costruttore.

Per realizzare la funzionalità di comunicazione con il motore di scansione, necessaria all'esecuzione dei passi di prova del tipo *Send Command* e che avviene tramite bus I^2C, si è utilizzata la software Application Programming Interface (API) fornita dallo stesso produttore del dispositivo *Aardvark* (descritto nel paragrafo 1.3). Tale interfaccia software viene fornita come una libreria dinamica (Dynamic-Link Library, DLL) precompilata e corredata da codice sorgente di adattamento da compilare insieme al resto del codice sorgente che utilizza l'interfaccia stessa.

L'interfaccia software *Aardvark* viene incapsulata nel sistema dalla libreria dinamica già citata nel paragrafo 1.3 e rappresentata in Figura 2.3; ciò rende il software applicativo indipendente dal particolare tipo di interfaccia fisica utilizzata a tale scopo, permettendo anche l'estensione in futuro a più tipi di interfacce I^2C.

La libreria di incapsulamento, dedicata esclusivamente alle funzioni di comunicazione con l'adattatore I^2C, è stata sviluppata in C++. Pertanto, il codice gestito dal *Common Language Runtime* del .NET Framework deve potersi avvalere di appositi meccanismi di *interoperabilità* con il codice non gestito della sopracitata libreria dinamica per il controllo della interfaccia I^2C.

Il *Common Language Runtime* (CLR) fornisce due meccanismi per l'interoperabilità con il codice non gestito:

- *Platform Invoke*, che consente la chiamata di funzioni esportate da una libreria non gestita da parte del codice gestito;

- l'interoperabilità COM (Component Object Model), che consente l'interazione del codice gestito con oggetti COM mediante interfacce.

Nel caso in oggetto è stato necessario sfruttare il meccanismo Platform Invoke, affidando al CLR la mappatura dei tipi di parametro di metodo cioè quel processo di conversione e "serializzazione" dei parametri indicato nella terminologia di lingua inglese come

"marshalling" e necessario per consentirne lo scambio di dati tra i due ambienti.

Per la maggior parte dei tipi di dati, infatti, esistono rappresentazioni comuni sia nell'ambiente gestito che in quello non gestito. Questi tipi, denominati *tipi copiabili*, vengono trattati dal gestore di marshalling di interoperabilità senza conversione. Per altri tipi la mappatura potrebbe essere ambigua oppure essi potrebbero non essere rappresentati affatto nell'ambiente gestito. È possibile pertanto fornire istruzioni esplicite al gestore di marshalling su come eseguire la mappatura in caso di ambiguità. Esistono molti fattori che influiscono sul modo in cui il CLR esegue il marshalling dei dati tra ambienti gestiti e non gestiti, tra cui la presenza di attributi quali [*MarshalAs*], [*StructLayout*], [*InAttribute*] e [*OutAttribute*] e di parole chiave del linguaggio quali *out* e *ref* in C#.

Per ogni metodo importato dalla libreria dinamica è stato necessario utilizzare in C# l'attributo [*DllImport*], contenuto nello spazio dei nomi *System.Runtime.InteropServices*, al fine di specificare il nome completo della libreria e la convenzione di chiamata dei metodi:

```
[DllImport("device.dll", CallingConvention =
    CallingConvention.Cdecl)]
```

All'attributo è stata fatta seguire la dichiarazione del metodo, nel seguente formato:

```
extern static tipo-restituito nome-metodo (
    lista-di-parametri)
```

Per esportare i metodi è stato necessario aggiungere, nel sorgente C++ della libreria, l'attributo *__declspec(dllexport)* prima del loro nome sia nella dichiarazione che nella definizione. Al fine di evitare la cosiddetta decorazione dei nomi dei metodi da parte del compilatore C++, è stato inoltre necessario definire i metodi esportati come *extern "C"*.

La Figura 2.14 mostra il diagramma di sequenza per un tipico caso in cui l'applicazione viene utilizzata per eseguire un piano di prova che prevede l'invio al dispositivo di due comandi sul canale I^2C. Un diagramma di sequenza viene usato per modellare le interazioni tra oggetti che realizzano un caso d'uso o una parte di un caso d'uso e rappresenta il modo più semplice per focalizzarsi sull'effettiva sequenza temporale degli eventi [11]. All'avvio dell'applicazione viene creato un oggetto di tipo TextBox per la casella di testo dedicata alla visualizzazione dei messaggi di servizio, un oggetto di tipo TestPlan per contenere il piano di prova, un oggetto di tipo TestCase per contenere l'unica prova presente nel piano di prova ed infine due oggetti di tipo TestStep per contenere i due passi di prova *Send Command* di cui è costituita tale prova. Successivamente l'utente preme il pulsante Run dell'interfaccia grafica per avviare il collaudo e viene quindi generato un evento del tipo buttonRunExecution_Click. A questo punto, come spiegato poc'anzi, il gestore dell'evento di avvio collaudo crea un oggetto del tipo ExecutionThread il quale a sua volta crea un oggetto della classe Thread fornita dalla Framework Class Library, cioè un thread secondario dedicato al motore di esecuzione che infine avvia. Il thread del motore di esecuzione, prima di iniziare l'interpretazione e l'esecuzione del piano di collaudo, tramite funzioni della libreria C++ device.dll apre la porta di comunicazione con l'adattatore I^2C e configura quest'ultimo. Terminata questa fase di inizializzazione dell'adattatore, vengono inviati i comandi al modulo di scansione secondo quanto previsto dal piano di collaudo e vengono lette le corrispondenti risposte provenienti dal dispositivo. Se le risposte ricevute sono congruenti con quelle previste per il superamento del relativo passo di prova, viene conteggiato un esito positivo altrimenti viene conteggiato un esito negativo. Ogni volta che viene determinato l'esito di un passo di prova o di una prova questo viene visualizzato nella casella di testo riservata ai messaggi di servizio. Infine, terminata l'esecuzione del piano di collaudo, viene chiusa la porta di comunicazione con l'adattatore I^2C.

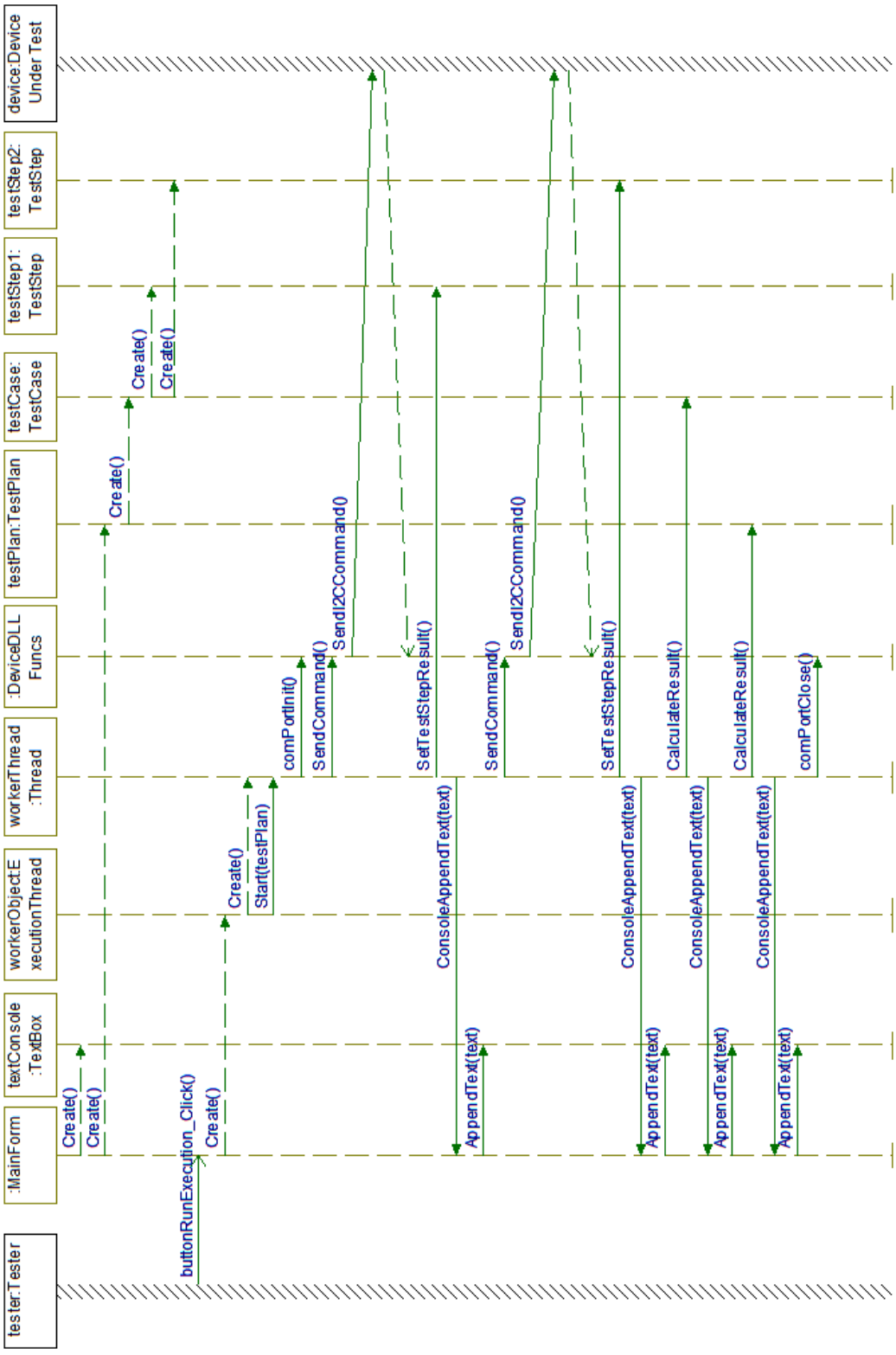

Figura 2.14: Diagramma di sequenza per un tipico caso di esecuzione (sessione di invio comandi).

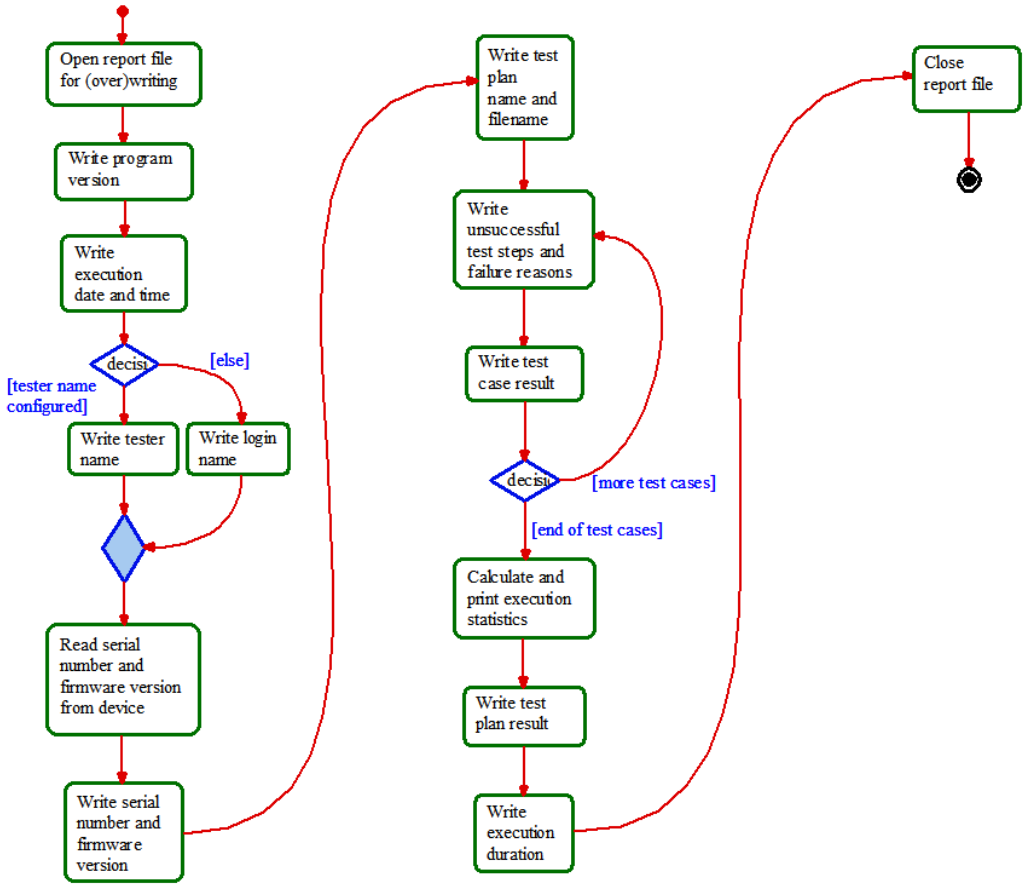

Figura 2.15: Diagramma di attività per il generatore di resoconto.

2.6 Sviluppo del generatore di resoconto

La generazione del resoconto, quando abilitata, è ottenuta da istruzioni ad esecuzione condizionale definite nel codice eseguibile del thread secondario creato dal motore di esecuzione descritto nel paragrafo 2.5. La Figura 2.15 mostra le attività che vengono portate a termine dal flusso di istruzioni relativo alla generazione del resoconto.

Di seguito viene riportato un esempio di resoconto prodotto dal sistema di collaudo al termine di una sessione comprendente 18 prove. Si può notare come vengano segnalate tutte le informazioni previste dai requisiti. Si può anche notare come due delle prove abbiano

avuto esito negativo: ciò si è verificato in quanto tali prove corrispondevano a comandi documentati nel piano di collaudo ufficiale ma non ancora implementati dalla azienda nel firmware del modulo di scansione.

```
Scan Engine Test Program version 1.0.4.1
------------------------------------------------------------------
Execution date and time: 19/12/2014 16:29:20
Tester Name: gtrentalancia
Serial number: G13M55536
Firmware version: APPL1.0.0.C
Family ID: DE2011
------------------------------------------------------------------
Test Plan Filename: C:\Users\gtrentalancia\Documents\PianoDiProva.tp
------------------------------------------------------------------
test case[1/18]: success
test case[2/18]: success
test case[3/18]: success
test case[4/18]: success
test case[5/18]: success
test case[6/18]: success
test case[7/18]: success
test case[8/18]: success
test case[9/18]: success
test case[10/18]: success
test case[11/18]: success
test case[12/18]: success
test case[13/18]: success
test case[14/18]: success
test step[1/1]: failed: Unexpected answer from device ([NACK]).
test case[15/18]: failure
test case[16/18]: success
test step[1/1]: failed: Unexpected answer from device ([NACK]).
test case[17/18]: failure
test case[18/18]: success
------------------------------------------------------------------
                          Execution Summary
Executed 18 out of 18 test cases (100%).
- Successful test cases: 16 out of 18 (88,88889%).
- Unsuccessful test cases: 2 out of 18 (11,11111%).
- Undetermined test cases: 0 out of 18 (0%).
Executed 50 out of 50 test steps (100%).
- Successful test steps: 48 out of 50 (96%).
```

```
- Unsuccessful test steps: 2 out of 50 (4%).
- Undetermined test steps: 0 out of 50 (0%).
------------------------------------------------------------------
Test Plan result: failure
Total execution duration: 00:00:12.8481080
```

Tramite questa funzionalità è possibile creare un archivio aziendale di tutti i collaudi eseguiti, tenere traccia dei difetti di prodotto e monitorare nel tempo lo sviluppo dal punto di vista della qualità. A partire dall'analisi del resoconto di collaudo è possibile poi produrre manualmente relazioni più estese sullo stato di avanzamento dello sviluppo del prodotto nonchè pianificare le attività di sviluppo e collaudo future.

Capitolo 3

Validazione

In questo capitolo vengono riportate tutte le prove che sono state eseguite sull'applicazione *Scan Engine Test Program* per validarne la corretta funzionalità.

3.1 Validazione dell'interfaccia grafica con l'utente

`PROVA 1 (Menù Config)`: Dal menù Config deve essere possibile configurare sia l'indirizzo che la velocità dell'adattatore I^2C. Deve essere visualizzata una finestra ausiliaria di errore se vengono inseriti parametri invalidi: provare ad inserire un valore troppo lungo, quale ad esempio "5CD", nella casella di testo relativa all'indirizzo I^2C ed un valore negativo, quale ad esempio "-400", nella casella di testo relativa alla velocità di comunicazione. La prova è superata se vengono prodotti in entrambe i casi messaggi di errore in finestre ausiliarie.

`PROVA 2 (Menù Config)`: Dal menù Config deve essere possibile configurare la risoluzione della scheda di acquisizione immagini (larghezza ed altezza). Deve essere visualizzata una finestra ausiliaria di errore se vengono inseriti parametri invalidi: provare ad inserire valori negativi o valori contenenti lettere per la larghezza e per la lunghezza (ad esempio "-480" o "A3"). La prova è superata

se viene visualizzata una finestra ausiliaria di errore ogni volta che viene inserito un carattere invalido.

`PROVA 3 (Menù Config)`: Dal menù Config deve essere possibile selezionare e configurare l'indirizzo IP della scheda di acquisizione immagini. Selezionare l'opzione *Frame grabber selection*. Si deve aprire una finestra ausiliaria dal titolo "GigE Vision Device Selection". Se la finestra ausiliaria si è aperta correttamente, la prova è superata.

`PROVA 4 (Menù Help)`: Selezionare il menù Help e poi selezionare l'opzione *About SETP*. Deve essere visualizzata una finestra ausiliaria in cui vengono stampate la versione dell'applicazione, la versione dell'ambiente .NET Framework e l'autore del programma. Tali informazioni devono coincidere con quelle visualizzate nell'area messaggi dell'applicazione quando essa viene avviata. Se la finestra ausiliaria viene aperta correttamente e vengono visualizzate le informazioni sull'applicazione, la prova è superata.

`PROVA 5 (Menù Help)`: Selezionare il menù Help e poi selezionare l'opzione *Help*. Deve apparire una finestra ausiliaria in cui viene visualizzata la guida d'utente. Selezionare il primo collegamento ipertestuale della guida denominato *Designer View* e poi ancora il primo collegamento ipertestuale della pagina successiva denominato *Design panel*. Nella pagina che appare deve venire visualizzata l'immagine corrispondente al pannello di progettazione dell'interfaccia grafica dell'applicazione. La prova si può considerare superate se la guida d'utente viene aperta e se vengono in essa visualizzate le immagini.

`PROVA 6 (Selezione della vista)`: Tramite il menù a tendina presente a destra del menù Help e a sinistra del pulsante Exit, deve essere possibile cambiare la vista (o modalità di esecuzione)

dell'applicazione. Provare a selezionare l'altra vista presente elencata nel menù a tendina: l'aspetto dell'applicazione deve cambiare (vedasi anche Figura 2.7 e Figura 2.8) perchè si è passati dalla vista di progetto alla vista di collaudo (o viceversa). Provare a cambiare nuovamente vista per tornare nella vista iniziale. Se è possibile cambiare correttamente da una vista all'altra, la prova è superata.

`PROVA 7 (Pulsante Exit)`: Premendo il pulsante sinistro del dispositivo di puntamento quando il puntatore si trova in corrispondenza del pulsate Exit presente sulla barra principale dei menù (a destra del menù a tendina per la selezione della vista), l'applicazione deve essere terminata. Provare a terminare l'applicazione tramite tale pulsante. Se l'applicazione viene terminata correttamente, la prova è superata.

`PROVA 8 (Uscita dall'applicazione)`: L'applicazione deve poter essere terminata anche ricorrendo al pulsante con il simbolo di "X" situato nell'angolo in alto a destra. Provare a terminare l'applicazione premendo il tasto sinistro del dispositivo di puntamento quando il puntatore si trova in corrispondenza del suddetto pulsante dell'applicazione. Se l'applicazione viene terminata, la prova ha avuto successo.

`PROVA 9 (Uscita dall'applicazione)`: Un terzo ed ultimo modo per uscire dall'applicazione è la scelta dell'opzione *Exit* dal menù attivabile premendo il tasto destro del dispositivo di puntamento sull'icona dell'applicazione dell'area di notifica. Se è possibile terminare l'applicazione tramite questa opzione, la prova ha avuto successo.

`PROVA 10 (Collegamento web all'azienda)`: Premere sull'indirizzo web dell'azienda visualizzato in alto a destra sulla barra dei menù. Se viene aperta nel navigatore web la pagina principale

dell'azienda (richiede connessione a Internet), la prova è superata.

3.2 Validazione della fase di progetto delle prove

`PROVA 11 (Salvataggio di una prova):` Nel pannello di progettazione della prova (a sinistra nella vista progettuale), premere il pulsante Append, verrà aggiunto un passo di prova del tipo "No Operation". Nella casella di testo etichettata "Test Case Name" inserire una breve descrizione, ad esempio "Test Case 1". A questo punto selezionare dal menù Test Design l'opzione *Save*: verrà aperta una finestra ausiliaria mediante la quale è possibile scegliere cartella di destinazione e nome del documento da salvare. Dopo essersi posizionati nella cartella di destinazione desiderata ed aver inserito il nome del documento sul quale si desidera salvare la prova, premere il pulsante Save in basso a destra nella finestra ausiliaria. Per verificare che il documento è stato scritto correttamente, anzitutto selezionare l'opzione *New* dal menù Test Design per rimuovere la prova attualmente visualizzata: dovrebbe venire visualizzata una prova vuota. Selezionare quindi l'opzione *Load* dal menù Test Design e, nella finestra ausiliaria che verrà visualizzata, selezionare il documento di prova precedentemente creato. Se è presente su memoria di massa il documento di prova creato precedentemente e se dopo averlo caricato viene visualizzata la prova contenente il solo passo di prova del tipo "No Operation" e recante la descrizione precedentemente inserita, la prova è stata superata con successo.

`PROVA 12 (Progettazione di una prova):` Con questa prova si intende verificare la funzionalità di progettazione delle prove di collaudo del modulo di scansione. Come prima cosa, creare una nuova prova selezionando l'opzione *New* dal menù Test Design: apparirà una prova vuota. A questo punto premere il pulsante Append per inserire un passo di prova: dovrebbe apparire un passo di prova

del tipo "No Operation" e dovrebbero venire anche attivati i pulsanti Remove ed Insert situati a sinistra del pulsante Append. Cambiare il tipo del passo di prova che è stato appena creato utilizzando il menù a tendina che appare sotto alla lista dei passi di prova, selezionando ad esempio "User Message" invece che "Sleep": dovrebbe cambiare anche la descrizione del passo di prova visualizzato nella lista. Ora premere il tasto Insert: dovrebbe venire inserito un nuovo passo di prova del tipo "No Operation" in cima alla lista. Premere nuovamente il pulsante Append: dovrebbe venire inserito un nuovo passo di prova del tipo "No Operation" però questa volta in fondo alla lista. Infine, selezionare con il dispositivo di puntamento il passo di prova del tipo "User Message" precedentemente creato e poi premere il pulsante Remove: dovrebbe venire rimosso il passo di prova selezionato. Se tutte le operazioni sono state eseguite correttamente e se nella lista rimangono solo due passi del tipo "No Operation", la prova è stata eseguita correttamente e superata.

`PROVA 13 (Menù contestuale):` Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Dopo essersi posizionati con il dispositivo di puntamento sull'area occupata dalla lista, premere il tasto destro del dispositivo di puntamento: dovrebbe apparire un riquadro contenente un piccolo menù (menù contestuale) con la sola opzione *Append*. Successivamente premere il pulsante Append per inserire un passo di prova: verrà visualizzato nella lista un nuovo passo di prova del tipo "No Operation". Posizionarsi nuovamente sull'area occupata dalla lista e quindi premere di nuovo il pulsante destro del dispositivo di puntamento: dovrebbe apparire un menù contestuale contenente le opzioni *Remove*, *Insert*, *Append*. Se tutte le operazioni si sono concluse come descritto e se il menù contestuale è apparso in entrambe i casi, la prova può considerarsi superata con successo.

`PROVA 14 (Richiesta di salvataggio):` Selezionare l'opzio-

ne *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. Ora uscire dalla applicazione ad esempio premendo il pulsante Exit sulla barra principale dell'applicazione. Verificare che appaia una finestra ausiliaria dal titolo "Exit or Continue?" in cui viene chiesta conferma di voler uscire dall'applicazione. Selezionare "No" in tale finestra ausiliaria: la finestra ausiliaria dovrebbe scomparire e l'applicazione dovrebbe rimanere aperta. Se è comparsa la finestra ausiliaria e se poi l'applicazione non è stata terminata, la prova è stata superata con successo.

`PROVA 15 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto provare ad aumentare la durata della pausa da 0 millisecondi ad un valore non nullo tramite la barra a scorrimento presente nel sottopannello di configurazione. Provare anche a cambiare la durata della pausa agendo direttamente sul valore contenuto nella casella di testa presente sotto la barra a scorrimento. Infine premere il pulsante Insert: dovrebbe essere inserita un nuovo passo di prova all'inizio della lista e la descrizione del passo di prova già presente dovrebbe contenere la dicitura "Sleep" ed indicare la durata precedentemente configurata per la pausa. Se è possibile cambiare il valore della durata della pausa e se la descrizione del passo di prova viene aggiornata correttamente, la prova può considerarsi superata con successo.

`PROVA 16 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto tramite il menù a tendina cambiare il tipo di passo di prova da "Sleep" a "Send Command". Tramite il menù a tendina presente nel sottopannello di configurazione selezionare uno tra i comandi disponibili. Quindi spostarsi sulla casella di testo sottostante con

etichetta "Parameters" selezionandola: se ci si sofferma con il dispositivo di puntamento su tale casella di testo, dovrebbe apparire una indicazione su come configurare tale campo. Nel menù a tendina sottostante con etichetta "Expected Answer" selezionare l'opzione "No Check". A questo punto nell'ultima casella di testo con etichetta "I2C Address" provare ad inserire il valore "5CD": dovrebbe apparire una finestra ausiliaria di errore, poichè il campo inserito è troppo lungo; alla chiusura della finestra ausiliaria di errore dovrebbe venire ripristinato il valore "5C". Premere il pulsante Insert: dovrebbe essere aggiunto in cima alla lista un nuovo passo di prova e questo dovrebbe risultare selezionato. Nella lista dei passi di prova selezionare il passo di prova del tipo "Send Command" precedentemente creato: dovrebbe riapparire il sottopannello di configurazione ed in particolare dovrebbero venire visualizzati i parametri che erano stati inseriti in precedenza. Se è stato possibile configurare il passo di prova e se tutte le altre condizioni sono risultate verificate, la prova è superata con successo.

`PROVA 17 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto tramite il menù a tendina cambiare il tipo di passo di prova da "Sleep" a "Load Image". Nel sottopannello inferiore premere il pulsante "Browse Images": si aprirà una finestra ausiliaria mediante la quale dovrà essere selezionata un'immagine da memoria di massa. Verificare che il nome dell'immagine selezionata tramite la finestra ausiliaria sia visibile nella casella di testo con etichetta "File Name" posta sopra al pulsante. Se è stato possibile selezionare un'immagine e se il nome dell'immagine è stato visualizzato nella casella di testo corrispondente, la prova è superata.

`PROVA 18 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova.

Aggiungere un passo di prova premendo il pulsante Append. A questo punto tramite il menù a tendina cambiare il tipo di passo di prova da "Sleep" a "Capture Image". Nel sottopannello inferiore i campi "File Name", "Save Folder" e "Number of frames" dovrebbero apparire ombreggiati. A questo punto selezionare la casella di spunta etichettata "Save Image": dovrebbe scomparire l'ombreggiatura dei campi sottostanti. Inserire un nome per l'immagine nella casella di testo etichettata "File Name", poi premere il pulsante "Browse Folder" e selezionare una cartella, infine aumentare il campo "Number of frames" da 1 ad un altro valore, utilizzando il pulsante con il simbolo di freccia in alto. Premere il pulsante Insert: dovrebbe essere aggiunto in cima alla lista un nuovo passo di prova e questo dovrebbe risultare selezionato. Nella lista dei passi di prova selezionare il passo di prova del tipo "Capture Image" precedentemente creato: dovrebbe riapparire il sottopannello di configurazione ed in particolare dovrebbero venire visualizzati i parametri che erano stati inseriti in precedenza. Se è stato possibile configurare il passo di prova, se tutte le operazioni si sono concluse come previsto e se alla fine è riapparso il sottopannello contenente tutti i parametri di configurazione precedentemente selezionati, la prova può considerarsi superata con successo.

`PROVA 19 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto, tramite il menù a tendina, cambiare il tipo di passo di prova da "Sleep" a "Analyze Image". Nel menù a tendina presente nel sottopannello inferiore dovrebbe apparire selezionata l'opzione "Brightness". Provare a selezionare un'altra opzione tra quelle disponibili. Premere il pulsante sinistro o destro del dispositivo di puntamento in corrispondenza del simbolo di disuguaglianza: ad ogni pressione dovrebbe venire cambiato il senso della disuguaglianza. Infine provare a cambiare il valore numerico contenuto nella casella

presente a destra del simbolo di disuguaglianza. Se è stato possibile configurare il passo di prova, la prova può considerarsi superata con successo.

`PROVA 20 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto tramite il menù a tendina cambiare il tipo di passo di prova da "Sleep" a "User Message". Al centro del sottopannello di configurazione è presente una casella di testo grande. In tale casella di testo inserire un messaggio di prova. Infine premere il pulsante "Browse Image" presente in basso e selezionare un'immagine tramite la finestra ausiliaria che apparirà: l'immagine selezionata dovrebbe apparire in basso. Se è stato possibile configurare il passo di prova come descritto, la prova può considerarsi superata con successo.

`PROVA 21 (Configurazione dei passi di prova)`: Selezionare l'opzione *New* dal menù Test Design per creare una nuova prova. Aggiungere un passo di prova premendo il pulsante Append. A questo punto tramite il menù a tendina cambiare il tipo di passo di prova da "Sleep" a "User Feedback". Al centro del sottopannello di configurazione è presente una casella di testo grande. In tale casella di testo inserire una domanda di prova e poi selezionare la risposta "No" tramite la casella di selezione. Quindi premere il tasto Insert per inserire un nuovo passo di prova. Selezionare il passo di prova del tipo "User Feedback" precedentemente creato: dovrebbero apparire nel sottopannello di configurazione la domanda precedentemente inserita e la risposta "No" selezionata. Se è stato possibile configurare il passo di prova e se è stato successivamente possibile richiamare tale configurazione, la prova può ritenersi superata con successo.

3.3 Validazione della fase di progetto dei piani di prova

`PROVA 22 (Salvataggio di un piano di prova)`: Nel pannello di progettazione del piano di prova (a destra nella vista di progettazione) premere il pulsante "Browse Test Cases" in alto a sinistra e tramite la finestra ausiliaria che apparirà selezionare da memoria di massa il documento precedentemente creato nella `PROVA 11`. Al centro del pannello dovrebbe apparire la prova. A questo punto selezionare tale prova premendo il pulsante sinistro del dispositivo di puntamento. Mentre la prova è selezionata premere il pulsante "Add Iteration": dovrebbe apparire un nuovo elemento in cima alla lista. Posizionarsi con il dispositivo di puntamento sopra tale elemento e premere il pulsante destro: dovrebbe apparire un menù contestuale. Nel menù contestuale configurare l'opzione "Number of loop iterations" con un valore di 10: anche la descrizione dell'elemento dovrebbe cambiare in modo da visualizzare la dicitura "— Iterate next 1 element 10 times —". Salvare il piano di prova così creato, selezionando l'opzione *Save* dal menù Test Plan e scegliendo nella finestra ausiliaria che apparirà una cartella di destinazione ed un nome per il documento. Per verificare che il salvataggio è avvenuto correttamente, anzitutto creare un nuovo piano di prova tramite l'opzione *New* del menù Test Plan. Dopo la selezione dell'opzione *New* devono risultare disabilitati i pulsanti Remove, Move Up e Move Down. Poi caricare il piano di prova precedentemente salvato tramite l'opzione *Load* del menù Test Plan: dovrebbe riapparire il piano di prova precedentemente creato. Se è stato possibile creare, salvare e poi ricaricare il piano di prova di esempio come descritto, la prova può considerarsi superata con successo.

`PROVA 23 (Progettazione di un piano di prova)`: Selezionare l'opzione *New* dal menù Test Plan per creare un nuovo piano di prova. Aggiungere, tramite il pulsante "Browse Test Cases", alme-

no tre diverse prove precedentemente create e salvate su memoria di massa. Tramite la pressione del tasto destro del dispositivo di puntamento attivare il menù contestuale e provare ciascuna delle seguenti funzionalità disponibili: *Move Up*, *Move Down*, *Move to Top*, *Move to Bottom* e *Remove*. Se è possibile spostare la prova selezionata in tutti i vari modi possibili (su, giù, in cima alla lista, in fondo alla lista) e se è possibile rimuovere una o più prove, l'esito di questa prova è positivo.

`PROVA 24 (Ricaricamento di un piano di prova):` Selezionare l'opzione *New* dal menù Test Plan per creare un nuovo piano di prova. Aggiungere, tramite il pulsante "Browse Test Cases", un numero di prove a piacere. Successivamente inserire nella casella di testo in alto con etichetta "Test Plan Name" una breve descrizione del piano di prova. Salvare il piano di prova su memoria di massa tramite l'opzione *Save* o *Save as...* del menù Test Plan. A questo punto rimuovere tutte le prove premendo ripetutamente il pulsante Remove e cancellare la descrizione inserita nella casella di testo in alto: rimarrà un piano di prova vuoto. Infine, selezionare l'opzione *Reload* dal menù Test Plan. Se il piano di prova viene ricaricato ed esso contiene tutte le prove che conteneva quando era stato salvato, si può concludere che questa prova è stata superata con successo.

`PROVA 25 (Modalità di collaudo):` Passare dalla modalità (o vista) progettuale a quella di collaudo tramite il menù a tendina situato in alto sulla barra principale dei menù dell'applicazione. Dal menù Test Execution (pannello di sinistra) selezionare l'opzione *Load* e caricare da memoria di massa un piano di prova non vuoto precedentemente creato. Questa prova può considerarsi superata con successo se, ogni volta che viene selezionata una prova nel piano di prova, nella parte inferiore vengono visualizzati i passi che costituiscono tale prova.

3.4 Validazione della fase di esecuzione dei piani di prova

`PROVA 26 (Vista di collaudo)`: Passare alla modalità (o vista) progettuale tramite il menù a tendina situato in alto sulla barra principale dei menù dell'applicazione. Caricare un piano di prova tramite l'opzione *Load* del menù Test Plan. Poi caricare una prova tramite l'opzione *Load* del menù Test Design. Passare infine alla modalità (o vista) di collaudo tramite il menù a tendina situato in alto sulla barra principale dei menù ed eseguire il piano di prova tramite il pulsante Run o tramite l'opzione *Run Test Plan* del menù Run Test nel pannello di destra. Se nella parte inferiore del pannello di sinistra vengono visualizzati i passi di prova del piano di prova (e non ad esempio i passi di prova della prova caricata successivamente), è possibile concludere che questa prova è stata superata con successo.

`PROVA 27 (Esecuzione di un piano di prova)`: Passare alla modalità (o vista) progettuale tramite il menù a tendina situato in alto sulla barra principale dei menù dell'applicazione. Creare una nuova prova tramite l'opzione *New* del menù Test Design. Aggiungere un passo di prova tramite il pulsante Append. Configurare il passo di prova appena aggiunto come "Load Image" e selezionare, tramite il pulsante "Browse Images", un'immagine da caricare da memoria di massa. Aggiungere un secondo passo di prova tramite il pulsante Append e configurarlo come "User Message". Nella casella di testo presente nel sottopannello inferiore inserire il seguente testo: "Selezionare un'area dell'immagine e poi premere OK". Aggiungere un terzo passo di prova tramite il pulsante Append e configurarlo come "Analyze Image". Nel sottopannello inferiore di configurazione aumentare il valore di soglia da 0 a 10. Salvare la prova così creata su memoria di massa tramite l'opzione *Save* o *Save as...* del menù Test Design. A questo punto passare alla modalità (o vista) di collaudo tramite il menù a tendina situato in alto sulla barra prin-

cipale dei menù. Nel menù Test Execution presente nel pannello di sinistra selezionare l'opzione *New* per creare un nuovo piano di prova. Quindi premere il pulsante "Browse Test Cases" presente nel pannello di sinistra e, tramite la finestra ausiliaria che apparirà, selezionare la prova precedentemente creata e salvata su memoria di massa. Ora è possibile avviare il collaudo: nel pannello di destra premere il pulsante Run oppure selezionare l'opzione *Run Test Plan* dal menù Run Test. Dovrebbe venire caricata l'immagine precedentemente selezionata nel riquadro di visualizzazione presente al centro del pannello di destra e dovrebbe anche apparire una finestra ausiliaria che visualizza il testo precedentemente inserito durante la composizione della prova. Selezionare un'area non troppo scura dell'immagine e quindi premere il pulsante OK sulla finestra ausiliaria: dovrebbe così completarsi l'esecuzione del collaudo. Nel pannello di sinistra dovrebbero essere visibili sulla sinistra gli esiti di tutti i passi di prova e l'esito complessivo della prova. Nell'area messaggi presente in basso nel pannello di destra dovrebbe essere visualizzato un resoconto dell'esecuzione (utilizzare, se necessario, le frecce su e giù per scorrere il testo). Se nel resoconto visualizzato nell'area messaggi per l'utente appare verso la fine la dicitura "Test Plan result: success" e se sono visibili a fianco dei passi di prova e della prova icone dal colore verde ad indicare esito positivo, è possibile concludere che questa prova ha avuto successo.

`PROVA 28 (Esecuzione di un piano di prova)`: Collegare il modulo di scansione al calcolatore elettronico. Passare alla modalità (o vista) progettuale tramite il menù a tendina situato in alto sulla barra principale dei menù dell'applicazione. Creare una nuova prova tramite l'opzione *New* del menù Test Design. Aggiungere un passo di prova tramite il pulsante Append. Configurare il passo di prova appena aggiunto come "Send Command" e selezionare, tramite il menù a tendina, il comando "CAMERA START (0x38)". Nel campo "Parameters" inserire il valore 1. Aggiungere un secon-

do passo di prova tramite il pulsante Append e configurarlo come "Send Command". Selezionare, tramite il menù a tendina, il comando "ILLUMINATION ENABLE (0x39)" e nel campo "Parameters" inserire il valore 1. Aggiungere un terzo passo di prova tramite il pulsante Append e configurarlo come "Capture Image". Nel sottopannello di configurazione inferiore selezionare la casella di spunta "Save Image", quindi inserire nel campo "File Name" un nome (senza estensione) da utilizzare per salvare l'immagine su memoria di massa ed infine premere il pulsante "Browse Folder" per selezionare, tramite finestra ausiliaria, la cartella di destinazione. Aggiungere un quarto passo di prova tramite il pulsante Append e configurarlo come "Send Command". Selezionare, tramite il menù a tendina, il comando "ILLUMINATION ENABLE (0x39)" e nel campo "Parameters" inserire il valore 0. Aggiungere un quinto passo di prova tramite il pulsante Append e configurarlo nuovamente come "Send Command". Selezionare, tramite il menù a tendina, il comando "CAMERA START (0x38)" e nel campo "Parameters" inserire il valore 0. Aggiungere un sesto passo di prova tramite il pulsante Append e configurarlo come "User Feedback". Nella casella di testo presente nel sottopannello inferiore inserire il seguente testo: "Se si è accesa la luce del modulo di scansione e se è apparsa l'immagine acquisita nel riquadro di visualizzazione, premere YES altrimenti premere NO.". Salvare la prova così creata su memoria di massa tramite l'opzione *Save* o *Save as...* del menù Test Design. A questo punto passare alla modalità (o vista) di collaudo tramite il menù a tendina situato in alto sulla barra principale dei menù. Nel menù Test Execution presente nel pannello di sinistra selezionare l'opzione *New* per creare un nuovo piano di prova. Quindi premere il pulsante "Browse Test Cases" presente nel pannello di sinistra e, tramite la finestra ausiliaria che apparirà, selezionare la prova precedentemente creata e salvata su memoria di massa. Ora è possibile avviare il collaudo: nel pannello di destra premere il pulsante Run oppure selezionare l'opzione *Run Test Plan* dal menù Run Test. Durante l'e-

secuzione verrà aperta una finestra ausiliaria mediante la quale dovrà essere selezionata la scheda di acquisizione immagini e dovrà essere configurato il suo indirizzo IP. Dovrebbe accendersi la luce presente nel modulo di scansione e dovrebbe venire mostrata un'immagine acquisita dalla telecamera nel riquadro di visualizzazione presente al centro del pannello di destra. Alla fine della prova dovrebbe apparire una finestra ausiliaria tramite la quale si chiede di verificare il risultato della prova: rispondere alla domanda premendo il pulsante corrispondente alla risposta nella finestra ausiliaria. Selezionare un'area non troppo scura dell'immagine e quindi premere il pulsante OK sulla finestra ausiliaria: dovrebbe così completarsi l'esecuzione del collaudo. Nel pannello di sinistra dovrebbero essere visibili sulla sinistra gli esiti di tutti i passi di prova e l'esito complessivo della prova. Nell'area messaggi presente in basso nel pannello di destra dovrebbe essere visualizzato un resoconto dell'esecuzione (utilizzare, se necessario, le frecce su e giù per scorrere il testo). Se nel resoconto visualizzato nell'area messaggi per l'utente appare verso la fine la dicitura "Test Plan result: success", questa prova è stata superata con successo.

`PROVA 29 (Esecuzione di un piano di prova)`: Se non si è già in modalità (o vista) di progettazione, passare a tale modalità. Creare un nuovo piano di prova, selezionando l'opzione *New* del menù Test Plan. Sempre tramite il pannello di progettazione del piano di prova, caricare il documento di prova che è stato precedentemente creato durante la `PROVA 28`: per fare questo premere il pulsante "Browse Test Cases" e, tramite la finestra ausiliaria che apparirà, selezionare il documento indicato. A questo punto selezionare la prova caricata nel piano di prova: i passi di cui essa è costituita appariranno nella lista presente nel pannello di sinistra. Selezionare il primo elemento di tale lista (passo di prova del tipo "Send Command") e quindi premere il pulsante Insert per inserire un nuovo passo di prova in cima alla lista. Configurare il nuovo passo di prova

appena inserito in modo che esso generi una attesa (pausa) di 5000 ms (ovvero 5 secondi). Salvare la prova in modo da non perdere le modifiche apportate: selezionare l'opzione *Save* del menù Test Design. Ricaricare la prova modificata nel piano di prova selezionando l'opzione *Reload* del menù Test Plan. Poi passare alla modalità (o vista) di collaudo. Avviare l'esecuzione del piano di prova premendo il pulsante Run del pannello di esecuzione situato sulla sinistra oppure selezionando l'opzione *Run Test Plan*. Subito appena l'esecuzione si è avviata, premere tempestivamente il pulsante Pause per mettere in pausa l'esecuzione del piano di prova. Attendere almeno 5 secondi per verificare che effettivamente l'esecuzione sia nello stato di pausa. Una volta verificata la funzionalità di pausa, premere il pulsante Debug o selezionare l'opzione *Debug Test Plan* del menù Run Test. L'esecuzione dovrebbe avanzare di un passo e poi fermarsi di nuovo ed inoltre, nell'area di visualizzazione dei messaggi per l'utente, dovrebbe venire notificato che l'applicazione è passata alla modalità di esecuzione passo-passo. Eseguire un ulteriore passo della prova premendo nuovamente il pulsante Debug o selezionando nuovamente l'opzione *Debug Test Plan* del menù Run Test: dovrebbe venire eseguito un altro passo della prova. Infine premere il pulsante Run oppure selezionare l'opzione *Run Test Plan* del menù Run Test: l'applicazione dovrebbe uscire dalla modalità di esecuzione passo-passo riprendendo la normale esecuzione fino alla fine del piano di prova. Se l'applicazione si è comportata come previsto ed, in particolare, se è stato possibile attivare la modalità di pausa dell'esecuzione e la modalità di esecuzione passo-passo, allora questa prova è stata superata con successo.

`PROVA 30 (Rimozione di aree selezionate)`: Avviare l'applicazione e passare alla modalità di collaudo. Nell'area di visualizzazione delle immagini presente nel pannello di destra, selezionare una o più aree tramite il dispositivo di puntamento, utilizzando il relativo pulsante sinistro. Poi, una volta che una o più aree sono state selezio-

nate, provare a rimuoverne una o più spostandosi sopra di esse con il dispositivo di puntamento ed attivando il menù contestuale con il pulsante destro. Se è stato possibile selezionare e poi deselezionare una o più aree, questa prova è stata superata con successo.

3.5 Validazione del generatore di resoconto

`PROVA 31 (Generazione del resoconto)`: Nel menù Test Plan (vista progettuale) o Test Execution (vista di collaudo) selezionare l'opzione *New* per creare un nuovo piano di prova. A questo punto, tramite l'opzione *Load* dello stesso menù utilizzato precedentemente, caricare da memoria di massa la documento di prova salvato durante la `PROVA 28`. Poi selezionare tale prova e quindi premere il pulsante "Add Iteration": verrà aggiunta come primo elemento del piano di prova una direttiva di iterazione. Configurare la direttiva di iterazione per effettuare 2 ripetizioni: per fare questo spostarsi con il puntatore sopra la direttiva di iterazione, premere il tasto destro del dispositivo di puntamento e tramite l'opzione "Number of loop iterations" configurare 2 ripetizioni. Salvare il piano di prova su memoria di massa tramite l'opzione *Save* o *Save as...* del menù Test Plan o Test Execution. Infine, se non si è ancora passati alla vista di collaudo effettuare tale passaggio e quindi avviare l'esecuzione del piano di collaudo tramite la pressione del pulsante Run o tramite la selezione dell'opzione *Run Test Plan* dal menù Run Test. Dovrebbe venire ripetuta due volte l'esecuzione della prova e dovrebbe venire creato su memoria di massa un documento avente lo stesso nome del piano di prova ma estensione .log. Ispezionare tale documento di resoconto: esso dovrebbe riportare le stesse informazioni che vengono riportate alla fine dell'esecuzione nell'area riservata ai messaggi per l'utente. Questa prova risulterà superata con successo se è presente su memoria di massa il documento di resoconto, se tale documento risulta esaustivo rispetto a quanto visualizzato nell'area messaggi

dell'interfaccia grafica ed in particolare se questo documento riporta alla fine la dicitura "Test Plan result: success".

Capitolo 4

Conclusioni: obiettivi raggiunti e possibili sviluppi futuri

In questo lavoro di tesi è stato progettato ed implementato un sistema software interattivo finalizzato alla definizione ed all'esecuzione di collaudi di moduli elettronici destinati alla lettura di codici ottici. I vincoli principali che il lavoro doveva soddisfare sono stati elencati nel paragrafo 1.5 ed ora, al termine dello sviluppo e della validazione, è possibile concludere che essi sono stati tutti rispettati.

Le prove definite nel Capitolo 3 sono state concepite per valutare la **correttezza** dell'applicazione in tutti i suoi aspetti ed in tutte le sue funzionalità. Al termine dello sviluppo, esse sono state tutte superate con successo e pertanto si può affermare che l'applicazione soddisfa il vincolo di correttezza.

L'applicazione è anche caratterizzata da un livello di **robustezza** appropriato per i compiti che deve svolgere e per lo scopo che deve raggiungere: questo vincolo è stato soddisfatto grazie alla particolare cura ed attenzione con cui è stata implementata e grazie all'elevato grado di completezza delle prove effettuate per validarne il funzionamento.

L'applicazione risulta soddisfare anche il prefissato vincolo di **usabilità** in quanto è di facile ed intuitivo utilizzo ed inoltre è corredata di ampia documentazione fruibile comodamente tramite l'interfaccia grafica sotto forma di guida d'utente e di indicazioni di

utilizzo.

La **modularità** è stata soddisfatta a due livelli: a livello di architettura software (come evidenziato nel paragrafo 2.2) ed a livello di codice sorgente poichè la computazione è suddivisa tra funzioni riutilizzabili in più parti.

La **manutenibilità** è garantita dalla disponibilità di descrizioni UML della struttura statica e del comportamento dinamico dell'intero sistema software *Scan Engine Test Program*, dall'elevato numero di commenti descrittivi presenti nel codice sorgente, dalla presenza di esaustive descrizioni dei controlli grafici utilizzati (Appendice A) e degli eventi gestiti dall'interfaccia grafica (Appendice B), oltre che dalla struttura modulare del codice sorgente.

L'Appendice C riporta il codice sorgente per la finestra principale dell'applicazione, l'Appendice D riporta il codice sorgente per il modello dei dati, l'Appendice E riporta il codice sorgente per il motore di esecuzione, l'Appendice F riporta infine il codice sorgente per la libreria di acquisizione immagini.

Durante lo svolgimento del progetto sono state acquisite nuove competenze ed ampliate altre già possedute: si è imparato un nuovo linguaggio di programmazione e si è conosciuto un nuovo ambiente di sviluppo software; si è imparato a beneficiare dell'interoperabilità tra il codice gestito dal CLR ed il codice non gestito; si sono migliorate le capacità di sviluppare in autonomia un progetto software e le capacità di produrre una esaustiva documentazione, utilizzando all'occorrenza anche diagrammi UML, per rendere fruibile e manutenibile da parte di altre persone il lavoro svolto.

Questo lavoro presenta anche molteplici possibili sviluppi futuri. Dal punto di vista funzionale, si potrebbe integrare il sistema sviluppato con gli strumenti aziendali di decodifica dei codici ottici. Sempre dal punto di vista funzionale, si potrebbe anche aggiungere un sottosistema di gestione dei difetti, collegato ad una base di dati, che permetta di catalogare e descrivere i difetti di prodotto e le relative risoluzioni applicate. Dal punto di vista applicativo, si po-

trebbe estendere il campo di utilizzo ad altri prodotti sia della stessa azienda sia di altre operanti nell'ambito delle telecomunicazioni o dell'elettronica. Ad esempio, nel campo delle telecomunicazioni si potrebbe reingegnerizzare il prodotto per configurare e controllare un banco di prova per sistemi di telecomunicazioni allo scopo di misurarne vari parametri quali ad esempio il tasso di errore in ricezione a valle della decodifica di canale e poter quindi validare un tale sistema. Da un punto di vista architetturale invece, si potrebbe portare l'applicazione ad altri sistemi operativi.

Un ulteriore ambito di sviluppo a più breve termine e di minore entità è dato dalla possibilità di apportare alcuni miglioramenti che sono stati già individuati e catalogati nella documentazione che accompagna l'applicazione. Ad esempio, è possibile nella maggioranza dei casi evitare di utilizzare finestre ausiliarie del tipo MessageBox ed inglobare invece tutta la comunicazione con l'utente all'interno della finestra principale utilizzando un'ulteriore area dedicata a questo scopo e già disponibile sopra o sotto il riquadro di visualizzazione immagini nella vista di collaudo. Una seconda opportunità di miglioramento è data dalla possibilità di introdurre un meccanismo di selezione dell'interfaccia I^2C qualora siano presenti più interfacce collegate allo stesso calcolatore elettronico (al momento viene scelta la prima disponibile). Dal punto di vista della robustezza, sarebbe necessario modificare la prima delle due librerie di incapsulamento (*device.dll*) in modo che, qualora mancasse nel sistema la libreria specifica dell'interfaccia I^2C, venga restituito un apposito codice di errore all'applicazione la quale dovrà poi provvedere ad informare l'utente. Una utile funzionalità che potrebbe essere introdotta con minimo costo è la determinazione del tempo che intercorre tra guasti nelle prove iterate da cui poter poi estrapolare un valore del tempo medio che intercorre tra i guasti dei componenti elettronici (anche indicato spesso in letteratura come Mean Time Between Failure o, in breve, MTBF). Infine si potrebbe migliorare il metodo con cui vengono aggiornati i controlli ListView facendo in modo che un

evento venga generato ogni volta che il modello dei dati cambia e facendo in modo che, a quel punto, il gestore di tale evento inneschi l'aggiornamento delle liste che contengono i passi di prova o le prove.

Il vantaggio di maggior rilievo è che, mediante l'utilizzo del sistema interattivo di collaudo sviluppato, si potrebbe aumentare notevolmente sia l'affidabilità che la produttività dei processi di collaudo tramite la razionalizzazione dei metodi di definizione, gestione ed esecuzione delle prove. Infatti, da una parte la crescita di affidabilità permetterà di minimizzare gli oneri di garanzia, dall'altra l'aumento di produttività del processo di collaudo permetterà di ridurre i costi di collaudo tramite una riduzione dei costi del personale.

Tuttavia è importante notare che il raggiungimento di adeguati livelli di affidabilità all'interno di una azienda dipende anche dalla presenza di una unità organizzativa, di complessità commisurata, di specialisti dell'affidabilità direttamente responsabili dell'esecuzione di alcuni compiti e del coordinamento di altri. Una tale unità organizzativa dovrebbe occuparsi globalmente dei problemi aziendali della qualità in senso lato [12].

Ancora più importante da notare è il fatto che miglioramenti sostanziali possono essere ottenuti solo mediante l'attuazione di un programma organico a maglia chiusa di azioni correttive e revisioni di progetto del modulo elettronico di lettura dei codici ottici. Il programma di azioni correttive dovrà tendere ad eliminare i guasti sistematici imputabili a debolezze del progetto. Quando il numero di azioni correttive da effettuare sul modulo elettronico sarà descresciuto sensibilmente e quando la frequenza di guasto sarà praticamente costante nel tempo, si potrà giudicare che i guasti sistematici saranno stati praticamente eliminati. A quel punto ogni ulteriore aumento di affidabilità potrà essere ottenuto solo con modifiche sostanziali del progetto [12].

Il processo di collaudo dovrebbe soddisfare almeno tre obiettivi principali: il collaudo durante lo sviluppo, il collaudo pre-produzione

ed il collaudo in fase di produzione. Nel caso del collaudo durante la fase di sviluppo, si dovrà mirare a realizzare apparati che soddisfino rigorosamente le specifiche ambientali e per i quali venga ottenuta una elevata affidabilità intrinseca prima di essere posti in produzione. Nel caso della fase di collaudo pre-produzione, sarà necessario assicurare che la fase di transizione dallo sviluppo alla produzione non causi debolezze potenziali. Infine, nel terzo ed ultimo caso, si dovrà puntare a conservare il livello di affidabilità durante tutta la produzione. Di fondamentale utilità per tutte le fasi di collaudo, risulterà la possibilità di applicare un fattore di accelerazione dei guasti.

Seppure un sistema interattivo di collaudo come quello sviluppato in questo progetto può contribuire in maniera significativa alla crescita di affidabilità ed alla riduzione dei costi di collaudo, molteplici altri fattori sono implicati nel raggiungimento di tali risultati e, data la loro importanza, si ritiene utile menzionarli a conclusione: l'esperienza del personale operativo e di manutenzione, le finalità e l'efficienza del programma di affidabilità, l'efficacia e l'efficienza del programma di prova e la capacità del controllo della produzione di ridurre l'incidenza dei difetti introdotti da componenti, fabbricazione e assiemamento. In particolare, l'efficacia e l'efficienza del programma di prova saranno determinate dal tempestivo inizio delle prove, dal numero degli apparati in prova, dal numero di prove effettuate, dall'accuratezza dell'analisi dei guasti e conseguentemente dall'efficienza delle azioni correttive.

Appendice A

Elenco dei controlli grafici utilizzati

La finestra principale (MainForm) è dotato dei seguenti controlli grafici:

- MenuStrip
  - 3 ToolStripMenuItem:
    - Config
      - ToolStripMenuItem menuI2CHostAdapterCfg (“I2C Host Adapter”)
        - ToolStripMenuItem “I2C Default Address”
          - ToolStripTextBox
        - ToolStripMenuItem “Bitrate”
          - ToolStripTextBox
      - ToolStripMenuItem toolStripMenuFrameGrabberResolution (“Frame grabber resolution”)
        - ToolStripMenuItem toolStripMenuFrameGrabberWidth
          - ToolStripTextBox
        - ToolStripMenuItem toolStripMenuFrameGrabberHeight
          - ToolStripTextBox
      - ToolStripMenuItem menuFrameGrabberSelect (“Frame grabber selection”)
    - Help
    - Exit (posizionato dopo ToolStripComboBox e prima di LinkLabel)
  - ToolStripComboBox per la selezione della vista (vedi sotto)
  - LinkLabel per il collegamento web al sito Datalogic
- Tre pannelli principali (Design, Plan ed Execute) separati da due Splitter

- StatusStrip in basso (inutilizzata al momento).

Alle due possibili viste della finestra principale (selezionabili tramite ToolStripComboBox) corrispondono:

- due pannelli visibili in vista Designer: Design, Plan;
- due pannelli visibili in vista Tester: Plan, Execute.

Controlli grafici del pannello Design (presente solo in vista Designer):

- ToolStripMenuItem
  - New
  - Load
  - Save (abilitato solo se la prova è stata modificata)
  - Save as... (abilitato solo se la prova contiene elementi)
- 3 ToolStripButton ed un ContextMenuStrip associato per le azioni:
  - Remove (disabilitato automaticamente in ListView vuota)
  - Insert (disabilitato automaticamente in ListView vuota)
  - Append
- TextBox (con annessa Label) per indicare il nome della prova
- ListView per elencare i passi (test steps) della prova
- Sottopannello per la configurazione dell'operazione associata a ciascun passo della prova:
  - Sleep
  - Send Message
  - Load Image
  - Capture Image
  - Analyze Image
  - User Message
  - User Feedback

Controlli grafici del pannello Plan (presente sia in vista Designer che in vista Tester):

- ToolStripMenuItem:
  - New
  - Load
  - Reload (abilitato solo se è caricato un piano di prova o se ci sono prove nel piano che viene progettato)
  - Save (abilitato solo se il piano di prova è stato modificato)
  - Save as... (abilitato solo se il piano di prova contiene elementi)
- 5 ToolStripButton ed un ContextMenuStrip associato per le azioni:
  - Browse (solo come ToolStripButton)
  - Add Iteration (solo come ToolStripButton)
  - LoopIterations (solo come ContextMenuStrip e solo se è selezionato un TestIterator)
    - ToolStripTextBox
  - LoopSize (solo come ContextMenuStrip e solo se è selezionato un TestIterator)
    - ToolStripTextBox
  - Remove (disabilitato automaticamente in ListView vuota)
  - Move Up (disabilitato automaticamente in ListView con meno di due elementi)
  - Move Down (disabilitato automaticamente in ListView con meno di due elementi)
  - Move to Top (solo come ContextMenuStrip)
  - Move to Bottom (solo come ContextMenuStrip)
- TextBox (con annessa Label) per indicare il nome del piano di prova
- SplitContainer
  - ListView per elencare le prove (test cases) del piano di prova
  - ListView per elencare i passi della prova selezionata (visibile solo in vista Tester)
  - TextBox per la visualizzazione di messaggi di servizio (visibile solo in vista Designer)

Controlli grafici del pannello Execute (presente solo in vista Tester):

- ToolStripMenuItem:
  - Debug Test Plan
  - Run Test Plan
  - Stop Execution
  - Generate Report
    - ToolStripComboBox: Disabled, Enabled
  - On Fail...
    - ToolStripComboBox: Ask, Break, Continue
- 4 ToolStripButton:
  - Run
  - Pause
  - Stop
  - Debug
- 2 SplitContainer con 2 SplitterPanel ciascuno per un totale di 4 sottopannelli
  - PictureBox per il secondo sottopannello
    - ContextMenuStrip (con 1 elemento che serve a rimuovere la regione dell'immagine selezionata)

  • TextBox per il quarto sottopannello (visualizzazione dei messaggi di servizio)

Appendice B

Elenco degli eventi

B.1 Finestra (form) principale

evento MainForm_FormClosing

B.2 MenuStrip della finestra principale

ToolStripMenuItem exitMenu:

evento exitMenu_Click

ToolStripMenuItem configToolStripMenuItem

ToolStripMenuItem menuI2CHostAdapterCfg

ToolStripMenuItem toolStripMenuI2CDefaultAddress

ToolStripTextBox toolStripTextBoxI2CDefaultAddress:

evento toolStripTextBoxI2CDefaultAddress_TextChanged

ToolStripMenuItem toolStripMenuBitrate

ToolStripTextBox toolStripTextBoxBitrate:

evento toolStripTextBoxBitrate_TextChanged

ToolStripMenuItem toolStripMenuFrameGrabberResolution

ToolStripMenuItem toolStripMenuFrameGrabberWidth

ToolStripTextBox toolStripTextBoxFGWidth:

evento toolStripTextBoxFGWidth_TextChanged

ToolStripMenuItem toolStripMenuFrameGrabberHeight

ToolStripTextBox toolStripTextBoxFGHeight:

evento toolStripTextBoxFGHeight_TextChanged

ToolStripMenuItem menuFrameGrabberSelect:

evento menuFrameGrabberSelect_Click

ToolStripMenuItem helpMenu

ToolStripMenuItem aboutMenu:

evento aboutMenu_Click

ToolStripMenuItem helpFileMenu:

evento helpFileMenu_Click

B.3 Pannello Design (progetto della prova)

evento panelDesign_Resize

ToolStripMenuItem testDesignMenu:

evento testDesignMenu_Click

ToolStripMenuItem newTestMenu:

evento newTestMenu_Click

ToolStripMenuItem loadTestMenu_Click:

evento loadTestMenu_Click

ToolStripMenuItem saveTestMenu_Click:

evento saveTestMenu_Click

ToolStripMenuItem saveAsTestMenu_Click:

evento saveAsTestMenu_Click

ToolStripButton buttonRemoveStep:

evento buttonRemoveStep_Click

ToolStripButton buttonInsertStep:

evento buttonInsertStep_Click

ToolStripButton buttonAppendStep:

evento buttonAppendStep_Click

TextBox textTestCaseName (con annessa Label):

evento textTestCaseName_TextChanged

ListView listViewTestSteps:

evento listViewTestSteps_KeyUp
evento listViewTestSteps_MouseClick

ContextMenuStrip contextMenuStepList:

evento contextMenuStepList_Opening
ToolStripMenuItem removeStepMenu:
evento buttonRemoveStep_Click
ToolStripMenuItem insertStepMenu:
evento buttonInsertStep_Click
ToolStripMenuItem appendStepMenu:
evento buttonAppendStep_Click

B.3.1 Sottopannello "Sleep"

TrackBar trackBarSleep:

evento trackBarSleep_ValueChanged

TextBox textBoxSleepDuration:

evento textBoxSleepDuration_TextChanged

B.3.2 Sottopannello "Send Command"

TextBox textSendCommandDescription:

evento textSendCommandDescription_TextChanged

ComboBox comboBoxSendCommand:

evento comboBoxSendCommand_SelectedIndexChanged

TextBox textCommandParameters:

evento textCommandParameters_TextChanged

ComboBox comboBoxExpectedAnswer

evento comboBoxExpectedAnswer_SelectedIndexChanged

TextBox textI2CAddress:

evento textI2CAddress_TextChanged

B.3.3 Sottopannello "Load Image"

TextBox textLoadImageDescription:

evento textLoadImageDescription_TextChanged

TextBox textLoadImageFileName:

evento textLoadImageFileName_TextChanged

Button buttonBrowseFileNameLoadImage:

evento buttonBrowseFileNameLoadImage_Click

B.3.4 Sottopannello "Capture Image"

TextBox textCaptureImageDescription:

evento textCaptureImageDescription_TextChanged

Checkbox checkSaveImage:

evento checkSaveImage_CheckedChanged

TextBox textSaveImageFileName:

evento textSaveImageFileName_TextChanged

TextBox textCurrentSaveImageFolder:

evento textCurrentSaveImageFolder_TextChanged

Button buttonBrowseImageSaveFolder:

evento buttonBrowseImageSaveFolder_Click

NumericUpDown numericUpDownCaptureImageNumberOfFrames

evento numericUpDownCaptureImageNumberOfFrames_ValueChanged

B.3.5 Sottopannello "Analyze Image"

TextBox textAnalizeImageDescription:

evento textAnalizeImageDescription_TextChanged

ComboBox comboImageResolution:

evento comboImageResolution_SelectedIndexChanged

Label labelMajorMinor:

evento labelMajorMinor_Click

NumericUpDown numericValueTarget:

evento numericValueTarget_ValueChanged

B.3.6 Sottopannello "User Message"

TextBox textUserMessageDescription:

evento textUserMessageDescription_TextChanged

TextBox textUserMessage:

evento textUserMessage_TextChanged

ToolStripButton buttonBrowseImage:

evento buttonBrowseImage_Click

B.3.7 Sottopannello "User Feedback"

TextBox textUserFeedbackDescription:

evento textUserFeedbackDescription_TextChanged

RadioButton radioUserFeedbackYes:

evento radioUserFeedbackYes_CheckedChanged

RadioButton radioUserFeedbackNo:

evento radioUserFeedbackNo_CheckedChanged

TextBox textUserFeedbackMessage:

evento textUserFeedbackMessage_TextChanged

B.4 Pannello Plan (progettazione del piano di prova)

evento panelPlan_Resize

ToolStripMenuItem testPlanMenu:

evento testPlanMenu_Click

ToolStripMenuItem newPlanMenu:

evento newPlanMenu_Click

ToolStripMenuItem loadPlanMenu:

evento loadPlanMenu_Click

ToolStripMenuItem reloadPlanMenu_Click:

evento reloadPlanMenu_Click

ToolStripMenuItem savePlanMenu:

evento savePlanMenu_Click

ToolStripMenuItem saveAsPlanMenu:

evento saveAsPlanMenu_Click

TextBox textTestPlanName (con annessa Label):

evento textTestPlanName_TextChanged

ListView listViewTestPlan:

evento listViewTestPlan_KeyUp

evento listViewTestPlan_MouseClick

evento listViewTestPlan_SelectedIndexChanged

ContextMenuStrip contextMenuTestPlan:

evento contextMenuTestPlan_Opening

ToolStripMenuItem menuConfigLoopIterations

ToolStripTextBox toolStripTextBoxLoopIterations:

evento toolStripTextBoxLoopIterations_TextChanged

ToolStripMenuItem menuConfigLoopSize

ToolStripTextBox toolStripTextBoxLoopSize:

evento toolStripTextBoxLoopSize_TextChanged

ToolStripMenuItem removeTestMenu:

evento buttonRemoveTest_Click

ToolStripMenuItem moveUpTestMenu:

evento moveUpTestMenu_Click

ToolStripMenuItem moveDownTestMenu:

evento moveDownTestMenu_Click

ToolStripMenuItem moveToTopTestMenu:

evento moveToTopTestMenu_Click

ToolStripMenuItem moveToBottomTestMenu:

evento moveToBottomTestMenu_Click

TextBox textConsoleForPlan (solo in vista Designer)

ListView listViewTestStepsForPlan (solo in vista Tester)

B.5 Pannello Execute (esecuzione del piano di prova)

ToolStripMenuItem testExecutionMenu

ToolStripMenuItem debugTestMenu:

evento buttonDebugExecution_Click

ToolStripMenuItem runTestPlanMenu:

evento buttonRunExecution_Click

ToolStripMenuItem breakExecutionMenu:

evento buttonStopExecution_Click

ToolStripMenuItem generateReportMenu

ToolStripComboBox generateReportCombo:

evento generateReportCombo_SelectedIndexChanged

ToolStripMenuItem onFailMenu

ToolStripComboBox onFailCombo:

evento onFailCombo_SelectedIndexChanged

ToolStripButton buttonRunExecution:

evento buttonRunExecution_Click

ToolStripButton buttonPauseExecution:

evento buttonPauseExecution_Click

ToolStripButton buttonStopExecution:

evento buttonStopExecution_Click

ToolStripButton buttonDebugExecution:

evento buttonDebugExecution_Click

TextBox textTesterName (con annessa Label):

evento textTesterName_TextChanged

SplitContainer SplitContainer3

SplitterPanel Panel1

SplitContainer SplitContainer4

SplitterPanel Panel1

SplitterPanel Panel2

PictureBox pictureBox1:

evento pictureBox1_BackgroundImageChanged

evento pictureBox1_MouseDown

evento pictureBox1_MouseMove

evento pictureBox1_MouseUp

evento pictureBox1_Paint

evento pictureBox1_Resize

ContextMenuStrip contextMenuPictureBox

ToolStripMenuItem toolStripMenuRemoveSelection:

evento toolStripMenuRemoveSelection_Click

SplitterPanel Panel2

SplitContainer SplitContainer5

SplitterPanel Panel1

SplitterPanel Panel2

TextBox textConsoleForExecute

B.6 Icona nell'area di notifica (barra delle applicazioni di Windows®)

NotifyIcon notifyIcon1

ContextMenuStrip iconContextMenuStrip

ToolStripMenuItem aboutIconMenuItem:
evento aboutMenu_Click
ToolStripMenuItem exitIconMenuItem:
evento exitMenu_Click

B.7 Eventi associati alle strutture dati

evento TestStep_ResultChanged

evento TestCase_ResultChanged

evento Framegrabber_ImageChanged

Appendice C

Codice sorgente per la classe MainForm

```
1  ï»¿/*
2   * Scan Engine Test Program
3   *
4   * developed by Guido Trentalancia for Datalogic S.p.A.
5   *
6   */
7
8  using System;
9  using System.Collections.Generic;
10 using System.ComponentModel;
11 using System.Drawing;
12 using System.Globalization;
13 using System.Reflection;
14 using System.Threading;
15 using System.Windows.Forms;
16
17 #if (FRAMEGRABBER)
18 using FramegrabberLibrary;
19 #endif
20
21 namespace ScanEngineTestProgram
22 {
23     public partial class MainForm : Form
24     {
25         const string I2C_DEFAULT_ADDRESS = "5C";
```

```
26          // I2C bitrate for version 3.x Aardvark hardware should be within 1kHz
                and 800kHz.
27          // The default power-on bitrate is 100kHz. Only certain discrete
                bitrates are
28          // supported by the Aardvark. As such, the actual bitrate set will be less
                than
29          // or equal to the requested bitrate.
30          //
31          // The Atmel microcontroller only officially supports 100kHz and 400
                kHz.
32          const string I2C_DEFAULT_BITRATE = "400";
33          public const int MIN_I2C_BITRATE = 1; // kHz
34          public const int MAX_I2C_BITRATE = 400; // kHz, The Atmel
                microcontroller only supports 100kHz and 400kHz
35
36          bool m_DesignerView = true;
37          TestCase m_TestCase = new TestCase();
38          TestPlan m_TestPlan = new TestPlan();
39          ExecutionThread workerObject;
40          int m_CurrentStepIndex = -1;
41          int m_CurrentPlanIndex = -1;
42          bool iterator_selected_in_plan = false;
43          ushort I2CAddress;
44          public enum report_level { Disabled = 0, Enabled };
45          report_level ReportLevel;
46          string TesterName = null;
47          public enum on_failure { Ask = 0, Break, Continue };
48          on_failure OnFailure;
49
50          Bitmap m_BackImage = null;
51          Bitmap m_SecondaryImage = null;
52          Bitmap m_ForeImage = null;
53          Graphics m_PictureGraphics = null;
54          Pen selectionPen = new Pen(Color.Blue, 1);
55          Rectangle imageSelectionRect;
56          List<Rectangle> selectionList = null;
57          List<Rectangle> scaledselectionList = null;
58          int m_CurrentSelectedRegion = -1;
59          ImageAnalyzer m_ImageAnalyzer = new ImageAnalyzer();
60
```

```
61          int BitmapSelectionXStart, BitmapSelectionYStart; // The top left
                corner of the picturebox selection
62
63  #if (FRAMEGRABBER)
64          Framegrabber Framegrabber = new Framegrabber();
65
66          public const long DefaultFrameGrabberWidth = Framegrabber.
                DefaultFrameGrabberWidth;
67          public const long DefaultFrameGrabberHeight = Framegrabber.
                DefaultFrameGrabberHeight;
68
69          public const int DefaultFrameGrabberPixelFormat = Framegrabber.
                DefaultFrameGrabberPixelFormat; // Mode8=17301505
70          public const long DefaultFrameGrabberAcquisitionMode =
                Framegrabber.DefaultFrameGrabberAcquisitionMode; //
                AcquisitionMode=Continuous -> 0
71  #endif
72
73  #if (FRAMEGRABBER)
74          long FrameGrabberWidth = DefaultFrameGrabberWidth;
75          long FrameGrabberHeight = DefaultFrameGrabberHeight;
76  #else
77          long FrameGrabberWidth = 752;
78          long FrameGrabberHeight = 480;
79  #endif
80
81          bool m_GuiLocked = false;
82
83          // This delegate enables asynchronous calls for a generic event
84          delegate void GenericEventCallback(object sender, EventArgs e);
85
86          // This delegate will call the thread stop method asynchronously (used
                within buttonStopExecution_Click())
87          delegate void AsyncRequestThreadStopCaller();
88
89          // This delegate will clear all selected areas from the picturebox in
                panelExecute (and refresh the picturebox)
90          delegate void ClearImageSelectionFromPictureBoxDelegate();
91
92          // This delegate will append text to the Console (textBoxConsole)
93          public delegate void ConsoleAppendTextDelegate(string text);
```

```
94
95          public MainForm()
96          {
97              InitializeComponent();
98
99              // Initialize the background image for the picturebox in
                    panelExecute
100             SetBackgroundImage(null);
101
102             // Initialize the foreground image for the picturebox in
                    panelExecute
103             m_ForeImage = new Bitmap(pictureBox1.Width, pictureBox1.
                    Height, System.Drawing.Imaging.PixelFormat.
                    Format32bppArgb);
104
105             if (m_BackImage != null)
106             {
107                 pictureBox1.BackgroundImage = (System.Drawing.Image)
                        m_BackImage.Clone();
108             }
109             else
110             {
111                 pictureBox1.BackgroundImage = null;
112             }
113
114             pictureBox1.Image = m_ForeImage;
115
116             m_PictureGraphics = Graphics.FromImage(m_ForeImage);
117
118             pictureBox1.Refresh();
119
120             // Create an object to store the regions of interest from the
                    background image in the picturebox
121             selectionList = new List<Rectangle>();
122
123             // Create an object to store the regions of interest from the
                    background image in the picturebox
124             // scaled accordingly to the zoom applied by
                    BackgroundImageLayout = ImageLayout.Zoom
125             scaledselectionList = new List<Rectangle>();
126
```

```
127             // Load the settings
128             Screen myScreen = Screen.FromControl(this);
129             Rectangle screen_area = myScreen.WorkingArea;
130             Point top_left_corner = new Point(0, 0);
131             if (Properties.Settings.Default.Location.X >= 0 && Properties.
                    Settings.Default.Location.X <= screen_area.Width &&
                    Properties.Settings.Default.Location.Y >= 0 && Properties.
                    Settings.Default.Location.Y <= screen_area.Height)
132                 this.Location = Properties.Settings.Default.Location;
133             else
134                 this.Location = top_left_corner;
135             if (Properties.Settings.Default.Width >= 132 && Properties.
                    Settings.Default.Width <= screen_area.Width)
136                 this.Width = Properties.Settings.Default.Width;
137             else
138             {
139                 this.Location = top_left_corner;
140                 this.Width = screen_area.Width;
141             }
142             if (Properties.Settings.Default.Height >= 38 && Properties.
                    Settings.Default.Height <= screen_area.Height)
143                 this.Height = Properties.Settings.Default.Height;
144             else
145             {
146                 this.Location = top_left_corner;
147                 this.Height = screen_area.Height;
148             }
149
150             // Print the version
151             ConsoleAppendText("SETP␣version␣" + Assembly.
                    GetExecutingAssembly().GetName().Version + "\r\n");
152             ConsoleAppendText("mscorlib.dll␣version␣" + typeof(String).
                    Assembly.GetName().Version + "\r\n");
153             ConsoleAppendText("Developed␣by␣Guido␣Trentalancia␣for␣" +
                     this.CompanyName + ".\r\n\r\n");
154
155             if (Properties.Settings.Default.I2CAddress.Length != 2)
156             {
157                 MessageBox.Show("Invalid␣hexadecimal␣number␣for␣the␣
                        I2C␣Default␣Address␣!␣Using␣default␣value␣" +
                        I2C_DEFAULT_ADDRESS + ".");
```

```
158             Properties.Settings.Default.I2CAddress =
                    I2C_DEFAULT_ADDRESS;
159             Properties.Settings.Default.Save(); // Recover from an hard-
                    coded value.
160         }
161
162         try
163         {
164             I2CAddress = UInt16.Parse(Properties.Settings.Default.
                    I2CAddress, NumberStyles.AllowHexSpecifier);
165         }
166         catch (FormatException)
167         {
168             MessageBox.Show("Invalid␣hexadecimal␣number␣for␣the␣
                    I2C␣Default␣Address␣!␣Using␣default␣value␣" +
                    I2C_DEFAULT_ADDRESS + ".");
169             I2CAddress = UInt16.Parse(I2C_DEFAULT_ADDRESS,
                    NumberStyles.AllowHexSpecifier);
170             Properties.Settings.Default.I2CAddress =
                    I2C_DEFAULT_ADDRESS;
171             Properties.Settings.Default.Save(); // Recover from an hard-
                    coded value.
172         }
173
174         string i2CDefaultAddress = Properties.Settings.Default.
                I2CAddress;
175         SendCommand.SetI2CDefaultAddress(i2CDefaultAddress);
176         toolStripTextBoxI2CDefaultAddress.Text = i2CDefaultAddress;
177
178         if (Properties.Settings.Default.I2CBitrate < MIN_I2C_BITRATE ||
                Properties.Settings.Default.I2CBitrate > 400)
179         {
180             MessageBox.Show("Invalid␣value␣for␣the␣I2C␣Default␣
                    Bitrate␣!␣Using␣default␣value␣" +
                    I2C_DEFAULT_BITRATE + ".");
181             Properties.Settings.Default.I2CBitrate = UInt16.Parse(
                    I2C_DEFAULT_BITRATE, NumberStyles.None);
182             Properties.Settings.Default.Save(); // Recover from an hard-
                    coded value.
183         }
184
```

```
185             toolStripTextBoxBitrate.Text = Properties.Settings.Default.
                    I2CBitrate.ToString();
186
187             // Adjust the width for the listview listViewTestSteps columns
188             if (listViewTestSteps.Columns.Count > 0)
189                 listViewTestSteps.Columns[0].Width = 50;
190             if (panelDesign.Width − 55 > 0)
191                 if (listViewTestSteps.Columns.Count > 1)
192                     listViewTestSteps.Columns[1].Width = panelDesign.
                            Width − 55;
193
194             // Adjust the width for the listview listViewTestPlan columns
195             if (listViewTestPlan.Columns.Count > 0)
196                 listViewTestPlan.Columns[0].Width = panelPlan.Width;
197
198             // Load initial selection of combo boxes
199             viewCombo.SelectedIndex = 0;
200             generateReportCombo.SelectedIndex = 1; // Enabled
201             onFailCombo.SelectedIndex = 2; // Continue on failure
202
203             // Disable unused buttons in panelExecute
204             buttonYes.Visible = false;
205             buttonNo.Visible = false;
206             buttonContinue.Visible = false;
207             buttonBreak.Visible = false;
208
209             // Set Fill Dock to all panels
210             panelStepAnalyzeImage.Dock = DockStyle.Fill;
211             panelStepCaptureImage.Dock = DockStyle.Fill;
212             panelStepLoadImage.Dock = DockStyle.Fill;
213             panelStepUserMessage.Dock = DockStyle.Fill;
214             panelStepSendCommand.Dock = DockStyle.Fill;
215             panelStepSleep.Dock = DockStyle.Fill;
216             panelStepUserFeedback.Dock = DockStyle.Fill;
217
218             // Hide unused buttons and contextual menus
219             ShowPlanDetails();
220
221             // Initialize Dialog Boxes
222             openDialogImage.CheckFileExists = true;
223             openDialogImage.Multiselect = false;
```

```
224                 // openDialogImage.InitialDirectory = m_InitialDirectory;
225                 openDialogImage.RestoreDirectory = true;
226                 openDialogImage.Filter = "Image␣Files␣(*.bmp;*.png;*.jpg)|*.
                        bmp;*.png;*.jpg";
227                 openDialogImage.FilterIndex = 1;
228                 openDialogImage.Title = "Open␣Image␣File";
229
230 #if (FRAMEGRABBER)
231                 // Enable the framegrabber options in the Config menu
232                 toolStripSeparatorConfigMenu.Visible = true;
233                 toolStripMenuFrameGrabberResolution.Visible = true;
234                 menuFrameGrabberSelect.Visible = true;
235
236                 // Set the default framegrabber resolution in the two
                        toolstriptextboxes (Config menu)
237                 toolStripTextBoxFGWidth.Text = FrameGrabberWidth.ToString();
238                 toolStripTextBoxFGHeight.Text = FrameGrabberHeight.ToString()
                        ;
239 #else
240                 // Disable the framegrabber options in the Config menu
241                 toolStripSeparatorConfigMenu.Visible = false;
242                 toolStripMenuFrameGrabberResolution.Visible = false;
243                 menuFrameGrabberSelect.Visible = false;
244 #endif
245
246 #if (FRAMEGRABBER)
247                 // Add an event handler to the framegrabber object that changes the
                        background image
248                 // in the picturebox in panelExecute
249                 Framegrabber.ImageChanged += new FramegrabberLibrary.
                        ImageChangedEventHandler(Framegrabber_ImageChanged);
250 #endif
251             }
252
253             private void viewCombo_SelectedIndexChanged(object sender,
                    EventArgs e)
254             {
255                 if (m_GuiLocked)
256                     return;
257
258                 bool m_DesignerView_old = m_DesignerView;
```

```
259             m_DesignerView = (viewCombo.SelectedIndex == 0);
260
261             if (m_DesignerView == m_DesignerView_old)
262                 return;
263
264             UpdateView();
265             ShowTestCase();
266             ShowStepDetails();
267
268             if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    listViewTestPlan.Items.Count)
269             {
270                 if (listViewTestPlan.Items[m_CurrentPlanIndex].Selected ==
                        false)
271                     listViewTestPlan.Items[m_CurrentPlanIndex].Selected =
                            true;
272                 listViewTestPlan.Items[m_CurrentPlanIndex].Focused = true;
273                 listViewTestPlan.Focus();
274             }
275         }
276
277         void UpdateView()
278         {
279             m_GuiLocked = true;
280
281             panelDesign.Visible = m_DesignerView;
282
283             panelExecute.Visible = !m_DesignerView;
284
285             // Select the listview style for different views and configure the state
                    image list
286             if (m_DesignerView)
287             {
288                 listViewTestPlan.View = View.List;
289                 listViewTestPlan.StateImageList = null;
290             }
291             else
292             {
293                 listViewTestPlan.View = View.Details;
294                 listViewTestPlan.StateImageList = imageListResult;
295             }
```

```
296
297             textConsoleForPlan.Visible = m_DesignerView;
298             listViewTestStepsForPlan.Visible = !m_DesignerView;
299
300             // Change the menu name
301             if (m_DesignerView)
302                 testPlanMenu.Text = "Test␣Plan";
303             else
304                 testPlanMenu.Text = "Test␣Execution";
305
306             // Disable the context menu in the test case listview
307             contextMenuStepList.Enabled = m_DesignerView;
308
309             m_GuiLocked = false;
310         }
311
312         private void ShowStepDetails()
313         {
314             m_GuiLocked = true;
315
316             bool stepSelected = (m_CurrentStepIndex >= 0 &&
                    m_CurrentStepIndex < listViewTestSteps.Items.Count);
317             bool needtosavetestcase_backup = m_TestCase.NeedToSave;
318
319             // Enable or disable the buttons according to a step being selected
320             buttonInsertStep.Enabled = stepSelected;
321             buttonRemoveStep.Enabled = stepSelected;
322
323             // Enable or disable the step type combobox according to a step
                    being selected
324             comboStepType.Enabled = stepSelected;
325
326             if (stepSelected)
327             {
328                 // Select the combo box type and show populate the panel
329                 string steptype = m_TestCase[m_CurrentStepIndex].GetType
                        ().Name;
330                 if (steptype == "Sleep")
331                 {
332                     comboStepType.SelectedIndex = 0;
333                     ShowSleep((Sleep)m_TestCase[m_CurrentStepIndex]);
```

```
334                 }
335                 else if (steptype == "SendCommand")
336                 {
337                     comboStepType.SelectedIndex = 1;
338                     ShowSendCommand((SendCommand)m_TestCase[
                            m_CurrentStepIndex]);
339                 }
340                 else if (steptype == "LoadImage")
341                 {
342                     comboStepType.SelectedIndex = 2;
343                     ShowLoadImage((LoadImage)m_TestCase[
                            m_CurrentStepIndex]);
344                 }
345                 else if (steptype == "CaptureImage")
346                 {
347                     comboStepType.SelectedIndex = 3;
348                     ShowCaptureImage((CaptureImage)m_TestCase[
                            m_CurrentStepIndex]);
349                 }
350                 else if (steptype == "AnalyzeImage")
351                 {
352                     comboStepType.SelectedIndex = 4;
353                     ShowAnalyzeImage((AnalyzeImage)m_TestCase[
                            m_CurrentStepIndex]);
354                 }
355                 else if (steptype == "UserMessage")
356                 {
357                     comboStepType.SelectedIndex = 5;
358                     ShowUserMessage((UserMessage)m_TestCase[
                            m_CurrentStepIndex]);
359                 }
360                 else if (steptype == "UserFeedback")
361                 {
362                     comboStepType.SelectedIndex = 6;
363                     ShowUserFeedback((UserFeedback)m_TestCase[
                            m_CurrentStepIndex]);
364                 }
365
366                 // Restore the save test case menu enabled/disabled state
367                 m_TestCase.NeedToSave = needtosavetestcase_backup;
368
```

```
369                 // Show the correct operation panel
370                 comboStepType.Visible = true;
371                 panelStepSleep.Visible = ((string)comboStepType.
                        SelectedItem == "Sleep");
372                 panelStepSendCommand.Visible = ((string)comboStepType.
                        SelectedItem == "Send␣Command");
373                 panelStepLoadImage.Visible = ((string)comboStepType.
                        SelectedItem == "Load␣Image");
374                 panelStepCaptureImage.Visible = ((string)comboStepType.
                        SelectedItem == "Capture␣Image");
375                 panelStepAnalyzeImage.Visible = ((string)comboStepType.
                        SelectedItem == "Analyze␣Image");
376                 panelStepUserMessage.Visible = ((string)comboStepType.
                        SelectedItem == "User␣Message");
377                 panelStepUserFeedback.Visible = ((string)comboStepType.
                        SelectedItem == "User␣Feedback");
378             }
379             else
380             {
381                 comboStepType.SelectedIndex = -1;
382                 panelStepSleep.Visible = false;
383                 panelStepSendCommand.Visible = false;
384                 panelStepLoadImage.Visible = false;
385                 panelStepCaptureImage.Visible = false;
386                 panelStepAnalyzeImage.Visible = false;
387                 panelStepUserMessage.Visible = false;
388                 panelStepUserFeedback.Visible = false;
389             }
390
391             m_GuiLocked = false;
392         }
393
394         private void comboStepType_SelectedIndexChanged(object sender,
                EventArgs e)
395         {
396             if (m_GuiLocked)
397                 return;
398
399             if (m_CurrentStepIndex < 0 || m_CurrentStepIndex >=
                    listViewTestSteps.Items.Count)
400             {
```

```
401                 return;
402             }
403
404             TestStep currentStep = null;
405             if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    listViewTestSteps.Items.Count)
406             {
407                 currentStep = (TestStep)m_TestCase[m_CurrentStepIndex];
408                 if ((currentStep.GetType().Name == "Sleep") && (
                        comboStepType.SelectedIndex == 0))
409                 {
410                     return;
411                 }
412                 if ((currentStep.GetType().Name == "SendCommand") && (
                        comboStepType.SelectedIndex == 1))
413                 {
414                     return;
415                 }
416                 if ((currentStep.GetType().Name == "LoadImage") && (
                        comboStepType.SelectedIndex == 2))
417                 {
418                     return;
419                 }
420                 if ((currentStep.GetType().Name == "CaptureImage") && (
                        comboStepType.SelectedIndex == 3))
421                 {
422                     return;
423                 }
424                 if ((currentStep.GetType().Name == "AnalyzeImage") && (
                        comboStepType.SelectedIndex == 4))
425                 {
426                     return;
427                 }
428                 if ((currentStep.GetType().Name == "UserMessage") && (
                        comboStepType.SelectedIndex == 5))
429                 {
430                     return;
431                 }
432                 if ((currentStep.GetType().Name == "UserFeedback") && (
                        comboStepType.SelectedIndex == 6))
433                 {
```

```
434                     return;
435                 }
436             }
437
438             // Create a new step
439             TestStep newStep = null;
440
441             switch (comboStepType.SelectedIndex)
442             {
443                 case 0:
444                     if (currentStep.GetType().Name != "Sleep")
445                     {
446                         newStep = new Sleep();
447                     }
448                     break;
449                 case 1:
450                     if (currentStep.GetType().Name != "SendCommand")
451                     {
452                         newStep = new SendCommand();
453                         int length;
454                         string combobox_description, command_text;
455                         SendCommand sendCommand = (SendCommand)
                                newStep;
456                         if (comboBoxSendCommand.SelectedItem != null)
457                         {
458                             combobox_description =
                                    comboBoxSendCommand.SelectedItem.
                                    ToString();
459                             length = combobox_description.Length;
460                             if (length >= 3)
461                             {
462                                 command_text = combobox_description.
                                        Substring(length - 3, 3).Substring(0, 2);
463                                 if (command_text == "..")
464                                     sendCommand.SetHexCommand("00");
                                            // Command "0x00" is reserved for
                                            custom command
465                                 else
466                                     sendCommand.SetHexCommand(
                                            command_text); // Set the hex
                                            command
```

```
467                             }
468                         }
469                     }
470                     break;
471                 case 2:
472                     if (currentStep.GetType().Name != "LoadImage")
473                     {
474                         newStep = new LoadImage();
475                     }
476                     break;
477                 case 3:
478                     if (currentStep.GetType().Name != "CaptureImage")
479                     {
480                         newStep = new CaptureImage();
481                     }
482                     break;
483                 case 4:
484                     if (currentStep.GetType().Name != "AnalyzeImage")
485                     {
486                         newStep = new AnalyzeImage();
487                     }
488                     break;
489                 case 5:
490                     if (currentStep.GetType().Name != "UserMessage")
491                     {
492                         newStep = new UserMessage();
493                     }
494                     break;
495                 case 6:
496                     if (currentStep.GetType().Name != "UserFeedback")
497                     {
498                         newStep = new UserFeedback();
499                     }
500                     break;
501                 default: break;
502             }
503 
504             if (newStep != null)
505             {
506                 m_TestCase[m_CurrentStepIndex] = newStep;
507                 m_TestCase.NeedToSave = true;
```

```
508                 currentStep = null;
509                 ShowTestCase();
510                 ShowStepDetails();
511             }
512         }
513
514         private void ShowSleep(Sleep sleep)
515         {
516             m_GuiLocked = true;
517
518             trackBarSleep.Value = sleep.Duration;
519             labelSleep.Text = sleep.Duration.ToString() + "␣ms";
520             textBoxSleepDuration.Text = sleep.Duration.ToString();
521
522             m_GuiLocked = false;
523         }
524
525         // Choose a suitable string for the command parameters tooltip (
                adaptive)
526         private string SelectCommandParametersToolTip(string
                HexCommand)
527         {
528             m_GuiLocked = true;
529
530             // Choose a suitable string for the tooltip
531             string tooltip_text;
532             switch (HexCommand)
533             {
534                 case "22":
535                     tooltip_text = "00␣=␣LVDS␣off␣PP␣on;␣01␣=␣LVDS␣
                            on␣PP␣off;␣02␣=␣LVDS␣on␣PP␣on;␣03␣=␣LVDS
                            ␣off␣PP␣off";
536                     break;
537                 case "23":
538                     tooltip_text = "00␣(Dummy␣parameter)";
539                     break;
540                 case "28":
541                     tooltip_text = "00−7F␣=␣resistance␣value";
542                     break;
543                 case "29":
544                     tooltip_text = "00␣(Dummy␣parameter)";
```

```
545                     break;
546                 case "2A":
547                     tooltip_text = "00␣(Dummy␣parameter)";
548                     break;
549                 case "2E":
550                     tooltip_text = "00␣=␣Disabled;␣01␣=␣Enabled";
551                      break;
552                 case "30":
553                     tooltip_text = "Register␣address␣(1␣byte)\r\nRegister␣
                            new␣value␣(2␣bytes)";
554                     break;
555                 case "31":
556                     tooltip_text = "Register␣address␣(1␣byte)";
557                     break;
558                 case "32":
559                     tooltip_text = "Register␣address:␣(1␣byte)\r\nRegister␣
                            value:␣(2␣bytes)";
560                     break;
561                 case "33":
562                     tooltip_text = "Register␣address␣(1␣byte)";
563                      break;
564                 case "34":
565                     tooltip_text = "Delay␣time␣(1␣byte):␣from␣00␣to␣FF,␣
                            corresponds␣to␣value␣*␣30␣us";
566                     break;
567                 case "35":
568                     tooltip_text = "00␣=␣Disabled;␣01␣=␣Enabled";
569                     break;
570                 case "37":
571                     tooltip_text = "00␣=␣Sensor−only␣Reset;␣01␣=␣Full␣
                            System␣Reset";
572                     break;
573                 case "38":
574                     tooltip_text = "00␣=␣Stop;␣01␣=␣Start";
575                     break;
576                 case "39":
577                     tooltip_text = "00␣=␣Disabled;␣01␣=␣Enabled";
578                     break;
579                 case "3B":
580                     tooltip_text = "00␣=␣Barcode␣Decode;␣01␣=␣
                            Document␣Capture;␣02␣=␣Motion␣Detect;␣03␣=␣
```

```
                    AIM␣Capture;␣04␣=␣LCD␣Screen␣Read";
581                 break;
582             case "3C":
583                 tooltip_text = "00␣=␣Normal;␣Row/Column␣binning␣
                        codes:␣00␣=␣No␣Row/Column␣Bin;␣01␣=␣Row␣
                        Bin␣2;␣02␣=␣Row␣Bin␣4;␣04␣=␣Column␣Bin␣2;
                        ␣08␣=␣Column␣Bin␣4";
584                 break;
585             case "3D":
586                 tooltip_text = "00␣=␣Top;␣01␣=␣Right;␣02␣=␣Bottom;
                        ␣03␣=␣Left";
587                 break;
588             case "3F":
589                 tooltip_text = "00␣=␣Normal␣mode;␣01␣=␣Low␣Power
                        ␣mode";
590                 break;
591             case "40":
592                 tooltip_text = "Parameter␣ID␣Code␣(2␣bytes)";
593                 break;
594             case "41":
595                 tooltip_text = "Parameter␣ID␣Code␣(2␣bytes)␣+␣
                        Parameter␣Data␣(n␣bytes)";
596                 break;
597             case "42":
598                 tooltip_text = "Signature␣(3␣bytes):␣AA,␣50,␣5F";
599                 break;
600             case "44":
601                 tooltip_text = "00␣=␣Disabled;␣01␣=␣Enabled";
602                 break;
603             case "45":
604                 tooltip_text = "0␣=␣5␣ms;␣01−0A␣=␣10−100␣ms␣
                        with␣10␣ms␣step;␣0B−14␣=␣100−900␣ms␣with␣
                        100␣ms␣step;␣15−FF␣=␣1−235␣s␣with␣1␣s␣step
                        ";
605                 break;
606             case "46":
607                 tooltip_text = "List␣number␣to␣set␣(0−9)␣+␣up␣to␣
                        150␣bytes␣of␣command␣scripts";
608                 break;
609             case "47":
610                 tooltip_text = "List␣number␣to␣run␣(0−9)";
```

```
611                     break;
612                 case "48":
613                     tooltip_text = "00␣=␣Off;␣01−0C␣=␣value␣∗␣50␣us";
614                     break;
615                 case "49":
616                     tooltip_text = "None";
617                     break;
618                 case "4A":
619                     tooltip_text = "None";
620                     break;
621                 case "4D":
622                     tooltip_text = "Action:␣00␣=␣get␣public␣key;␣01␣=␣
                            Authenticate␣+␣message␣(see␣documentation)";
623                     break;
624                 case "4E":
625                     tooltip_text = "00␣=␣Default␣(8.5␣ms);␣01−30␣=␣
                            value␣∗␣0.5␣ms";
626                     break;
627                 case "4F":
628                     tooltip_text = "On␣duration:␣00␣=␣Always␣On;␣01−FF
                            ␣=␣n.␣of␣frames␣On;␣\n\rOff␣duration:␣00−FF␣=
                            ␣n␣of␣frames␣Off";
629                     break;
630                 case "F7":
631                     tooltip_text = "On␣duration:␣00␣=␣Disable;␣01−FF␣=␣
                            on␣time␣in␣sec␣;␣\n\rTotal␣cycle␣duration:␣00␣=␣
                            Disable;␣01−FF␣total␣cycle␣time␣in␣sec";
632                     break;
633                 case "F8":
634                     tooltip_text = "00␣=␣off;␣01␣=␣on";
635                     break;
636                 case "F9":
637                     tooltip_text = "00␣=␣Exit;␣01−FF␣=␣Enter␣test␣Mode
                            ␣wirh␣a␣timeout␣of␣N␣sec";
638                      break;
639                 default:
640                     tooltip_text = "Type␣command␣parameters␣here...";
641                     break;
642             }
643
644             m_GuiLocked = false;
```

```
645
646             return tooltip_text;
647         }
648
649         private void ShowSendCommand(SendCommand sendCommand)
650         {
651             int count, length;
652             string HexCommand, combobox_description, command_text;
653
654             m_GuiLocked = true;
655
656             // Show the command description
657             textSendCommandDescription.Text = sendCommand.
                    m_Description;
658
659             // Get the Hex Command
660             HexCommand = sendCommand.GetHexCommand();
661
662             // Determine the index in the combobox for the hex command
663             // The description of each item in the combobox MUST end with the
                    corresponding
664             // hex command within parentheses. For example, "CAMERA
                    RESET (0x37)",
665             // "CAMERA START (0x38)" and so on...
666             // The last index (comboBoxSendCommand.Items.Count - 1) is
                    reserved for the
667             // "CUSTOM COMMAND ...".
668             count = comboBoxSendCommand.Items.Count;
669             for (int i = 0; i < comboBoxSendCommand.Items.Count - 1; i++)
670             {
671                 combobox_description = comboBoxSendCommand.Items[i].
                        ToString();
672                 length = combobox_description.Length;
673                 if (length >= 3)
674                 {
675                     command_text = combobox_description.Substring(length
                            - 3, 3).Substring(0, 2);
676                     if (String.Equals(command_text, HexCommand,
                            StringComparison.OrdinalIgnoreCase))
677                     {
678                         count = i;
```

```
679                         break;
680                     }
681                 }
682             }
683
684             // Select the index in the comboBoxSendCommand which
                    corresponds to HexCommand
685             if (count < comboBoxSendCommand.Items.Count)
686                 comboBoxSendCommand.SelectedIndex = count;
687             else if (HexCommand == "00")
688                 comboBoxSendCommand.SelectedIndex =
                        comboBoxSendCommand.Items.Count - 1;
689
690             // Show the command parameters
691             textCommandParameters.Text = sendCommand.GetHexParameters
                    ();
692
693             // Set up a suitable tooltip for the command parameters textbox
694             toolTipCommandParameters.SetToolTip(this.
                    textCommandParameters, SelectCommandParametersToolTip(
                    HexCommand));
695
696             // Show the I2C address configured for the command
697             textI2CAddress.Text = sendCommand.GetI2CAddress();
698
699             // Show the expected answer (expected status from the slave device)
700             comboBoxExpectedAnswer.SelectedIndex = (int)sendCommand.
                    GetExpectedStatus();
701
702             m_GuiLocked = false;
703         }
704
705         private void buttonBrowseImage_Click(object sender, EventArgs e)
706         {
707             if (m_GuiLocked)
708                 return;
709
710             UserMessage userMessage;
711             DialogResult dialogResult = openDialogImage.ShowDialog();
712             Bitmap bitmap;
713
```

```
714             if (dialogResult == System.Windows.Forms.DialogResult.OK)
715             {
716                 if (openDialogImage.FileName != String.Empty)
717                 {
718                     try
719                     {
720                         bitmap = new Bitmap(openDialogImage.FileName);
721                     }
722                     catch (ArgumentException)
723                     {
724                         ConsoleAppendText("Cannot␣load␣the␣specified␣
                               user␣message␣image␣!\r\n");
725                         return;
726                     }
727
728                     pictureUserMessage.BackgroundImage = bitmap;
729                     pictureUserMessage.Refresh();
730
731                     if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                           m_TestCase.Count)
732                     {
733                         userMessage = (UserMessage)m_TestCase[
                               m_CurrentStepIndex];
734                         userMessage.ImageFileName = openDialogImage.
                               FileName;
735
736                         m_TestCase.NeedToSave = true;
737                     }
738                 }
739             }
740         }
741
742         private void buttonAppendStep_Click(object sender, EventArgs e)
743         {
744             if (m_GuiLocked)
745                 return;
746
747             AppendTestStep();
748             if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                   listViewTestSteps.Items.Count)
```

```
749                 if (listViewTestSteps.Items[m_CurrentStepIndex].Selected ==
                        false)
750                     listViewTestSteps.Items[m_CurrentStepIndex].Selected =
                            true;
751             ShowStepDetails();
752         }
753
754         private void AppendTestStep()
755         {
756             Sleep testStep = new Sleep();
757             m_TestCase.Add(testStep);
758
759             // Execute the following block of code instead of calling
760             // ShowTestCase() from the event buttonAppendStep_Click().
761             if (m_TestCase.Count > 0)
762             {
763                 string[] subItems = new string[2];
764                 subItems[0] = (m_TestCase.Count).ToString();
765                 subItems[1] = m_TestCase[m_TestCase.Count - 1].ToString()
                        ;
766                 ListViewItem listViewItem = new ListViewItem(subItems);
767                 listViewItem.StateImageIndex = -1; // Test step not executed (
                        no icon shown)
768                 listViewTestSteps.Items.Add(listViewItem);
769             }
770
771             m_TestCase.NeedToSave = (m_TestCase.Count > 0);
772
773             m_CurrentStepIndex = m_TestCase.Count - 1;
774         }
775
776         // Shows the listViewTestSteps in the panelDesign
777         private void ShowTestCase()
778         {
779             m_GuiLocked = true;
780
781             if (m_TestCase.Name != null)
782                 textTestCaseName.Text = m_TestCase.Name;
783             else
784                 textTestCaseName.Text = "";
785
```

```
786             listViewTestSteps.Items.Clear();
787
788             // Do not show step details for iterator elements
789             if (iterator_selected_in_plan)
790             {
791                 m_CurrentStepIndex = -1;
792
793                 m_GuiLocked = false;
794
795                 return;
796             }
797
798             if (!m_DesignerView)
799                 if (m_CurrentPlanIndex == -1)
800                 {
801                     m_GuiLocked = false;
802
803                     return;
804                 }
805
806             // In Tester View, set it up to show a pass/fail result image
807             if (m_DesignerView)
808                 listViewTestSteps.StateImageList = null;
809             else
810                 listViewTestSteps.StateImageList = imageListResult;
811
812             for (int i = 0; i < m_TestCase.Count; i++)
813             {
814                 string[] subItems = new string[2];
815                 subItems[0] = (i + 1).ToString();
816                 subItems[1] = m_TestCase[i].ToString();
817                 ListViewItem listViewItem = new ListViewItem(subItems);
818
819                 // In Tester View, show a pass/fail result image
820                 int teststep_result = -1; // Not executed yet (no icon shown).
821                 if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        m_TestPlan.Count)
822                 {
823                     if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                            "TestCase")
```

```
824                         if (i < ((TestCase)m_TestPlan[m_CurrentPlanIndex]).
                                Count)
825                             teststep_result = ((TestStep)((TestCase)
                                    m_TestPlan[m_CurrentPlanIndex])[i]).Result
                                    ;
826                 }
827                 listViewItem.StateImageIndex = teststep_result;
828
829                 listViewTestSteps.Items.Add(listViewItem);
830             }
831
832             if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    listViewTestSteps.Items.Count)
833                 if (listViewTestSteps.Items[m_CurrentStepIndex].Selected ==
                        false)
834                 {
835                     listViewTestSteps.Items[m_CurrentStepIndex].Selected =
                            true;
836                     listViewTestSteps.Items[m_CurrentStepIndex].Focused =
                            true;
837                 }
838
839             m_GuiLocked = false;
840         }
841
842         // Shows the listViewTestStepsForPlan in the panelPlan
843         private void ShowTestCaseForPlan()
844         {
845             m_GuiLocked = true;
846
847             listViewTestStepsForPlan.Items.Clear();
848
849             // Do not show step details for iterator elements
850             if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
851                 if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestIterator")
852                 {
853                     m_CurrentStepIndex = -1;
854
855                     m_GuiLocked = false;
```

```
856
857                     return;
858                 }
859
860             if (!m_DesignerView)
861                 if (m_CurrentPlanIndex == -1)
862                 {
863                     m_GuiLocked = false;
864
865                     return;
866                 }
867
868             listViewTestStepsForPlan.StateImageList = imageListResult;
869
870             for (int i = 0; i < m_TestCase.Count; i++)
871             {
872                 string[] subItems = new string[2];
873                 subItems[0] = (i + 1).ToString();
874                 subItems[1] = m_TestCase[i].ToString();
875                 ListViewItem listViewItem = new ListViewItem(subItems);
876
877                 // In Tester View, show a pass/fail result image
878                 int teststep_result = -1; // Not executed yet (no icon shown).
879                 if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        m_TestPlan.Count)
880                 {
881                     if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                            "TestCase")
882                         if (i < ((TestCase)m_TestPlan[m_CurrentPlanIndex]).
                                Count)
883                             teststep_result = ((TestStep)((TestCase)
                                    m_TestPlan[m_CurrentPlanIndex])[i]).Result
                                    ;
884                 }
885                 listViewItem.StateImageIndex = teststep_result;
886
887                 listViewTestStepsForPlan.Items.Add(listViewItem);
888             }
889
890             m_GuiLocked = false;
891         }
```

```
892
893         private void TestStep_ResultChanged(object sender, EventArgs e)
894         {
895             // This call needs to be asynchronous otherwise it causes a
                    deadlock when the execution
896             // thread is stopped from the main thread.
897             if (this.listViewTestSteps.InvokeRequired)
898             {
899                 GenericEventCallback callback = new GenericEventCallback(
                        TestStep_ResultChanged);
900                 this.BeginInvoke(callback, new object[] { sender, e });
901             }
902             else
903             {
904                 ShowTestCaseForPlan();
905             }
906         }
907
908         private void TestCase_ResultChanged(object sender, EventArgs e)
909         {
910             // This call needs to be asynchronous otherwise it causes a
                    deadlock when the execution
911             // thread is stopped from the main thread.
912             if (this.listViewTestPlan.InvokeRequired)
913             {
914                 GenericEventCallback callback = new GenericEventCallback(
                        TestCase_ResultChanged);
915                 this.BeginInvoke(callback, new object[] { sender, e });
916             }
917             else
918             {
919                 ShowTestPlan();
920             }
921         }
922
923         private void Unregister_TestSteps_ResultChanged_EventHandlers()
924         {
925             // Unregister/remove all result change event handlers in all test
                    steps of each test case of the test plan
926             for (int i = 0; i < m_TestPlan.Count; i++)
927             {
```

```
928                 if (m_TestPlan[i].GetType().Name == "TestCase")
929                 {
930                     TestCase testCase = (TestCase)m_TestPlan[i];
931                     for (int j = 0; j < testCase.Count; j++)
932                     {
933                         TestStep testStep = (TestStep)testCase[j];
934                         testStep.ResultChanged -= TestStep_ResultChanged;
935                     }
936                 }
937             }
938         }
939
940         private void Unregister_TestCase_ResultChanged_EventHandlers()
941         {
942             // Unregister/remove all result change event handlers in all test
                    cases of the test plan
943             for (int i = 0; i < m_TestPlan.Count; i++)
944             {
945                 if (m_TestPlan[i].GetType().Name == "TestCase")
946                 {
947                     TestCase testCase = (TestCase)m_TestPlan[i];
948                     testCase.ResultChanged -= TestCase_ResultChanged;
949                 }
950             }
951         }
952
953         private void LoadTestCaseAtIndex(int index)
954         {
955             m_GuiLocked = true;
956
957             if (index >= 0 && index < m_TestPlan.Count)
958             {
959                 // If it's not a Test Case, return immediately
960                 if (m_TestPlan[index].GetType().Name != "TestCase")
961                 {
962                     m_GuiLocked = false;
963
964                     return;
965                 }
966
967                 // Load the test case at the given index (if there is one)
```

```
968                 TestCase testCase = (TestCase)m_TestPlan[index];
969                 try
970                 {
971                     m_TestCase.ReadXml(testCase.FileName);
972                 }
973                 catch (System.IO.FileNotFoundException)
974                 {
975                     ConsoleAppendText("Error: test case file " + testCase.
                         FileName + " cannot be found on the filesystem.\
                         r\n");
976                     m_GuiLocked = false;
977 
978                     return;
979                 }
980                 catch (System.Xml.XmlException)
981                 {
982                     ConsoleAppendText("Error: unreadable XML in Test 
                         Case file " + testCase.FileName + ".\r\n");
983                     m_GuiLocked = false;
984 
985                     return;
986                 }
987 
988                 // Unregister all test step result changed event handlers
989                 Unregister_TestSteps_ResultChanged_EventHandlers();
990 
991                 // Add an event handler to each test step that refreshes
                     listViewTestStepsForPlan by calling ShowTestCaseForPlan
                     ()
992                 // when the test step result changes (used in Tester View only)
993                 for (int i = 0; i < testCase.Count; i++)
994                 {
995                     TestStep testStep = (TestStep)testCase[i];
996                     testStep.ResultChanged += new
                         ResultChangedEventHandler(
                         TestStep_ResultChanged);
997                 }
998             }
999 
1000            m_GuiLocked = false;
1001        }
```

```
1002
1003        private void buttonRemoveStep_Click(object sender, EventArgs e)
1004        {
1005            if (m_GuiLocked)
1006                return;
1007
1008            if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    m_TestCase.Count) // Avoids ArgumentOutOfRangeException
1009            {
1010                m_TestCase.RemoveAt(m_CurrentStepIndex);
1011                m_TestCase.NeedToSave = true;
1012            }
1013
1014            // Adjust the current index if incorrect and show the Test Case
1015            if (m_CurrentStepIndex >= m_TestCase.Count)
1016            {
1017                m_CurrentStepIndex = m_TestCase.Count - 1;
1018            }
1019
1020            ShowTestCase();
1021
1022            if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    listViewTestSteps.Items.Count)
1023                if (listViewTestSteps.Items[m_CurrentStepIndex].Selected ==
                        false)
1024                    listViewTestSteps.Items[m_CurrentStepIndex].Selected =
                            true;
1025
1026            ShowStepDetails();
1027        }
1028
1029        private void buttonInsertStep_Click(object sender, EventArgs e)
1030        {
1031            if (m_GuiLocked)
1032                return;
1033
1034            Sleep testStep = new Sleep();
1035
1036            if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    m_TestCase.Count) // Avoids ArgumentOutOfRangeException
1037            {
```

```
1038                m_TestCase.Insert(m_CurrentStepIndex, testStep);
1039                m_TestCase.NeedToSave = true;
1040            }
1041
1042            // Show the Test Case, keep the current index
1043            ShowTestCase();
1044
1045            if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    listViewTestSteps.Items.Count)
1046                if (listViewTestSteps.Items[m_CurrentStepIndex].Selected ==
                        false)
1047                    listViewTestSteps.Items[m_CurrentStepIndex].Selected =
                            true;
1048
1049            ShowStepDetails();
1050        }
1051
1052        private void textUserMessage_TextChanged(object sender, EventArgs
                e)
1053        {
1054            if (m_GuiLocked)
1055                return;
1056
1057            UserMessage userMessage = (UserMessage)m_TestCase[
                    m_CurrentStepIndex];
1058            userMessage.SetUserMessage(textUserMessage.Text);
1059
1060            m_TestCase.NeedToSave = true;
1061        }
1062
1063        private void ShowUserMessage(UserMessage userMessage)
1064        {
1065            m_GuiLocked = true;
1066
1067            textUserMessage.Text = userMessage.GetUserMessage();
1068            textUserMessageDescription.Text = userMessage.m_Description;
1069
1070            if (userMessage.ImageFileName != null && userMessage.
                    ImageFileName != String.Empty)
1071            {
1072                Bitmap bitmap;
```

```
1073
1074                    try
1075                    {
1076                        bitmap = new Bitmap(userMessage.ImageFileName);
1077                    }
1078                    catch (ArgumentException)
1079                    {
1080                        ConsoleAppendText("Cannot␣load␣the␣specified␣user␣
                                message␣image␣!\r\n");
1081                        return;
1082                    }
1083
1084                    pictureUserMessage.BackgroundImage = bitmap;
1085                }
1086                else
1087                    pictureUserMessage.BackgroundImage = null;
1088
1089                pictureUserMessage.Refresh();
1090
1091                m_GuiLocked = false;
1092            }
1093
1094            public void ShowUserMessageImage(string imageFileName)
1095            {
1096                Bitmap bitmap;
1097
1098                try
1099                {
1100                    bitmap = new Bitmap(imageFileName);
1101                }
1102                catch (ArgumentException)
1103                {
1104                    ConsoleAppendText("Cannot␣load␣the␣specified␣user␣
                            message␣image␣!\r\n");
1105                    return;
1106                }
1107
1108                ClearImageSelectionFromPictureBox();
1109
1110                // Set the image as the background image for the picturebox in
                        panelExecute
```

```
1111            pictureBox1.BackgroundImage = bitmap;
1112        }
1113
1114        private void radioUserFeedbackNo_CheckedChanged(object sender,
                EventArgs e)
1115        {
1116            if (m_GuiLocked)
1117                return;
1118
1119            //UserFeedback userFeedback = (UserFeedback)m_TestCase[
                    m_CurrentStepIndex];
1120            //userFeedback.SetFeedback(!radioUserFeedbackNo.Checked);
1121        }
1122
1123        private void ShowUserFeedback(UserFeedback userFeedback)
1124        {
1125            m_GuiLocked = true;
1126
1127            radioUserFeedbackYes.Checked = userFeedback.GetFeedback();
1128            radioUserFeedbackNo.Checked = !radioUserFeedbackYes.
                    Checked;
1129            textUserFeedbackDescription.Text = userFeedback.m_Description;
1130            textUserFeedbackMessage.Text = userFeedback.Message;
1131
1132            m_GuiLocked = false;
1133        }
1134
1135        private void radioUserFeedbackYes_CheckedChanged(object sender,
                EventArgs e)
1136        {
1137            if (m_GuiLocked)
1138                return;
1139
1140            UserFeedback userFeedback = (UserFeedback)m_TestCase[
                    m_CurrentStepIndex];
1141            userFeedback.SetFeedback(radioUserFeedbackYes.Checked);
1142
1143            m_TestCase.NeedToSave = true;
1144        }
1145
```

```
1146        private void textSendCommandDescription_TextChanged(object
                sender, EventArgs e)
1147        {
1148            if (m_GuiLocked)
1149                return;
1150
1151            TestStep sendCommand = (TestStep)m_TestCase[
                    m_CurrentStepIndex];
1152            sendCommand.m_Description = textSendCommandDescription.
                    Text;
1153
1154            m_TestCase.NeedToSave = true;
1155
1156            ShowTestCase();
1157        }
1158
1159        private void textLoadImageDescription_TextChanged(object sender,
                EventArgs e)
1160        {
1161            if (m_GuiLocked)
1162                return;
1163
1164            LoadImage loadImage = (LoadImage)m_TestCase[
                    m_CurrentStepIndex];
1165            loadImage.m_Description = textLoadImageDescription.Text;
1166
1167            m_TestCase.NeedToSave = true;
1168
1169            ShowTestCase();
1170        }
1171
1172        private void textCaptureImageDescription_TextChanged(object sender
                , EventArgs e)
1173        {
1174            if (m_GuiLocked)
1175                return;
1176
1177            TestStep descriptionText = (TestStep)m_TestCase[
                    m_CurrentStepIndex];
1178            descriptionText.m_Description = textCaptureImageDescription.
                    Text;
```

```
1179
1180            m_TestCase.NeedToSave = true;
1181
1182            ShowTestCase();
1183        }
1184
1185        private void textAnalyzeImageDescription_TextChanged(object
                sender, EventArgs e)
1186        {
1187            if (m_GuiLocked)
1188                return;
1189
1190            TestStep descriptionText = (TestStep)m_TestCase[
                    m_CurrentStepIndex];
1191            descriptionText.m_Description = textAnalyzeImageDescription.
                    Text;
1192
1193            m_TestCase.NeedToSave = true;
1194
1195            ShowTestCase();
1196        }
1197
1198        private void textUserFeedbackDescription_TextChanged(object sender
                , EventArgs e)
1199        {
1200            if (m_GuiLocked)
1201                return;
1202
1203            TestStep descriptionText = (TestStep)m_TestCase[
                    m_CurrentStepIndex];
1204            descriptionText.m_Description = textUserFeedbackDescription.
                    Text;
1205
1206            m_TestCase.NeedToSave = true;
1207
1208            ShowTestCase();
1209        }
1210
1211        private void textUserMessageDescription_TextChanged(object sender,
                 EventArgs e)
1212        {
```

```
1213            if (m_GuiLocked)
1214                return;
1215
1216            TestStep descriptionText = (TestStep)m_TestCase[
                    m_CurrentStepIndex];
1217            descriptionText.m_Description = textUserMessageDescription.Text
                    ;
1218
1219            m_TestCase.NeedToSave = true;
1220
1221            ShowTestCase();
1222        }
1223
1224        private void checkSaveImage_CheckedChanged(object sender,
                EventArgs e)
1225        {
1226            if (m_GuiLocked)
1227                return;
1228
1229            textCaptureImageSaveImageFileName.Enabled = checkSaveImage.
                    Checked;
1230            buttonBrowseImageSaveFolder.Enabled = checkSaveImage.
                    Checked;
1231            textCaptureImageSaveImageFolder.Enabled = checkSaveImage.
                    Checked;
1232            numericUpDownCaptureImageNumberOfFrames.Enabled =
                    checkSaveImage.Checked;
1233            CaptureImage saveImage = (CaptureImage)m_TestCase[
                    m_CurrentStepIndex];
1234            saveImage.SaveImage = checkSaveImage.Checked;
1235
1236            m_TestCase.NeedToSave = true;
1237        }
1238
1239        private void loadTestMenu_Click(object sender, EventArgs e)
1240        {
1241            if (m_GuiLocked)
1242                return;
1243
1244            openTestCase.CheckFileExists = true;
1245            openTestCase.Multiselect = false;
```

```
1246            // openDialogImage.InitialDirectory = m_InitialDirectory;
1247            openTestCase.RestoreDirectory = true;
1248            openTestCase.Filter = "Test␣Case␣Files␣(*.tc)|*.tc";
1249            openTestCase.FilterIndex = 1;
1250            openTestCase.Title = "Open␣Test␣Case␣File";
1251            DialogResult dialogResult = openTestCase.ShowDialog();
1252
1253            if (dialogResult == System.Windows.Forms.DialogResult.OK)
1254            {
1255                // Load an existing test case XML file
1256                if (openTestCase.FileName != String.Empty) // Avoids
                        ArgumentException in m_TestCase.ReadXml().
1257                {
1258                    try
1259                    {
1260                        m_TestCase.ReadXml(openTestCase.FileName);
1261                    }
1262                    catch (System.IO.FileNotFoundException)
1263                    {
1264                        ConsoleAppendText("Error:␣test␣case␣file␣" +
                                openTestCase.FileName + "␣cannot␣be␣found␣
                                on␣the␣filesystem.\r\n");
1265                        return;
1266                    }
1267                    catch (System.Xml.XmlException)
1268                    {
1269                        ConsoleAppendText("Error:␣unreadable␣XML␣in␣
                                Test␣Case␣file␣" + openTestCase.FileName + ".\
                                r\n");
1270                        return;
1271                    }
1272                }
1273
1274                iterator_selected_in_plan = false;
1275
1276                // Show it
1277                ShowTestCase();
1278
1279                if (listViewTestSteps.Items.Count > 0)
1280                {
```

```
1281                    listViewTestSteps.Items[0].Selected = true; // GT
                            19092014: Select the first step...
1282                    listViewTestSteps.Items[0].Focused = true;
1283                    if (listViewTestSteps.SelectedIndices.Count > 0)
1284                        m_CurrentStepIndex = listViewTestSteps.
                                SelectedIndices[0];
1285                    else
1286                        m_CurrentStepIndex = -1;
1287                    listViewTestSteps.Focus();
1288                }
1289
1290                ShowStepDetails();
1291
1292                m_TestCase.NeedToSave = false;
1293            }
1294        }
1295
1296        private void saveAsTestMenu_Click(object sender, EventArgs e)
1297        {
1298            if (m_GuiLocked)
1299                return;
1300
1301            string fileName = AskTestCaseFileName();
1302
1303            // Save the current test case
1304            if (fileName != null && fileName != String.Empty)
1305            {
1306                m_TestCase.WriteXml(fileName);
1307                m_TestCase.NeedToSave = false;
1308            }
1309        }
1310
1311        private string AskTestCaseFileName()
1312        {
1313            saveTestCase.RestoreDirectory = true;
1314            saveTestCase.Filter = "Test␣Case␣Files␣(*.tc)|*.tc";
1315            saveTestCase.FilterIndex = 1;
1316            saveTestCase.Title = "Save␣Test␣Case␣File";
1317
1318            DialogResult dialogResult = saveTestCase.ShowDialog();
1319
```

```
1320            if (dialogResult != System.Windows.Forms.DialogResult.OK)
1321            {
1322                return (null);
1323            }
1324
1325            return saveTestCase.FileName;
1326        }
1327
1328        private void toolStripTextBoxI2CDefaultAddress_TextChanged(object
                    sender, EventArgs e)
1329        {
1330            if (m_GuiLocked)
1331                return;
1332
1333            if (toolStripTextBoxI2CDefaultAddress.Text.Length == 0)
1334                return;
1335
1336            if (toolStripTextBoxI2CDefaultAddress.Text.Length > 2)
1337            {
1338                MessageBox.Show("Invalid␣hexadecimal␣number␣for␣the␣
                        I2C␣Default␣Address␣!␣Using␣default␣value␣" +
                        I2C_DEFAULT_ADDRESS + ".");
1339                I2CAddress = UInt16.Parse(Properties.Settings.Default.
                        I2CAddress, NumberStyles.AllowHexSpecifier);
1340                toolStripTextBoxI2CDefaultAddress.Text = Properties.Settings
                        .Default.I2CAddress;
1341                SendCommand.SetI2CDefaultAddress(
                        toolStripTextBoxI2CDefaultAddress.Text);
1342                return;
1343            }
1344
1345            // Controllare che sia una stringa esadecimale di due caratteri e
                    poi salvare in SendCommand
1346            try
1347            {
1348                I2CAddress = UInt16.Parse(
                        toolStripTextBoxI2CDefaultAddress.Text, NumberStyles.
                        AllowHexSpecifier);
1349            }
1350            catch (FormatException)
1351            {
```

```
1352                MessageBox.Show("Invalid␣hexadecimal␣number␣for␣the␣
                        I2C␣Default␣Address␣!␣Using␣default␣value␣" +
                        I2C_DEFAULT_ADDRESS + ".");
1353                I2CAddress = UInt16.Parse(Properties.Settings.Default.
                        I2CAddress, NumberStyles.AllowHexSpecifier);
1354                toolStripTextBoxI2CDefaultAddress.Text = Properties.Settings
                        .Default.I2CAddress;
1355                SendCommand.SetI2CDefaultAddress(
                        toolStripTextBoxI2CDefaultAddress.Text);
1356                return;
1357            }
1358
1359            Properties.Settings.Default.I2CAddress =
                    toolStripTextBoxI2CDefaultAddress.Text;
1360            SendCommand.SetI2CDefaultAddress(
                    toolStripTextBoxI2CDefaultAddress.Text);
1361        }
1362
1363        private void CheckTestCaseWantToSave()
1364        {
1365            if (m_TestCase.NeedToSave && m_TestCase.Count > 0)
1366            {
1367                DialogResult dialogResult = MessageBox.Show("The␣test␣
                        case␣has␣not␣been␣saved.␣Do␣you␣want␣to␣save␣it?",
                         "Save␣or␣Discard␣changes?", MessageBoxButtons.
                        YesNo);
1368                if (dialogResult == DialogResult.Yes)
1369                {
1370                    SaveTestCase();
1371                }
1372            }
1373        }
1374
1375        private void MainForm_FormClosing(object sender,
                FormClosingEventArgs e)
1376        {
1377            if (!CheckTestCaseNeedToSave())
1378            {
1379                e.Cancel = true;
1380                return;
1381            }
```

```
1382
1383            if (!CheckTestPlanNeedToSave())
1384            {
1385                e.Cancel = true;
1386                return;
1387            }
1388
1389            JustBeforeClose();
1390        }
1391
1392        private void exitMenu_Click(object sender, EventArgs e)
1393        {
1394            Application.Exit();
1395        }
1396
1397        private bool CheckTestCaseNeedToSave()
1398        {
1399            if (m_TestCase.NeedToSave && m_TestCase.Count > 0)
1400            {
1401                DialogResult dialogResult = MessageBox.Show("The␣test␣
                        case␣has␣not␣been␣saved.␣Exit␣anyway?", "Exit␣or␣
                        Continue?", MessageBoxButtons.YesNo);
1402                if (dialogResult == DialogResult.Yes)
1403                    return true;
1404                else
1405                    return false;
1406            }
1407            else
1408                return true;
1409        }
1410
1411        private bool CheckTestPlanNeedToSave()
1412        {
1413            if (m_TestPlan.NeedToSave && m_TestPlan.Count > 0)
1414            {
1415                DialogResult dialogResult = MessageBox.Show("The␣test␣
                        plan␣has␣not␣been␣saved.␣Exit␣anyway?", "Exit␣or␣
                        Continue?", MessageBoxButtons.YesNo);
1416                if (dialogResult == DialogResult.Yes)
1417                    return true;
1418                else
```

```
1419                        return false;
1420                }
1421                else
1422                    return true;
1423            }
1424
1425            private void JustBeforeClose()
1426            {
1427                // TODO: It should save the report level to the separate report
                        configuration file
1428                Properties.Settings.Default.Location = this.Location;
1429                Properties.Settings.Default.Width = this.Width;
1430                Properties.Settings.Default.Height = this.Height;
1431                Properties.Settings.Default.Save();
1432
1433    #if (FRAMEGRABBER)
1434                // Stop framegrabber streaming
1435                Framegrabber.StopStreaming();
1436
1437                // Disconnect from the framegrabber device
1438                Framegrabber.Disconnect();
1439    #endif
1440
1441                if (workerObject != null && workerObject.workerThread.IsAlive)
1442                {
1443                    // Request that the worker thread stop itself (async call to
                            avoid a deadlock in subsequent Join())
1444                    AsyncRequestThreadStopCaller caller = new
                            AsyncRequestThreadStopCaller(workerObject.
                            RequestStop);
1445                    caller.BeginInvoke(null, null);
1446
1447                    // Use the Join method to block the current thread
1448                    // until the object's thread terminates.
1449                    workerObject.workerThread.Join();
1450                }
1451            }
1452
1453            private void textTestCaseName_TextChanged(object sender,
                    EventArgs e)
1454            {
```

```
1455            if (m_GuiLocked)
1456                return;
1457
1458            m_TestCase.Name = textTestCaseName.Text;
1459
1460            m_TestCase.NeedToSave = true;
1461        }
1462
1463        private void buttonBrowseTests_Click(object sender, EventArgs e)
1464        {
1465            if (m_GuiLocked)
1466                return;
1467
1468            // TODO aggiungere anche la possibilitÃ  di caricare altri test plan
1469            browseTestCase.CheckFileExists = true;
1470            browseTestCase.Multiselect = true;
1471            // openDialogImage.InitialDirectory = m_InitialDirectory;
1472            browseTestCase.RestoreDirectory = true;
1473            browseTestCase.Filter = "Test␣Case␣Files␣(*.tc)|*.tc";
1474            browseTestCase.FilterIndex = 1;
1475            browseTestCase.Title = "Select␣Test␣Case␣Files";
1476            DialogResult dialogResult = browseTestCase.ShowDialog();
1477
1478            if (dialogResult == System.Windows.Forms.DialogResult.OK)
1479            {
1480                // For each filename
1481                for (int i = 0; i < browseTestCase.FileNames.Length; i++)
1482                {
1483                    // Create a test case and open the file
1484                    TestCase testCase = new TestCase();
1485                    try
1486                    {
1487                        testCase.ReadXml(browseTestCase.FileNames[i]);
1488                    }
1489                    catch (System.IO.FileNotFoundException)
1490                    {
1491                        ConsoleAppendText("Error:␣test␣case␣file␣" +
                                browseTestCase.FileNames[i] + "␣cannot␣be␣
                                found␣on␣the␣filesystem.\r\n");
1492                        continue;
1493                    }
```

```
1494                    catch (System.Xml.XmlException)
1495                    {
1496                        ConsoleAppendText("Error: unreadable XML in
                                Test Case file " + browseTestCase.FileNames[i
                                ] + " at position index " + (i + 1) + ".\r\n");
1497                        continue;
1498                    }
1499
1500                    // Add an event handler to the test case that refreshes
                            listViewTestPlan
1501                    // by calling ShowTestPlan()
1502                    testCase.ResultChanged += new
                            ResultChangedEventHandler(
                            TestCase_ResultChanged);
1503
1504                    // Insert the test case at the currently selected position of
                            the test plan
1505                    if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                            m_TestPlan.Count)
1506                    {
1507                        m_TestPlan.Insert(m_CurrentPlanIndex, testCase);
1508                        ListViewItem listViewItem = new ListViewItem(
                                m_TestPlan[m_CurrentPlanIndex].ToString());
1509                        listViewItem.StateImageIndex = ((TestCase)
                                m_TestPlan[m_CurrentPlanIndex]).Result;
1510                        listViewTestPlan.Items.Insert(m_CurrentPlanIndex,
                                listViewItem);
1511                        m_CurrentPlanIndex += 1;
1512                    }
1513                    else
1514                    {
1515                        m_TestPlan.Add(testCase);
1516                        ListViewItem listViewItem = new ListViewItem(
                                m_TestPlan[m_TestPlan.Count - 1].ToString());
1517                        listViewItem.StateImageIndex = ((TestCase)
                                m_TestPlan[m_TestPlan.Count - 1]).Result;
1518                        listViewTestPlan.Items.Add(listViewItem);
1519                    }
1520
1521                    m_TestPlan.NeedToSave = true;
1522                }
```

```
1523
1524                // Show the test plan details
1525                ShowPlanDetails();
1526            }
1527        }
1528
1529        private void buttonAddIteration_Click(object sender, EventArgs e)
1530        {
1531            if (m_GuiLocked)
1532                return;
1533
1534            // Create an iterator objectt
1535            TestIterator testIterator = new TestIterator();
1536
1537            // Insert the (unconfigured) iterator in the test plan at the currently
                    selected position
1538            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
1539            {
1540                m_TestPlan.Insert(m_CurrentPlanIndex, testIterator);
1541                ListViewItem listViewItem = new ListViewItem(m_TestPlan[
                        m_CurrentPlanIndex].ToString());
1542                listViewItem.StateImageIndex = -1;
1543                listViewTestPlan.Items.Insert(m_CurrentPlanIndex,
                        listViewItem);
1544                m_CurrentPlanIndex += 1;
1545            }
1546            else
1547            {
1548                m_TestPlan.Add(testIterator);
1549                ListViewItem listViewItem = new ListViewItem(m_TestPlan[
                        m_TestPlan.Count - 1].ToString());
1550                listViewItem.StateImageIndex = -1;
1551                listViewTestPlan.Items.Add(listViewItem);
1552            }
1553
1554            m_TestPlan.NeedToSave = true;
1555
1556            // Show the test plan details
1557            ShowPlanDetails();
1558        }
```

```
1559
1560            // Shows the current test plan as it is stored in m_TestPlan
1561            private void ShowTestPlan()
1562            {
1563                m_GuiLocked = true;
1564
1565                if (m_TestPlan.Name != null)
1566                    textTestPlanName.Text = m_TestPlan.Name;
1567                else
1568                    textTestPlanName.Text = "";
1569
1570                listViewTestPlan.Items.Clear();
1571
1572                // In Tester View, set it up to show a pass/fail result image
1573                if (m_DesignerView)
1574                    listViewTestPlan.StateImageList = null;
1575                else
1576                    listViewTestPlan.StateImageList = imageListResult;
1577
1578                for (int i = 0; i < m_TestPlan.Count; i++)
1579                {
1580                    ListViewItem listViewItem = new ListViewItem(m_TestPlan[i
                            ].ToString());
1581
1582                    if (m_TestPlan[i].GetType().Name == "TestCase")
1583                        listViewItem.StateImageIndex = ((TestCase)m_TestPlan[i
                                ]).Result;
1584                    else
1585                        listViewItem.StateImageIndex = -1;
1586
1587                    listViewTestPlan.Items.Add(listViewItem);
1588                }
1589
1590                // The following block has caused NullReferenceException at
                        Selected = true
1591                // before introducing the lock scheme.
1592                if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        listViewTestPlan.Items.Count)
1593                    if (listViewTestPlan.Items[m_CurrentPlanIndex].Selected ==
                            false)
1594                    {
```

```
1595                    listViewTestPlan.Items[m_CurrentPlanIndex].Selected =
                            true;
1596                    listViewTestPlan.Items[m_CurrentPlanIndex].Focused =
                            true;
1597                }
1598
1599            m_GuiLocked = false;
1600        }
1601
1602        private void ShowPlanDetails()
1603        {
1604            m_GuiLocked = true;
1605
1606            // If there are no test cases, then disable the "Remove" button from
                    panelPlan.
1607            // The entries in the contextual menu are configured in the opening
                    event
1608            // of the contextual menu.
1609            // Also, disable or enable the file save items in the toolstrip menu.
1610            if (m_TestPlan.Count == 0)
1611                buttonRemoveTest.Enabled = false;
1612            else
1613                buttonRemoveTest.Enabled = true;
1614
1615            // If there are less than two test cases, then disable the "Move Up"
                    and "Move Down"
1616            // buttons from panelPlan.
1617            // The entries in the contextual menu are configured in the opening
                    event of the
1618            // contextual menu.
1619            if (m_TestPlan.Count <= 1)
1620            {
1621                buttonMoveUp.Enabled = false;
1622                buttonMoveDown.Enabled = false;
1623            }
1624            else
1625            {
1626                buttonMoveUp.Enabled = true;
1627                buttonMoveDown.Enabled = true;
1628            }
1629
```

```
1630            m_GuiLocked = false;
1631        }
1632
1633        private void labelMajorMinor_Click(object sender, EventArgs e)
1634        {
1635            if (m_GuiLocked)
1636                return;
1637
1638            AnalyzeImage analyzeImage = (AnalyzeImage)m_TestCase[
                    m_CurrentStepIndex];
1639
1640            if (labelMajorMinor.Text == ">=")
1641            {
1642                labelMajorMinor.Text = "<=";
1643            }
1644            else
1645            {
1646                labelMajorMinor.Text = ">=";
1647            }
1648
1649            analyzeImage.SetOperator(labelMajorMinor.Text);
1650
1651            m_TestCase.NeedToSave = true;
1652        }
1653
1654        private void textUserFeedbackMessage_TextChanged(object sender,
                EventArgs e)
1655        {
1656            if (m_GuiLocked)
1657                return;
1658
1659            UserFeedback userFeedback = (UserFeedback)m_TestCase[
                    m_CurrentStepIndex];
1660            userFeedback.Message = textUserFeedbackMessage.Text;
1661
1662            m_TestCase.NeedToSave = true;
1663        }
1664
1665        private void comboImageAlgorithm_SelectedIndexChanged(object
                sender, EventArgs e)
1666        {
```

```
1667            if (m_GuiLocked)
1668                return;
1669
1670            AnalyzeImage analyzeImage = (AnalyzeImage)m_TestCase[
                    m_CurrentStepIndex];
1671            analyzeImage.SetImageAnalysis((ImageAnalysis)
                    comboImageAlgorithm.SelectedIndex);
1672
1673            m_TestCase.NeedToSave = true;
1674        }
1675
1676        private void numericValueTarget_ValueChanged(object sender,
                EventArgs e)
1677        {
1678            if (m_GuiLocked)
1679                return;
1680
1681            AnalyzeImage analyzeImage = (AnalyzeImage)m_TestCase[
                    m_CurrentStepIndex];
1682            analyzeImage.SetValueTarget((int)numericValueTarget.Value);
1683
1684            m_TestCase.NeedToSave = true;
1685        }
1686
1687        private void ShowAnalyzeImage(AnalyzeImage analyzeImage)
1688        {
1689            m_GuiLocked = true;
1690
1691            textAnalyzeImageDescription.Text = analyzeImage.m_Description;
1692            comboImageAlgorithm.SelectedIndex = (int)analyzeImage.
                    GetImageAnalysis();
1693            numericValueTarget.Value = (decimal)analyzeImage.
                    GetValueTarget();
1694            labelMajorMinor.Text = analyzeImage.GetOperator() ? ">=" : "<="
                    ;
1695
1696            m_GuiLocked = false;
1697        }
1698
1699        private double CalculatePictureBoxZoom()
1700        {
```

```
1701            double zoomX, zoomY;
1702            double zoom = 1.0;
1703            int imageHeight = 0;
1704            int imageWidth = 0;
1705            float pictureBoxHeight = 0;
1706            float pictureBoxWidth = 0;
1707
1708            if (m_BackImage != null)
1709            {
1710                try
1711                {
1712                    imageWidth = m_BackImage.Width;
1713                    imageHeight = m_BackImage.Height;
1714                }
1715                catch (ArgumentException)
1716                {
1717                    return zoom; // Error is reported by the caller
1718                }
1719            }
1720            else
1721                return zoom;
1722
1723            if (pictureBox1.BackgroundImage != null)
1724            {
1725                pictureBoxWidth = pictureBox1.Width;
1726                pictureBoxHeight = pictureBox1.Height;
1727            }
1728            else
1729                return zoom;
1730
1731            zoomX = (double)pictureBoxWidth / imageWidth;
1732            zoomY = (double)pictureBoxHeight / imageHeight;
1733
1734            // Get the minimum between zoomX and zoomY (the one which
                    produces a smaller image)
1735            zoom = zoomX < zoomY ? zoomX : zoomY;
1736
1737            return zoom;
1738        }
1739
1740        // Create a list of rectangles that are scaled according to zoom and
```

```
1741        // repositioned in one coordinate according to the resulting offset
1742        private void CreateScaledImageSelectionList(double zoom, int Xpos,
                int Ypos)
1743        {
1744            Rectangle sourceRect, outputRect;
1745
1746            scaledselectionList.Clear();
1747
1748            for (int i = 0; i < selectionList.Count; i++)
1749            {
1750                sourceRect = selectionList[i];
1751                outputRect = new Rectangle((int)((sourceRect.X - Xpos) /
                        zoom), (int)((sourceRect.Y - Ypos) / zoom), (int)(
                        sourceRect.Width / zoom), (int)(sourceRect.Height / zoom
                        ));
1752                scaledselectionList.Add(outputRect);
1753            }
1754        }
1755
1756        public void AnalyzeImage(ref AnalyzeImage analyzeImage)
1757        {
1758            m_GuiLocked = true;
1759
1760            double zoom = 1.0;
1761            int expandedwidth = 0, expandedheight = 0;
1762            int Xoffset = 0, Yoffset = 0;
1763
1764            ImageAnalysis imageAnalysis = analyzeImage.GetImageAnalysis
                    ();
1765            bool missingselection = false;
1766            double value = 0;
1767
1768            if (m_BackImage != null)
1769                m_ImageAnalyzer.MainBitmap = m_BackImage;
1770            else
1771            {
1772                analyzeImage.Result = -1;
1773                analyzeImage.failure_reason = "No␣bitmap";
1774
1775                m_GuiLocked = false;
1776
```

```
1777                return;
1778            }
1779
1780            // The secondary image is only used for Inter-frame Noise and
                    Inter-frame
1781            // Brightness Stability
1782            if (m_SecondaryImage != null)
1783                m_ImageAnalyzer.SecondBitmap = m_SecondaryImage;
1784            else
1785                m_ImageAnalyzer.SecondBitmap = null;
1786
1787            if (pictureBox1.BackgroundImageLayout == ImageLayout.Zoom)
1788            {
1789                zoom = CalculatePictureBoxZoom();
1790                try
1791                {
1792                    expandedwidth = (int)(m_BackImage.Width * zoom);
1793                    expandedheight = (int)(m_BackImage.Height * zoom);
1794                }
1795                catch (ArgumentException)
1796                {
1797                    ConsoleAppendText("Analyze␣Image:␣Cannot␣
                            determine␣image␣size␣!\r\n");
1798
1799                    analyzeImage.Result = 0;
1800                    analyzeImage.failure_reason = "Cannot␣determine␣
                            image␣size.";
1801
1802                    m_GuiLocked = false;
1803
1804                    return;
1805                }
1806                Xoffset = (pictureBox1.Width - expandedwidth) / 2;
1807                Yoffset = (pictureBox1.Height - expandedheight) / 2;
1808            }
1809
1810            // Scale the selected areas according to the zoom and reposition
                    them
1811            // according to the resulting offset in one coordinate.
1812            CreateScaledImageSelectionList(zoom, Xoffset, Yoffset);
1813
```

```
1814            switch (imageAnalysis)
1815            {
1816                case ImageAnalysis.Brightness:
1817                    if (selectionList.Count > 0)
1818                        value = m_ImageAnalyzer.ComputeBrightness(
                                scaledselectionList[0]);
1819                    else
1820                        missingselection = true;
1821                    break;
1822                case ImageAnalysis.BrightnessLoss:
1823                    if (selectionList.Count > 1)
1824                        value = m_ImageAnalyzer.ComputeBrightnessLoss(
                                scaledselectionList[0], scaledselectionList[1]);
1825                    else
1826                        missingselection = true;
1827                    break;
1828                case ImageAnalysis.BrightnessDistribution:
1829                    value = m_ImageAnalyzer.
                            ComputeBrightnessDistribution(scaledselectionList
                            [0]);
1830                    break;
1831                case ImageAnalysis.Contrast:
1832                    if (selectionList.Count > 1)
1833                        value = m_ImageAnalyzer.ComputeContrast(
                                scaledselectionList[0], scaledselectionList[1]);
1834                    else
1835                        missingselection = true;
1836                    break;
1837                case ImageAnalysis.ContrastBalance:
1838                    value = m_ImageAnalyzer.ComputeContrastBalance(
                            scaledselectionList[0]);
1839                    break;
1840                case ImageAnalysis.PixelNoise:
1841                    value = m_ImageAnalyzer.ComputePixelNoise(
                            scaledselectionList[0]);
1842                    break;
1843                case ImageAnalysis.Snr:
1844                    if (selectionList.Count > 1)
1845                        value = m_ImageAnalyzer.ComputeSNR(
                                scaledselectionList[0], scaledselectionList[1]);
1846                    else
```

```
1847                        missingselection = true;
1848                    break;
1849                case ImageAnalysis.InterFrameNoise:
1850                    value = m_ImageAnalyzer.ComputeInterframeNoise(
                            scaledselectionList[0]);
1851                    break;
1852                case ImageAnalysis.InterFrameBrightnessStability:
1853                    value = m_ImageAnalyzer.
                            ComputeInterframeBrightnessStability(
                            scaledselectionList[0]);
1854                    break;
1855                case ImageAnalysis.BrightSaturation:
1856                    if (selectionList.Count > 0)
1857                        value = m_ImageAnalyzer.ComputeWhiteSaturation(
                                scaledselectionList[0]);
1858                    else
1859                        missingselection = true;
1860                    break;
1861                case ImageAnalysis.DarkSaturation:
1862                    if (selectionList.Count > 0)
1863                        value = m_ImageAnalyzer.ComputeBlackSaturation(
                                scaledselectionList[0]);
1864                    else
1865                        missingselection = true;
1866                    break;
1867                case ImageAnalysis.Blur:
1868                    if (scaledselectionList.Count >= 3)
1869                        value = m_ImageAnalyzer.ComputeBlur(
                                scaledselectionList[0], scaledselectionList[1],
                                scaledselectionList[2]);
1870                    else
1871                        missingselection = true;
1872                    break;
1873                case ImageAnalysis.AimVisibility:
1874                    if (scaledselectionList.Count >= 1)
1875                        value = m_ImageAnalyzer.
                                ComputeAimingCrossVisibility(
                                scaledselectionList[0]);
1876                    else
1877                        missingselection = true;
1878                     break;
```

```
1879                    default:
1880                        if (selectionList.Count > 0)
1881                            value = m_ImageAnalyzer.ComputeBrightness(
                                    scaledselectionList[0]);
1882                        else
1883                            missingselection = true;
1884                        break;
1885                }
1886
1887                if (missingselection)
1888                {
1889                    analyzeImage.Result = -1;
1890                    analyzeImage.failure_reason = "Region(s) of interest not
                            selected.";
1891                }
1892                else if (value < 0)
1893                {
1894                    analyzeImage.Result = -1;
1895                    analyzeImage.failure_reason = "Image Analyzer error.";
1896                }
1897                else
1898                    if (analyzeImage.GetOperator())
1899                    {
1900                        analyzeImage.Result = (value >= analyzeImage.
                                GetValueTarget()) ? 1 : 0;
1901                    }
1902                    else
1903                    {
1904                        analyzeImage.Result = (value <= analyzeImage.
                                GetValueTarget()) ? 1 : 0;
1905                    }
1906
1907  #if (VERBOSE_ANALYZE_IMAGE)
1908                ConsoleAppendText("Analyze Image: " + imageAnalysis.
                        ToString() + " produced a value of " + value + "\r\n");
1909  #endif
1910
1911                // Forget secondary image
1912                if (m_SecondaryImage != null)
1913                {
1914                    m_SecondaryImage.Dispose();
```

```
1915                m_SecondaryImage = null;
1916            }
1917
1918            // Clear selected areas from the picturebox in panelExecute
1919            ClearImageSelectionFromPictureBox();
1920
1921            m_GuiLocked = false;
1922
1923            return;
1924        }
1925
1926        private void textLoadImageFileName_TextChanged(object sender,
                EventArgs e)
1927        {
1928            if (m_GuiLocked)
1929                return;
1930
1931            LoadImage loadImage = (LoadImage)m_TestCase[
                    m_CurrentStepIndex];
1932            loadImage.FileName = textLoadImageFileName.Text;
1933
1934            m_TestCase.NeedToSave = true;
1935        }
1936
1937        private void checkBoxLoadSecondaryImage_CheckedChanged(object
                sender, EventArgs e)
1938        {
1939            if (m_GuiLocked)
1940                return;
1941
1942            LoadImage loadImage = (LoadImage)m_TestCase[
                    m_CurrentStepIndex];
1943            loadImage.SecondaryImage = checkBoxLoadSecondaryImage.
                    Checked;
1944
1945            m_TestCase.NeedToSave = true;
1946        }
1947
1948        private void ShowLoadImage(LoadImage loadImage)
1949        {
1950            m_GuiLocked = true;
```

```
1951
1952            textLoadImageDescription.Text = loadImage.m_Description;
1953            textLoadImageFileName.Text = loadImage.FileName;
1954            checkBoxLoadSecondaryImage.Checked = loadImage.
                    SecondaryImage;
1955
1956            m_GuiLocked = false;
1957        }
1958
1959        private void buttonBrowseFileNameLoadImage_Click(object sender,
                EventArgs e)
1960        {
1961            if (m_GuiLocked)
1962                return;
1963
1964            browseLoadImage.CheckFileExists = true;
1965            browseLoadImage.Multiselect = false;
1966            // browseLoadImage.InitialDirectory = m_InitialDirectory;
1967            browseLoadImage.RestoreDirectory = true;
1968            browseLoadImage.Filter = "Image␣Files␣(*.bmp␣*.gif␣*.jpeg␣*.
                    jpg␣*.png␣*.wmf)|*.bmp;␣*.gif;␣*.jpeg;␣*.jpg;␣*.png;␣*.
                    wmf";
1969            browseLoadImage.FilterIndex = 1;
1970            browseLoadImage.Title = "Select␣Image␣File";
1971            DialogResult dialogResult = browseLoadImage.ShowDialog();
1972
1973            if (dialogResult == System.Windows.Forms.DialogResult.OK)
1974            {
1975                if (browseLoadImage.FileName != String.Empty)
1976                {
1977                    LoadImage loadImage = (LoadImage)m_TestCase[
                            m_CurrentStepIndex];
1978                    loadImage.FileName = browseLoadImage.FileName;
1979                    textLoadImageFileName.Text = loadImage.FileName;
1980                }
1981
1982                // Show the test case (removes the "(unconfigured)" label)
1983                ShowTestCase();
1984            }
1985        }
1986
```

```
1987        // Clear all selected areas from the picturebox in panelExecute
1988        private void ClearImageSelectionFromPictureBox()
1989        {
1990            // This call needs to be asynchronous otherwise it causes a
                    deadlock when the execution
1991            // thread is stopped from the main thread.
1992            if (this.pictureBox1.InvokeRequired)
1993            {
1994                ClearImageSelectionFromPictureBoxDelegate callback = new
                        ClearImageSelectionFromPictureBoxDelegate(
                        ClearImageSelectionFromPictureBox);
1995                this.BeginInvoke(callback);
1996            }
1997            else
1998            {
1999                if (selectionList.Count > 0 || scaledselectionList.Count > 0)
2000                {
2001                    m_GuiLocked = true;
2002
2003                    // Create an empty image area selection list
2004                    selectionList.Clear();
2005                    scaledselectionList.Clear();
2006
2007                    // Create a new foreground image to hold the regions of
                            interest
2008                    if (m_ForeImage != null)
2009                        m_ForeImage.Dispose();
2010                    if (m_PictureGraphics != null)
2011                        m_PictureGraphics.Dispose();
2012                    m_ForeImage = new Bitmap(pictureBox1.Width,
                            pictureBox1.Height, System.Drawing.Imaging.
                            PixelFormat.Format32bppArgb);
2013                    m_PictureGraphics = Graphics.FromImage(m_ForeImage
                            );
2014
2015                    // Display the new foreground image in the picturebox
2016                    pictureBox1.Image = m_ForeImage;
2017
2018                    pictureBox1.Refresh();
2019
2020                    m_GuiLocked = false;
```

```
2021                }
2022            }
2023        }
2024
2025        public void LoadImage(ref LoadImage loadImage)
2026        {
2027            m_GuiLocked = true;
2028
2029            if (loadImage.FileName == null || loadImage.FileName == String.
                    Empty)
2030            {
2031                loadImage.failure_reason = "Empty␣image␣filename";
2032                loadImage.Result = 0;
2033
2034                m_GuiLocked = false;
2035
2036                return;
2037            }
2038
2039            if (loadImage.SecondaryImage == false)
2040            {
2041                try
2042                {
2043                    m_BackImage = new Bitmap(loadImage.FileName);
2044                }
2045                catch (System.IO.FileNotFoundException)
2046                {
2047                    ConsoleAppendText("Error:␣image␣file␣" +
                            browseLoadImage.FileName + "␣cannot␣be␣found␣
                            on␣the␣filesystem.\r\n");
2048                    loadImage.failure_reason = "Image␣file␣" +
                            browseLoadImage.FileName + "␣cannot␣be␣found␣
                            on␣the␣filesystem";
2049                    loadImage.Result = 0;
2050
2051                    m_GuiLocked = false;
2052
2053                    return;
2054                }
2055                catch (ArgumentException)
2056                {
```

```
2057                    ConsoleAppendText("Error:␣invalid␣image␣file␣" +
                            browseLoadImage.FileName + ".\r\n");
2058                    loadImage.failure_reason = "Image␣file␣" +
                            browseLoadImage.FileName + "␣is␣invalid";
2059                    loadImage.Result = 0;
2060
2061                    m_GuiLocked = false;
2062
2063                    return;
2064                }
2065            }
2066            else
2067            {
2068                if (m_SecondaryImage != null)
2069                {
2070                    m_SecondaryImage.Dispose();
2071                    m_SecondaryImage = null;
2072                }
2073                try
2074                {
2075                    m_SecondaryImage = new Bitmap(loadImage.FileName);
2076                }
2077                catch (System.IO.FileNotFoundException)
2078                {
2079                    ConsoleAppendText("Error:␣image␣file␣" +
                            browseLoadImage.FileName + "␣cannot␣be␣found␣
                            on␣the␣filesystem.\r\n");
2080                    loadImage.failure_reason = "Image␣file␣" +
                            browseLoadImage.FileName + "␣cannot␣be␣found␣
                            on␣the␣filesystem";
2081                    loadImage.Result = 0;
2082
2083                    m_GuiLocked = false;
2084
2085                    return;
2086                }
2087                catch (ArgumentException)
2088                {
2089                    ConsoleAppendText("Error:␣invalid␣image␣file␣" +
                            browseLoadImage.FileName + ".\r\n");
```

```
2090                    loadImage.failure_reason = "Image␣file␣" +
                            browseLoadImage.FileName + "␣is␣invalid";
2091                    loadImage.Result = 0;
2092
2093                    m_GuiLocked = false;
2094
2095                    return;
2096                }
2097            }
2098
2099            if (loadImage.SecondaryImage == false)
2100            {
2101                ClearImageSelectionFromPictureBox();
2102                pictureBox1.BackgroundImage = (System.Drawing.Image)
                        m_BackImage.Clone();
2103            }
2104
2105            // Successful test step
2106            loadImage.Result = 1;
2107
2108            m_GuiLocked = false;
2109        }
2110
2111        private void textCaptureImageSaveImageFileName_TextChanged(
                object sender, EventArgs e)
2112        {
2113            if (m_GuiLocked)
2114                return;
2115
2116            CaptureImage captureImage = (CaptureImage)m_TestCase[
                    m_CurrentStepIndex];
2117            captureImage.FileName = textCaptureImageSaveImageFileName.
                    Text;
2118
2119            m_TestCase.NeedToSave = true;
2120        }
2121
2122        private void textCurrentSaveImageFolder_TextChanged(object sender,
                EventArgs e)
2123        {
2124            if (m_GuiLocked)
```

```
2125                return;
2126
2127            CaptureImage captureImage = (CaptureImage)m_TestCase[
                    m_CurrentStepIndex];
2128            captureImage.SaveFolder = textCaptureImageSaveImageFolder.
                    Text;
2129
2130            m_TestCase.NeedToSave = true;
2131        }
2132
2133        private void buttonBrowseImageSaveFolder_Click(object sender,
                EventArgs e)
2134        {
2135            if (m_GuiLocked)
2136                return;
2137
2138            DialogResult dialogResult = folderBrowserSaveImage.ShowDialog
                    ();
2139            if (dialogResult == System.Windows.Forms.DialogResult.OK)
2140            {
2141                textCaptureImageSaveImageFolder.Text =
                        folderBrowserSaveImage.SelectedPath;
2142            }
2143        }
2144
2145        private void
                numericUpDownCaptureImageNumberOfFrames_ValueChanged(
                object sender, EventArgs e)
2146        {
2147            if (m_GuiLocked)
2148                return;
2149
2150            CaptureImage captureImage = (CaptureImage)m_TestCase[
                    m_CurrentStepIndex];
2151            captureImage.Frames = (int)
                    numericUpDownCaptureImageNumberOfFrames.Value;
2152
2153            m_TestCase.NeedToSave = true;
2154        }
2155
2156        private void ShowCaptureImage(CaptureImage captureImage)
```

```
2157            {
2158                m_GuiLocked = true;
2159
2160                textCaptureImageDescription.Text = captureImage.m_Description;
2161                textCaptureImageSaveImageFileName.Text = captureImage.
                        FileName;
2162                textCaptureImageSaveImageFolder.Text = captureImage.
                        SaveFolder;
2163                numericUpDownCaptureImageNumberOfFrames.Value = (
                        decimal)captureImage.Frames;
2164                checkSaveImage.Checked = captureImage.SaveImage;
2165
2166                // Enable/disable optional controls used only when the image
                        should be saved
2167                textCaptureImageSaveImageFileName.Enabled = checkSaveImage.
                        Checked;
2168                buttonBrowseImageSaveFolder.Enabled = checkSaveImage.
                        Checked;
2169                textCaptureImageSaveImageFolder.Enabled = checkSaveImage.
                        Checked;
2170                numericUpDownCaptureImageNumberOfFrames.Enabled =
                        checkSaveImage.Checked;
2171
2172                m_GuiLocked = false;
2173            }
2174
2175            public void CaptureImage(ref CaptureImage captureImage)
2176            {
2177                int result = -1;
2178
2179                ClearImageSelectionFromPictureBox();
2180
2181    #if (FRAMEGRABBER)
2182                result = Framegrabber.Capture(FrameGrabberWidth,
                        FrameGrabberHeight, captureImage.Frames, captureImage.
                        SaveImage, captureImage.FileName, captureImage.SaveFolder
                        );
2183
2184                if (result > 1)
2185                    ConsoleAppendText("Capture␣Image:␣captured␣" + result + "
                            ␣out␣of␣" + captureImage.Frames + "␣images.\r\n");
```

```
2186
2187            if (result <= 0) // unsuccessful image capture
2188                switch (result)
2189                {
2190                    case -1:
2191                        ConsoleAppendText("Capture␣Image:␣Framegrabber
                                ␣device␣has␣not␣been␣selected.␣Cannot␣
                                connect␣to␣a␣framegrabber␣device␣!\r\n");
2192                        break;
2193                    case -2:
2194                        ConsoleAppendText("Capture␣Image:␣Cannot␣
                                select␣a␣framegrabber␣device.␣Try␣disabling␣
                                the␣firewall.\r\n");
2195                        break;
2196                    case -3:
2197                        ConsoleAppendText("Capture␣Image:␣Cannot␣get␣
                                framegrabber␣device␣information.\r\n");
2198                        break;
2199                    case -4:
2200                        ConsoleAppendText("Capture␣Image:␣Cannot␣
                                connect␣to␣selected␣framegrabber␣device.\r\n")
                                ;
2201                        break;
2202                    case -5:
2203                        ConsoleAppendText("Capture␣Image:␣Cannot␣
                                enable␣streaming␣from␣framegrabber␣device␣
                                !\r\n");
2204                        break;
2205                    case 0: // It means 0 images have been captured...
2206                    case -6:
2207                        ConsoleAppendText("Capture␣Image:␣Image␣
                                acquisition␣unsuccessful␣!\r\n");
2208                        break;
2209                    case -7:
2210                        ConsoleAppendText("Capture␣Image:␣Image␣
                                acquisition␣unsuccessful:␣could␣not␣start␣
                                acquisition␣manager␣!\r\n");
2211                        break;
2212                    case -8:
2213                        ConsoleAppendText("Capture␣Image:␣Framegrabber
                                ␣configuration␣failed␣!\r\n");
```

```
2214                            break;
2215                        default:
2216                            ConsoleAppendText("Capture␣Image:␣Unknown␣
                                    error␣during␣image␣capturing␣!\r\n");
2217                            break;
2218                    }
2219 #endif
2220
2221                // Each time a new image is available from the framegrabber it will
                        be automatically
2222                // set as the new background image by the
                        Framegrabber_ImageChanged() event and it
2223                // will then be automatically refreshed in the picturebox by the
                        pictureBox1_BackgroundImageChanged()
2224                // event.
2225
2226                if (result > 0)
2227                    captureImage.Result = 1;
2228                else
2229                    captureImage.Result = 0;
2230            }
2231
2232            private void newTestMenu_Click(object sender, EventArgs e)
2233            {
2234                if (m_GuiLocked)
2235                    return;
2236
2237                // Clean the current test case
2238                m_TestCase.Reset();
2239                m_CurrentStepIndex = -1;
2240                ShowTestCase();
2241                ShowStepDetails();
2242            }
2243
2244            private void saveTestMenu_Click(object sender, EventArgs e)
2245            {
2246                if (m_GuiLocked)
2247                    return;
2248
2249                SaveTestCase();
2250            }
```

```
2251
2252        private void SaveTestCase()
2253        {
2254            string fileName = m_TestCase.FileName;
2255            if (m_TestCase.FileName == null || m_TestCase.FileName ==
                    String.Empty)
2256            {
2257                fileName = AskTestCaseFileName();
2258            }
2259
2260            // Save the current test case
2261            if (fileName != null && fileName != String.Empty)
2262            {
2263                m_TestCase.WriteXml(fileName);
2264                m_TestCase.NeedToSave = false;
2265            }
2266        }
2267
2268        private void buttonRemoveTest_Click(object sender, EventArgs e)
2269        {
2270            int m_CurrentPlanIndex_old = m_CurrentPlanIndex;
2271
2272            if (m_GuiLocked)
2273                return;
2274
2275            if (m_TestPlan.Count > 1)
2276                CheckTestCaseWantToSave();
2277
2278            for (int i = m_TestPlan.Count - 1; i >= 0; i--)
2279            {
2280                if (i < listViewTestPlan.Items.Count)
2281                    if (listViewTestPlan.Items[i].Selected == true)
2282                    {
2283                        m_TestPlan.RemoveAt(i);
2284                        m_TestPlan.NeedToSave = true;
2285                        listViewTestPlan.Items[i].Remove();
2286                    }
2287            }
2288
2289            // Adjust the current index if incorrect (last entry has been removed
                )
```

```
2290            if (m_CurrentPlanIndex >= m_TestPlan.Count)
2291                m_CurrentPlanIndex = m_TestPlan.Count − 1;
2292
2293            // If the selection has been lost, recover it
2294            if (m_CurrentPlanIndex < 0)
2295            {
2296                // Keep the previous list selection index
2297                if (m_CurrentPlanIndex_old >= 0 &&
                        m_CurrentPlanIndex_old < m_TestPlan.Count)
2298                    m_CurrentPlanIndex = m_CurrentPlanIndex_old;
2299
2300                // If the last element has been removed, select what becomes
                        the new last element
2301                if (m_CurrentPlanIndex_old >= 0 &&
                        m_CurrentPlanIndex_old >= m_TestPlan.Count)
2302                    m_CurrentPlanIndex = m_TestPlan.Count − 1;
2303            }
2304
2305            // Show the selected line
2306            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
2307            {
2308                listViewTestPlan.Items[m_CurrentPlanIndex].Selected = true;
2309                listViewTestPlan.Items[m_CurrentPlanIndex].Focused = true;
2310            }
2311
2312            // Load the next test case (if there is one) and handle the test step
                    result change event
2313            LoadTestCaseAtIndex(m_CurrentPlanIndex);
2314
2315            // Show the test plan and the test case
2316            ShowPlanDetails();
2317            ShowTestCase();
2318
2319            // Show the test case in the panelPlan (Tester View only)
2320            ShowTestCaseForPlan();
2321        }
2322
2323        private void moveUpTestMenu_Click(object sender, EventArgs e)
2324        {
2325            if (m_GuiLocked)
```

```
2326                return;
2327
2328            if (m_CurrentPlanIndex > 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
2329            {
2330                int index = m_CurrentPlanIndex;
2331                if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestCase")
2332                {
2333                    TestCase testCase = new TestCase();
2334                    testCase = (TestCase)m_TestPlan[m_CurrentPlanIndex];
2335                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2336                    m_TestPlan.Insert(m_CurrentPlanIndex - 1, testCase);
2337                    m_TestPlan.NeedToSave = true;
2338                    listViewTestPlan.Items[index].Remove();
2339                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[index - 1].ToString());
2340                    listViewItem.StateImageIndex = ((TestCase)m_TestPlan[
                            index - 1]).Result;
2341                    listViewTestPlan.Items.Insert(index - 1, listViewItem);
2342                }
2343                else if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                         "TestIterator")
2344                {
2345                    TestIterator testIterator = new TestIterator();
2346                    testIterator = (TestIterator)m_TestPlan[
                            m_CurrentPlanIndex];
2347                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2348                    m_TestPlan.Insert(m_CurrentPlanIndex - 1, testIterator);
2349                    m_TestPlan.NeedToSave = true;
2350                    listViewTestPlan.Items[index].Remove();
2351                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[index - 1].ToString());
2352                    listViewItem.StateImageIndex = -1;
2353                    listViewTestPlan.Items.Insert(index - 1, listViewItem);
2354                }
2355                else
2356                    return;
2357
2358                m_CurrentPlanIndex -= 1;
2359            }
```

```
2360        }
2361
2362        private void moveDownTestMenu_Click(object sender, EventArgs e)
2363        {
2364            if (m_GuiLocked)
2365                return;
2366
2367            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count - 1)
2368            {
2369                int index = m_CurrentPlanIndex;
2370                if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestCase")
2371                {
2372                    TestCase testCase = new TestCase();
2373                    testCase = (TestCase)m_TestPlan[m_CurrentPlanIndex];
2374                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2375                    m_TestPlan.Insert(m_CurrentPlanIndex + 1, testCase);
2376                    m_TestPlan.NeedToSave = true;
2377                    listViewTestPlan.Items[index].Remove();
2378                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[index + 1].ToString());
2379                    listViewItem.StateImageIndex = ((TestCase)m_TestPlan[
                            index + 1]).Result;
2380                    listViewTestPlan.Items.Insert(index + 1, listViewItem);
2381                }
2382                else if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                         "TestIterator")
2383                {
2384                    TestIterator testIterator = new TestIterator();
2385                    testIterator = (TestIterator)m_TestPlan[
                            m_CurrentPlanIndex];
2386                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2387                    m_TestPlan.Insert(m_CurrentPlanIndex + 1, testIterator);
2388                    m_TestPlan.NeedToSave = true;
2389                    listViewTestPlan.Items[index].Remove();
2390                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[index + 1].ToString());
2391                    listViewItem.StateImageIndex = -1;
2392                    listViewTestPlan.Items.Insert(index + 1, listViewItem);
2393                }
```

```
2394                else
2395                    return;
2396
2397                m_CurrentPlanIndex += 1;
2398            }
2399        }
2400
2401        private void moveToTopTestMenu_Click(object sender, EventArgs e)
2402        {
2403            if (m_GuiLocked)
2404                return;
2405
2406            if (m_CurrentPlanIndex > 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
2407            {
2408                int index = m_CurrentPlanIndex;
2409                if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestCase")
2410                {
2411                    TestCase testCase = new TestCase();
2412                    testCase = (TestCase)m_TestPlan[m_CurrentPlanIndex];
2413                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2414                    m_TestPlan.Insert(0, testCase);
2415                    m_TestPlan.NeedToSave = true;
2416                    listViewTestPlan.Items[index].Remove();
2417                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[0].ToString());
2418                    listViewItem.StateImageIndex = ((TestCase)m_TestPlan
                            [0]).Result;
2419                    listViewTestPlan.Items.Insert(0, listViewItem);
2420                }
2421                else if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                         "TestIterator")
2422                {
2423                    TestIterator testIterator = new TestIterator();
2424                    testIterator = (TestIterator)m_TestPlan[
                            m_CurrentPlanIndex];
2425                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2426                    m_TestPlan.Insert(0, testIterator);
2427                    m_TestPlan.NeedToSave = true;
2428                    listViewTestPlan.Items[index].Remove();
```

```
2429                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[0].ToString());
2430                    listViewItem.StateImageIndex = -1;
2431                    listViewTestPlan.Items.Insert(0, listViewItem);
2432                }
2433                else
2434                    return;
2435
2436                m_CurrentPlanIndex = 0;
2437            }
2438        }
2439
2440        private void moveToBottomTestMenu_Click(object sender, EventArgs
                e)
2441        {
2442            if (m_GuiLocked)
2443                return;
2444
2445            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count - 1)
2446            {
2447                int index = m_CurrentPlanIndex;
2448                if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestCase")
2449                {
2450                    TestCase testCase = new TestCase();
2451                    testCase = (TestCase)m_TestPlan[m_CurrentPlanIndex];
2452                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2453                    m_TestPlan.Insert(m_TestPlan.Count, testCase);
2454                    m_TestPlan.NeedToSave = true;
2455                    listViewTestPlan.Items[index].Remove();
2456                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[m_TestPlan.Count - 1].ToString());
2457                    listViewItem.StateImageIndex = ((TestCase)m_TestPlan[
                            m_TestPlan.Count - 1]).Result;
2458                    listViewTestPlan.Items.Insert(listViewTestPlan.Items.
                            Count, listViewItem);
2459                }
2460                else if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                         "TestIterator")
2461                {
```

```
2462                    TestIterator testIterator = new TestIterator();
2463                    testIterator = (TestIterator)m_TestPlan[
                            m_CurrentPlanIndex];
2464                    m_TestPlan.RemoveAt(m_CurrentPlanIndex);
2465                    m_TestPlan.Insert(m_TestPlan.Count, testIterator);
2466                    m_TestPlan.NeedToSave = true;
2467                    listViewTestPlan.Items[index].Remove();
2468                    ListViewItem listViewItem = new ListViewItem(
                            m_TestPlan[m_TestPlan.Count - 1].ToString());
2469                    listViewItem.StateImageIndex = -1;
2470                    listViewTestPlan.Items.Insert(listViewTestPlan.Items.
                            Count, listViewItem);
2471                }
2472                else
2473                    return;
2474
2475                m_CurrentPlanIndex = m_TestPlan.Count - 1;
2476            }
2477        }
2478
2479        private void savePlanMenu_Click(object sender, EventArgs e)
2480        {
2481            if (m_GuiLocked)
2482                return;
2483
2484            string fileName = m_TestPlan.FileName;
2485
2486            if (m_TestPlan.FileName == null || m_TestPlan.FileName ==
                    String.Empty)
2487            {
2488                fileName = AskTestPlanFileName();
2489            }
2490
2491            // Save the current test plan
2492            if (fileName != null && fileName != String.Empty)
2493            {
2494                m_TestPlan.WriteXml(fileName);
2495                m_TestPlan.NeedToSave = false;
2496            }
2497        }
2498
```

```
2499        private void saveAsPlanMenu_Click(object sender, EventArgs e)
2500        {
2501            if (m_GuiLocked)
2502                return;
2503
2504            string fileName = AskTestPlanFileName();
2505
2506            // Save the current test plan
2507            if (fileName != null && fileName != String.Empty)
2508            {
2509                m_TestPlan.WriteXml(fileName);
2510                m_TestPlan.NeedToSave = false;
2511            }
2512        }
2513
2514        private string AskTestPlanFileName()
2515        {
2516            saveTestPlan.RestoreDirectory = true;
2517            saveTestPlan.Filter = "Test␣Plan␣Files␣(*.tp)|*.tp";
2518            saveTestPlan.FilterIndex = 1;
2519            saveTestPlan.Title = "Save␣Test␣Plan␣File";
2520
2521            DialogResult dialogResult = saveTestPlan.ShowDialog();
2522
2523            if (dialogResult != System.Windows.Forms.DialogResult.OK)
2524            {
2525                return (null);
2526            }
2527
2528            return saveTestPlan.FileName;
2529        }
2530
2531        private void loadPlanMenu_Click(object sender, EventArgs e)
2532        {
2533            if (m_GuiLocked)
2534                return;
2535
2536            openTestPlan.CheckFileExists = true;
2537            openTestPlan.Multiselect = false;
2538            // openDialogImage.InitialDirectory = m_InitialDirectory;
2539            openTestPlan.RestoreDirectory = true;
```

```
2540            openTestPlan.Filter = "Test␣Plan␣Files␣(*.tp)|*.tp";
2541            openTestPlan.FilterIndex = 1;
2542            openTestPlan.Title = "Open␣Test␣Plan␣File";
2543            DialogResult dialogResult = openTestPlan.ShowDialog();
2544
2545            if (dialogResult == System.Windows.Forms.DialogResult.OK)
2546            {
2547                // Load an existing test plan XML file
2548                if (openTestPlan.FileName != String.Empty) // Avoids
                        ArgumentException in m_TestPlan.ReadXml().
2549                {
2550                    try
2551                    {
2552                        m_TestPlan.ReadXml(openTestPlan.FileName);
2553                    }
2554                    catch (System.IO.FileNotFoundException)
2555                    {
2556                        ConsoleAppendText("Error:␣test␣plan␣file␣" +
                                openTestPlan.FileName + "␣cannot␣be␣found␣
                                on␣the␣filesystem.\r\n");
2557                        return;
2558                    }
2559                    catch (System.Xml.XmlException)
2560                    {
2561                        ConsoleAppendText("Error:␣unreadable␣XML␣in␣
                                Test␣Plan␣file␣" + openTestPlan.FileName + ".\r
                                \n");
2562                        return;
2563                    }
2564                }
2565
2566                // Add an event handler to each test case so that it refreshes
                        listViewTestPlan
2567                // by calling ShowTestPlan()
2568                for (int i = 0; i < m_TestPlan.Count; i++)
2569                {
2570                    if (m_TestPlan[i].GetType().Name == "TestCase")
2571                    {
2572                        TestCase testCase = (TestCase)m_TestPlan[i];
2573                        testCase.ResultChanged += new
                                ResultChangedEventHandler(
```

```
                    TestCase_ResultChanged);
2574            }
2575        }
2576
2577        // Show it
2578        ShowTestPlan();
2579        ShowPlanDetails();
2580
2581        if (listViewTestPlan.Items.Count > 0)
2582        {
2583            listViewTestPlan.Items[0].Selected = true; // GT
                    19092014: Select the first step
2584            listViewTestPlan.Items[0].Focused = true;
2585            listViewTestPlan.Focus();
2586        }
2587
2588        if (listViewTestPlan.SelectedIndices.Count > 0)
2589            m_CurrentPlanIndex = listViewTestPlan.SelectedIndices
                    [0];
2590
2591        CheckTestCaseWantToSave();
2592
2593        // Load the selected test case (if there is one) and handle the
                test step result change event
2594        LoadTestCaseAtIndex(m_CurrentPlanIndex);
2595
2596        // If there is at least one step, select it
2597        if (m_TestCase.Count > 0)
2598            m_CurrentStepIndex = 0;
2599
2600        // Show the test case and the first step details
2601        ShowTestCase();
2602        ShowStepDetails();
2603
2604        // Show the test case in the panelPlan (visible only in Tester
                View)
2605        ShowTestCaseForPlan();
2606
2607        if (listViewTestSteps.Items.Count > 0)
2608            listViewTestSteps.Items[0].Selected = true; // GT
                    19092014: After having selected the first step, show it
```

```
                    as selected...
2609            }
2610        }
2611
2612        private void newPlanMenu_Click(object sender, EventArgs e)
2613        {
2614            if (m_GuiLocked)
2615                return;
2616
2617            m_TestPlan.Reset();
2618            m_CurrentPlanIndex = -1;
2619            ShowTestPlan();
2620            ShowPlanDetails();
2621            ShowTestCase();
2622            ShowTestCaseForPlan();
2623        }
2624
2625        private void aboutMenu_Click(object sender, EventArgs e)
2626        {
2627            if (m_GuiLocked)
2628                return;
2629
2630            MessageBox.Show("SETP version " + Assembly.
                    GetExecutingAssembly().GetName().Version + "\nmscorlib.dll
                    version " + typeof(String).Assembly.GetName().Version + "\
                    nDeveloped by Guido Trentalancia for " + this.
                    CompanyName + ".", "About SETP", MessageBoxButtons.
                    OK, MessageBoxIcon.Information);
2631        }
2632
2633        // The HTML Help Workshop produces a CHM file. It does only accept
                GIF, JPEG and PNG images.
2634        private void helpFileMenu_Click(object sender, EventArgs e)
2635        {
2636            if (System.IO.File.Exists(helpProvider1.HelpNamespace))
2637                Help.ShowHelp(this, helpProvider1.HelpNamespace);
2638            else
2639                Help.ShowHelp(this, "SETP.chm");
2640        }
2641
```

```
2642        private void listViewTestPlan_MouseClick(object sender,
                MouseEventArgs e)
2643        {
2644            int m_CurrentPlanIndex_old = m_CurrentPlanIndex;
2645
2646            if (m_GuiLocked)
2647                return;
2648
2649            // Determine the current index
2650            if (listViewTestPlan.SelectedIndices.Count == 0)
2651            {
2652                m_CurrentPlanIndex = -1;
2653            }
2654            else // load the selected Test Case
2655            {
2656                m_CurrentPlanIndex = listViewTestPlan.SelectedIndices[0];
2657
2658                if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        listViewTestPlan.Items.Count)
2659                    iterator_selected_in_plan = (m_TestPlan[
                            m_CurrentPlanIndex].GetType().Name == "
                            TestIterator");
2660
2661                if (m_CurrentPlanIndex != m_CurrentPlanIndex_old)
2662                {
2663                    CheckTestCaseWantToSave();
2664
2665                    // Load the selected test case (if there is one) and handle
                            the test step result change event
2666                    LoadTestCaseAtIndex(m_CurrentPlanIndex);
2667
2668                    // If there is at least one step, select it
2669                    if (m_TestCase.Count > 0)
2670                        m_CurrentStepIndex = 0;
2671
2672                    // Show it
2673                    ShowTestCase();
2674                    ShowStepDetails();
2675
2676                    if (listViewTestSteps.Items.Count > 0)
```

```
2677                        listViewTestSteps.Items[0].Selected = true; // GT
                                19092014: After having selected the first step,
                                show it as selected...
2678
2679                    // Show the test case in the panelPlan (Tester View only)
2680                    ShowTestCaseForPlan();
2681                }
2682            }
2683
2684            if (m_CurrentPlanIndex != m_CurrentPlanIndex_old)
2685                ShowPlanDetails();
2686        }
2687
2688        public void ConsoleAppendText(string text)
2689        {
2690            m_GuiLocked = true;
2691
2692            // This call needs to be asynchronous otherwise it causes a
                    deadlock when the execution
2693            // thread is stopped from the main thread.
2694            if (this.textConsoleForExecute.InvokeRequired)
2695            {
2696                ConsoleAppendTextDelegate callback = new
                        ConsoleAppendTextDelegate(ConsoleAppendText);
2697                this.BeginInvoke(callback, new object[] { text });
2698            }
2699            else
2700            {
2701                try
2702                {
2703                    this.textConsoleForPlan.AppendText(text);
2704                    this.textConsoleForExecute.AppendText(text);
2705                }
2706                catch (ObjectDisposedException)
2707                {
2708                    MessageBox.Show("ObjectDisposedException␣in␣
                            ConsoleAppendText()");
2709                    m_GuiLocked = false;
2710
2711                    return;
2712                }
```

```
2713            }
2714
2715            m_GuiLocked = false;
2716        }
2717
2718        private void buttonRunExecution_Click(object sender, EventArgs e)
2719        {
2720            if (m_GuiLocked)
2721                return;
2722
2723            if (workerObject != null && workerObject.workerThread.IsAlive)
2724            {
2725                if (workerObject.debug)
2726                {
2727                    workerObject.StopDebug(); // stop debug mode
2728                    return;
2729                }
2730                else if (workerObject.paused)
2731                {
2732                    workerObject.RequestPause(); // resume from pause
2733                    return;
2734                }
2735                else
2736                    return;
2737            }
2738
2739            CreateExecutionThread();
2740        }
2741
2742        private void buttonPauseExecution_Click(object sender, EventArgs e)
2743        {
2744            if (m_GuiLocked)
2745                return;
2746
2747            if (workerObject == null || !workerObject.workerThread.IsAlive)
2748                return;
2749
2750            // Request that the worker thread pause or resume
2751            workerObject.RequestPause();
2752        }
2753
```

```
2754        private void buttonStopExecution_Click(object sender, EventArgs e)
2755        {
2756            if (m_GuiLocked)
2757                return;
2758
2759            if (workerObject == null || !workerObject.workerThread.IsAlive)
2760                return;
2761
2762            // Request that the worker thread stop itself (async call to avoid a
                    deadlock in subsequent Join())
2763            AsyncRequestThreadStopCaller caller = new
                    AsyncRequestThreadStopCaller(workerObject.RequestStop);
2764            caller.BeginInvoke(null, null);
2765
2766            // Use the Join method to block the current thread
2767            // until the object's thread terminates.
2768            workerObject.workerThread.Join();
2769            ConsoleAppendText("Execution␣thread␣has␣terminated.\r\n");
2770        }
2771
2772        private void buttonDebugExecution_Click(object sender, EventArgs e)
2773        {
2774            if (m_GuiLocked)
2775                return;
2776
2777            // If the execution thread is not running, then start it
2778            if (workerObject == null || !workerObject.workerThread.IsAlive)
2779                CreateExecutionThread();
2780
2781            // Request that the worker thread put itself into step-by-step
                    execution mode
2782            workerObject.RequestDebug();
2783        }
2784
2785        private void CreateExecutionThread()
2786        {
2787            // Return on empty execution plan
2788            if (m_TestPlan.Count <= 0)
2789                return;
2790
```

```
2791            // Make sure the test case selected in panelPlan is displayed and
                    not the
2792            // test case loaded in panelDesign (if there is one)
2793            LoadTestCaseAtIndex(m_CurrentPlanIndex);
2794
2795            // Start the worker thread.
2796            ConsoleAppendText("Creating␣a␣new␣execution␣thread...\r\n");
2797
2798            workerObject = new ExecutionThread(this, m_TestPlan,
                    ReportLevel, TesterName, OnFailure);
2799
2800            // Loop until worker thread activates.
2801            while (!workerObject.workerThread.IsAlive) ;
2802
2803            // Put the main thread to sleep for 1 millisecond to
2804            // allow the worker thread to do some work
2805            Thread.Sleep(1);
2806        }
2807
2808        private void textTestPlanName_TextChanged(object sender,
                EventArgs e)
2809        {
2810            if (m_GuiLocked)
2811                return;
2812
2813            m_TestPlan.Name = textTestPlanName.Text;
2814
2815            m_TestPlan.NeedToSave = true;
2816        }
2817
2818        private void panelDesign_Resize(object sender, EventArgs e)
2819        {
2820            if (m_GuiLocked)
2821                return;
2822
2823            // Adjust the width for the listview listViewTestSteps columns
2824            if (panelDesign.Width − 55 > 0)
2825                if (listViewTestSteps.Columns.Count > 1)
2826                    listViewTestSteps.Columns[1].Width = panelDesign.
                            Width − 55;
2827
```

```
2828            return;
2829        }
2830
2831        private void panelPlan_Resize(object sender, EventArgs e)
2832        {
2833            if (m_GuiLocked)
2834                return;
2835
2836            // Adjust the width for the listview listViewTestPlan columns
2837            if (listViewTestPlan.Columns.Count > 0)
2838                listViewTestPlan.Columns[0].Width = panelPlan.Width;
2839
2840            // Adjust the width for the listview listViewTestStepsForPlan
                    columns
2841            if (listViewTestStepsForPlan.Columns.Count > 0)
2842                listViewTestStepsForPlan.Columns[0].Width = 50;
2843            if (listViewTestStepsForPlan.Columns.Count > 1)
2844                listViewTestStepsForPlan.Columns[1].Width = panelPlan.
                        Width - 55;
2845
2846            return;
2847        }
2848
2849        private void comboBoxSendCommand_SelectedIndexChanged(object
                sender, EventArgs e)
2850        {
2851            // The description of each item in the combobox MUST end with the
                    corresponding
2852            // hex command within parentheses. For example, "CAMERA
                    RESET (0x37)",
2853            // "CAMERA START (0x38)" and so on...
2854
2855            int length;
2856            string combobox_description, command_text;
2857
2858            if (m_GuiLocked)
2859                return;
2860
2861            SendCommand sendCommand = (SendCommand)m_TestCase[
                    m_CurrentStepIndex];
2862
```

```
2863            // Get the hex command from the combobox description (see above)
2864            combobox_description = comboBoxSendCommand.SelectedItem.
                    ToString();
2865            length = combobox_description.Length;
2866            if (length >= 3)
2867            {
2868                command_text = combobox_description.Substring(length - 3,
                        3).Substring(0, 2);
2869
2870                if (command_text == "..")
2871                    sendCommand.SetHexCommand("00"); // Command "0
                            x00" is reserved for custom command
2872                else
2873                    sendCommand.SetHexCommand(command_text); // Set
                            the hex command
2874            }
2875
2876            // Set up or update a suitable tooltip for the command parameters
                    textbox
2877            toolTipCommandParameters.SetToolTip(this.
                    textCommandParameters, SelectCommandParametersToolTip(
                    sendCommand.GetHexCommand()));
2878
2879            m_TestCase.NeedToSave = true;
2880        }
2881
2882        private void textCommandParameters_TextChanged(object sender,
                EventArgs e)
2883        {
2884            if (m_GuiLocked)
2885                return;
2886
2887            SendCommand sendCommand = (SendCommand)m_TestCase[
                    m_CurrentStepIndex];
2888            sendCommand.SetHexParameters(textCommandParameters.Text);
2889
2890            m_TestCase.NeedToSave = true;
2891        }
2892
2893        private void comboBoxExpectedAnswer_SelectedIndexChanged(
                object sender, EventArgs e)
```

```
2894        {
2895            int index;
2896
2897            if (m_GuiLocked)
2898                return;
2899
2900            SendCommand sendCommand = (SendCommand)m_TestCase[
                    m_CurrentStepIndex];
2901            index = comboBoxExpectedAnswer.SelectedIndex;
2902
2903            if (index >= 0 && index <= (int)StatusResponse.NoCheck)
2904                sendCommand.SetExpectedStatus((StatusResponse)index);
2905
2906            m_TestCase.NeedToSave = true;
2907        }
2908
2909        private void linkLabel1_LinkClicked(object sender,
                LinkLabelLinkClickedEventArgs e)
2910        {
2911            if (m_GuiLocked)
2912                return;
2913
2914            System.Diagnostics.Process.Start("http://" + this.linkLabel1.Text);
2915        }
2916
2917        private void listViewTestSteps_KeyUp(object sender, KeyEventArgs e
                )
2918        {
2919            int m_CurrentStepIndex_old = m_CurrentStepIndex;
2920
2921            if (m_GuiLocked)
2922                return;
2923
2924            // The OS does not support scrolling on the listview using
                    PageDown/PageUp
2925            if (e.KeyCode == Keys.PageDown)
2926            {
2927                if (listViewTestSteps.SelectedIndices.Count > 0)
2928                {
2929                    foreach (int i in listViewTestSteps.SelectedIndices)
2930                        if (i < listViewTestSteps.Items.Count)
```

```
2931                            listViewTestSteps.Items[i].Selected = false;
2932                    listViewTestSteps.Items[listViewTestSteps.Items.Count −
                            1].Selected = true;
2933                    listViewTestSteps.Items[listViewTestSteps.Items.Count −
                            1].Focused = true;
2934                }
2935            }
2936            else if (e.KeyCode == Keys.PageUp)
2937            {
2938                if (listViewTestSteps.SelectedIndices.Count > 0)
2939                {
2940                    foreach (int i in listViewTestSteps.SelectedIndices)
2941                        if (i < listViewTestSteps.Items.Count)
2942                            listViewTestSteps.Items[i].Selected = false;
2943                    if (listViewTestSteps.Items.Count > 0)
2944                    {
2945                        listViewTestSteps.Items[0].Selected = true;
2946                        listViewTestSteps.Items[0].Focused = true;
2947                    }
2948                }
2949            }
2950
2951            if (listViewTestSteps.SelectedIndices.Count > 0)
2952                m_CurrentStepIndex = listViewTestSteps.SelectedIndices[0];
2953            else
2954            {
2955                m_CurrentStepIndex = −1;
2956                return;
2957            }
2958
2959            if (m_CurrentStepIndex != m_CurrentStepIndex_old)
2960            {
2961                // Show the test case and the step details
2962                ShowTestCase();
2963                ShowStepDetails();
2964            }
2965        }
2966
2967        private void listViewTestPlan_KeyUp(object sender, KeyEventArgs e)
2968        {
2969            int m_CurrentPlanIndex_old = m_CurrentPlanIndex;
```

```
2970
2971            if (m_GuiLocked)
2972                return;
2973
2974            // The OS does not support scrolling on the listview using
                    PageDown/PageUp
2975            if (e.KeyCode == Keys.PageDown)
2976            {
2977                if (listViewTestPlan.SelectedIndices.Count > 0)
2978                {
2979                    foreach (int i in listViewTestPlan.SelectedIndices)
2980                        if (i < listViewTestPlan.Items.Count)
2981                            listViewTestPlan.Items[i].Selected = false;
2982                    listViewTestPlan.Items[listViewTestPlan.Items.Count -
                            1].Selected = true;
2983                    listViewTestPlan.Items[listViewTestPlan.Items.Count -
                            1].Focused = true;
2984                }
2985            }
2986            else if (e.KeyCode == Keys.PageUp)
2987            {
2988                if (listViewTestPlan.SelectedIndices.Count > 0)
2989                {
2990                    foreach (int i in listViewTestPlan.SelectedIndices)
2991                        if (i < listViewTestPlan.Items.Count)
2992                            listViewTestPlan.Items[i].Selected = false;
2993                    if (listViewTestPlan.Items.Count > 0)
2994                    {
2995                        listViewTestPlan.Items[0].Selected = true;
2996                        listViewTestPlan.Items[0].Focused = true;
2997                    }
2998                }
2999            }
3000
3001            if (listViewTestPlan.SelectedIndices.Count > 0)
3002                m_CurrentPlanIndex = listViewTestPlan.SelectedIndices[0];
3003
3004            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    listViewTestPlan.Items.Count)
3005                iterator_selected_in_plan = (m_TestPlan[m_CurrentPlanIndex
                        ].GetType().Name == "TestIterator");
```

```
3006
3007            // load the selected Test Case
3008            if (m_CurrentPlanIndex != m_CurrentPlanIndex_old)
3009            {
3010                CheckTestCaseWantToSave();
3011
3012                // Load the selected test case (if there is one) and handle the
                        test step result change event
3013                LoadTestCaseAtIndex(m_CurrentPlanIndex);
3014
3015                // If there is at least one step, select it
3016                if (m_TestCase.Count > 0)
3017                    m_CurrentStepIndex = 0;
3018
3019                // Show the test case and the first step details
3020                ShowTestCase();
3021                ShowStepDetails();
3022
3023                if (listViewTestSteps.Items.Count > 0)
3024                    listViewTestSteps.Items[0].Selected = true; // GT
                            19092014: After having selected the first step, show it
                            as selected...
3025
3026                // Show the test case in the panelPlan (Tester View only)
3027                ShowTestCaseForPlan();
3028
3029                ShowPlanDetails();
3030            }
3031        }
3032
3033        private void listViewTestPlan_SelectedIndexChanged(object sender,
                EventArgs e)
3034        {
3035            int m_CurrentPlanIndex_old = m_CurrentPlanIndex;
3036
3037            if (m_GuiLocked)
3038                return;
3039
3040            if (listViewTestPlan.SelectedIndices.Count > 0)
3041            {
3042                m_CurrentPlanIndex = listViewTestPlan.SelectedIndices[0];
```

```
3043
3044                if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        m_TestPlan.Count)
3045                    iterator_selected_in_plan = (m_TestPlan[
                            m_CurrentPlanIndex].GetType().Name == "
                            TestIterator");
3046            }
3047            else
3048            {
3049                m_CurrentPlanIndex = -1;
3050                iterator_selected_in_plan = false;
3051                menuConfigLoopIterations.Visible = false;
3052                menuConfigLoopSize.Visible = false;
3053                toolStripSeparatorLoopConfiguration.Visible = false;
3054
3055                ShowTestCase();
3056
3057                // Show the test case in the panelPlan (Tester View only)
3058                ShowTestCaseForPlan();
3059
3060                return;
3061            }
3062
3063            // If selected element is a TestIterator let the user edit its properties
3064            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
3065            {
3066                menuConfigLoopIterations.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3067                menuConfigLoopSize.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3068                toolStripSeparatorLoopConfiguration.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3069                if (m_TestPlan[m_CurrentPlanIndex].GetType().Name == "
                        TestIterator")
3070                {
3071                    toolStripTextBoxLoopIterations.Text = (((TestIterator)
                            m_TestPlan[m_CurrentPlanIndex]).LoopIterations).
                            ToString();
3072                    toolStripTextBoxLoopSize.Text = (((TestIterator)
                            m_TestPlan[m_CurrentPlanIndex]).LoopSize).
```

```
                    ToString();
3073            }
3074        }
3075
3076        if (m_CurrentPlanIndex != m_CurrentPlanIndex_old)
3077        {
3078            CheckTestCaseWantToSave();
3079
3080            // Load the selected test case (if there is one) and handle the
                    test step result change event
3081            LoadTestCaseAtIndex(m_CurrentPlanIndex);
3082
3083            // If there is at least one step, select it
3084            if (m_TestCase.Count > 0)
3085                m_CurrentStepIndex = 0;
3086
3087            // Show the test case and the first step details
3088            ShowTestCase();
3089            ShowStepDetails();
3090
3091            if (listViewTestSteps.Items.Count > 0)
3092                listViewTestSteps.Items[0].Selected = true; // GT
                        19092014: After having selected the first step, show it
                        as selected...
3093
3094            // Show the test case in the panelPlan (Tester View only)
3095            ShowTestCaseForPlan();
3096
3097            ShowPlanDetails();
3098        }
3099    }
3100
3101    private void textI2CAddress_TextChanged(object sender, EventArgs e
            )
3102    {
3103        if (m_GuiLocked)
3104            return;
3105
3106        if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                m_TestCase.Count)
3107        {
```

```
3108                SendCommand sendCommand = (SendCommand)m_TestCase
                        [m_CurrentStepIndex];
3109
3110                ushort UInt16fromHex;
3111
3112                if (textI2CAddress.Text.Length == 0)
3113                    return;
3114
3115                if (textI2CAddress.Text.Length > 2)
3116                {
3117                    MessageBox.Show("Invalid␣hexadecimal␣number␣for␣
                            the␣I2C␣Address␣!␣Using␣default␣value␣" +
                            Properties.Settings.Default.I2CAddress + ".");
3118                    textI2CAddress.Text = Properties.Settings.Default.
                            I2CAddress;
3119                    sendCommand.SetI2CAddress(textI2CAddress.Text);
3120                    return;
3121                }
3122
3123                // Controllare che sia una stringa esadecimale di due caratteri
                        e poi salvare nell'oggetto SendCommand
3124                try
3125                {
3126                    UInt16fromHex = UInt16.Parse(textI2CAddress.Text,
                            NumberStyles.AllowHexSpecifier);
3127                }
3128                catch (FormatException)
3129                {
3130                    MessageBox.Show("Invalid␣hexadecimal␣number␣for␣
                            the␣I2C␣Address␣!␣Using␣default␣value␣" +
                            Properties.Settings.Default.I2CAddress + ".");
3131                    textI2CAddress.Text = Properties.Settings.Default.
                            I2CAddress;
3132                    sendCommand.SetI2CAddress(textI2CAddress.Text);
3133                    return;
3134                }
3135
3136                sendCommand.SetI2CAddress(textI2CAddress.Text);
3137
3138                m_TestCase.NeedToSave = true;
3139            }
```

```
3140        }
3141
3142        private void toolStripTextBoxBitrate_TextChanged(object sender,
                EventArgs e)
3143        {
3144            ushort bitrate;
3145
3146            if (m_GuiLocked)
3147                return;
3148
3149            if (toolStripTextBoxBitrate.Text.Length == 0)
3150                return;
3151
3152            try
3153            {
3154                bitrate = UInt16.Parse(toolStripTextBoxBitrate.Text,
                        NumberStyles.None);
3155            }
3156            catch (FormatException)
3157            {
3158                MessageBox.Show("Invalid␣bitrate␣!");
3159                toolStripTextBoxBitrate.Text = Properties.Settings.Default.
                        I2CBitrate.ToString();
3160
3161                return;
3162            }
3163            catch (OverflowException)
3164            {
3165                MessageBox.Show("Invalid␣bitrate␣!");
3166                toolStripTextBoxBitrate.Text = Properties.Settings.Default.
                        I2CBitrate.ToString();
3167
3168                return;
3169            }
3170
3171            Properties.Settings.Default.I2CBitrate = bitrate;
3172        }
3173
3174        private void menuFrameGrabberSelect_Click(object sender,
                EventArgs e)
3175        {
```

```
3176                if (m_GuiLocked)
3177                    return;
3178
3179  #if (FRAMEGRABBER)
3180                int result;
3181
3182                result = Framegrabber.Select();
3183
3184                switch (result)
3185                {
3186                    case 0:
3187                        break;
3188                    case -1:
3189                        ConsoleAppendText("Capture␣Image:␣Framegrabber␣
                                device␣has␣not␣been␣selected.␣Cannot␣connect␣to
                                ␣a␣framegrabber␣device␣!\r\n");
3190                        break;
3191                    case -2:
3192                        ConsoleAppendText("Capture␣Image:␣Cannot␣get␣
                                framegrabber␣device␣information.\r\n");
3193                        break;
3194                    case -3:
3195                        ConsoleAppendText("Capture␣Image:␣Cannot␣connect␣
                                to␣the␣selected␣framegrabber␣device␣!\r\n");
3196                        break;
3197                    default:
3198                        ConsoleAppendText("Capture␣Image:␣Unknown␣error␣
                                during␣framegrabber␣device␣selection␣!\r\n");
3199                        break;
3200                }
3201  #endif
3202            }
3203
3204            private void toolStripTextBoxFGWidth_TextChanged(object sender,
                    EventArgs e)
3205            {
3206                ushort width;
3207
3208                if (m_GuiLocked)
3209                    return;
3210
```

```
3211            if (toolStripTextBoxFGWidth.Text.Length == 0)
3212                return;
3213
3214            try
3215            {
3216                width = UInt16.Parse(toolStripTextBoxFGWidth.Text,
                        NumberStyles.None);
3217            }
3218            catch (FormatException)
3219            {
3220                MessageBox.Show("Invalid␣framegrabber␣width␣!");
3221                toolStripTextBoxFGWidth.Text = DefaultFrameGrabberWidth.
                        ToString();
3222
3223                return;
3224            }
3225            catch (OverflowException)
3226            {
3227                MessageBox.Show("Invalid␣framegrabber␣width␣!");
3228                toolStripTextBoxFGWidth.Text = DefaultFrameGrabberWidth.
                        ToString();
3229
3230                return;
3231            }
3232
3233            FrameGrabberWidth = (long)width;
3234        }
3235
3236        private void toolStripTextBoxFGHeight_TextChanged(object sender,
                EventArgs e)
3237        {
3238            ushort height;
3239
3240            if (m_GuiLocked)
3241                return;
3242
3243            if (toolStripTextBoxFGHeight.Text.Length == 0)
3244                return;
3245
3246            try
3247            {
```

```
3248                height = UInt16.Parse(toolStripTextBoxFGHeight.Text,
                        NumberStyles.None);
3249            }
3250            catch (FormatException)
3251            {
3252                MessageBox.Show("Invalid␣framegrabber␣height␣!");
3253                toolStripTextBoxFGHeight.Text =
                        DefaultFrameGrabberHeight.ToString();
3254
3255                return;
3256            }
3257            catch (OverflowException)
3258            {
3259                MessageBox.Show("Invalid␣framegrabber␣width␣!");
3260                toolStripTextBoxFGHeight.Text =
                        DefaultFrameGrabberHeight.ToString();
3261
3262                return;
3263            }
3264
3265            FrameGrabberHeight = (long)height;
3266        }
3267
3268        private void reloadPlanMenu_Click(object sender, EventArgs e)
3269        {
3270            if (m_GuiLocked)
3271                return;
3272
3273            if (m_TestPlan.FileName != null && m_TestPlan.FileName !=
                    String.Empty) // Avoids ArgumentException in m_TestCase.
                    ReadXml().
3274            {
3275                try
3276                {
3277                    m_TestPlan.ReadXml(m_TestPlan.FileName);
3278                }
3279                catch (System.IO.FileNotFoundException)
3280                {
3281                    ConsoleAppendText("Error:␣test␣plan␣file␣" +
                            m_TestPlan.FileName + "␣cannot␣be␣found␣on␣the
                            ␣filesystem.\r\n");
```

```
3282                    return;
3283                }
3284                catch (System.Xml.XmlException)
3285                {
3286                    ConsoleAppendText("Error:␣unreadable␣XML␣in␣Test␣
                        Plan␣file␣" + m_TestPlan.FileName + ".\r\n");
3287                    return;
3288                }
3289
3290                Unregister_TestCase_ResultChanged_EventHandlers();
3291
3292                // Add an event handler to each test case so that it refreshes
                        listViewTestPlan
3293                // by calling ShowTestPlan()
3294                for (int i = 0; i < m_TestPlan.Count; i++)
3295                    if (m_TestPlan[i].GetType().Name == "TestCase")
3296                    {
3297                        TestCase testCase = (TestCase)m_TestPlan[i];
3298                        testCase.ResultChanged += new
                                ResultChangedEventHandler(
                                TestCase_ResultChanged);
3299                    }
3300
3301                // Show it
3302                ShowTestPlan();
3303                ShowPlanDetails();
3304            }
3305            else // A test plan has not been loaded but there might be a new test
                        plan being edited...
3306                for (int i = 0; i < m_TestPlan.Count; i++)
3307                    if (m_TestPlan[i].GetType().Name == "TestCase")
3308                    {
3309                        TestCase testCase = (TestCase)m_TestPlan[i];
3310                        if (testCase.FileName != null && testCase.FileName
                                != String.Empty)
3311                        {
3312                            try
3313                            {
3314                                testCase.ReadXml(testCase.FileName);
3315                            }
```

```
3316                            catch (System.IO.FileNotFoundException) // It
                                    might need to remove the test case from the
                                    test plan
3317                            {
3318                                ConsoleAppendText("Error: test case file
                                        " + testCase.FileName + " cannot be
                                        found on the filesystem.\r\n");
3319                                continue;
3320                            }
3321                            catch (System.Xml.XmlException)
3322                            {
3323                                ConsoleAppendText("Error: unreadable
                                        XML in Test Case file " + testCase.
                                        FileName + " at position index " + (i
                                        + 1) + ".\r\n");
3324                                continue;
3325                            }
3326                        }
3327                    }
3328
3329                // Reload the currently selected test case and register its test steps
                        result
3330                // changed event handlers
3331                if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        m_TestPlan.Count)
3332                    LoadTestCaseAtIndex(m_CurrentPlanIndex);
3333            }
3334
3335            private void trackBarSleep_ValueChanged(object sender, EventArgs e
                    )
3336            {
3337                if (m_GuiLocked)
3338                    return;
3339
3340                if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                        m_TestCase.Count)
3341                {
3342                    Sleep sleep = (Sleep)m_TestCase[m_CurrentStepIndex];
3343                    sleep.Duration = trackBarSleep.Value;
3344                    labelSleep.Text = trackBarSleep.Value.ToString() + " ms";
3345                    textBoxSleepDuration.Text = trackBarSleep.Value.ToString();
```

```
3346
3347                m_TestCase.NeedToSave = true;
3348            }
3349        }
3350
3351        private void textBoxSleepDuration_TextChanged(object sender,
                EventArgs e)
3352        {
3353            int duration;
3354
3355            if (m_GuiLocked)
3356                return;
3357
3358            if (m_CurrentStepIndex >= 0 && m_CurrentStepIndex <
                    m_TestCase.Count)
3359            {
3360                Sleep sleep = (Sleep)m_TestCase[m_CurrentStepIndex];
3361                if (textBoxSleepDuration.Text.Length > 0)
3362                {
3363                    try
3364                    {
3365                        duration = UInt16.Parse(textBoxSleepDuration.Text,
                                NumberStyles.None);
3366                    }
3367                    catch (FormatException)
3368                    {
3369                        MessageBox.Show("Invalid␣sleep␣duration␣!");
3370                        return;
3371                    }
3372
3373                    sleep.Duration = duration;
3374                    trackBarSleep.Value = duration;
3375                    labelSleep.Text = duration.ToString() + "␣ms";
3376
3377                    m_TestCase.NeedToSave = true;
3378                }
3379            }
3380        }
3381
3382        private void listViewTestSteps_MouseClick(object sender,
                MouseEventArgs e)
```

```
3383        {
3384            int m_CurrentStepIndex_old = m_CurrentStepIndex;
3385
3386            if (m_GuiLocked)
3387                return;
3388
3389            // Determine the current index
3390            if (listViewTestSteps.SelectedIndices.Count > 0)
3391                m_CurrentStepIndex = listViewTestSteps.SelectedIndices[0];
3392            else
3393                m_CurrentStepIndex = -1;
3394
3395            if (m_CurrentStepIndex_old != m_CurrentStepIndex)
3396            {
3397                // Show the test case and the step details
3398                ShowTestCase();
3399                ShowStepDetails();
3400            }
3401        }
3402
3403        private void textTesterName_TextChanged(object sender, EventArgs e
                )
3404        {
3405            if (m_GuiLocked)
3406                return;
3407
3408            TesterName = textTesterName.Text;
3409        }
3410
3411        private void generateReportCombo_SelectedIndexChanged(object
                sender, EventArgs e)
3412        {
3413            if (m_GuiLocked)
3414                return;
3415
3416            if (generateReportCombo.SelectedIndex == 1)
3417                ReportLevel = report_level.Enabled;
3418            else
3419                ReportLevel = report_level.Disabled;
3420        }
3421
```

```
3422        private void onFailCombo_SelectedIndexChanged(object sender,
                EventArgs e)
3423        {
3424            if (m_GuiLocked)
3425                return;
3426
3427            if (onFailCombo.SelectedIndex == 0)
3428                OnFailure = on_failure.Ask;
3429            else if (onFailCombo.SelectedIndex == 1)
3430                OnFailure = on_failure.Break;
3431            else
3432                OnFailure = on_failure.Continue;
3433        }
3434
3435        private void toolStripTextBoxLoopIterations_TextChanged(object
                sender, EventArgs e)
3436        {
3437            int LoopIterations;
3438
3439            if (m_GuiLocked)
3440                return;
3441
3442            if (toolStripTextBoxLoopIterations.Text.Length > 0)
3443            {
3444                try
3445                {
3446                    LoopIterations = UInt16.Parse(
                            toolStripTextBoxLoopIterations.Text, NumberStyles.
                            None);
3447                }
3448
3449                catch (FormatException)
3450                {
3451                    return;
3452                }
3453
3454                if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                        m_TestPlan.Count)
3455                    if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                            "TestIterator")
3456                    {
```

```
3457                        ((TestIterator)m_TestPlan[m_CurrentPlanIndex]).
                                LoopIterations = LoopIterations;
3458                        if (listViewTestPlan.SelectedIndices.Count > 0)
3459                            listViewTestPlan.Items[listViewTestPlan.
                                    SelectedIndices[0]].Text = ((TestIterator)
                                    m_TestPlan[m_CurrentPlanIndex]).ToString
                                    ();
3460                    }
3461                }
3462            }
3463
3464            private void toolStripTextBoxLoopSize_TextChanged(object sender,
                    EventArgs e)
3465            {
3466                int LoopSize;
3467
3468                if (m_GuiLocked)
3469                    return;
3470
3471                if (toolStripTextBoxLoopSize.Text.Length > 0)
3472                {
3473                    try
3474                    {
3475                        LoopSize = UInt16.Parse(toolStripTextBoxLoopSize.Text,
                                NumberStyles.None);
3476                    }
3477
3478                    catch (FormatException)
3479                    {
3480                        return;
3481                    }
3482
3483                    if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                            m_TestPlan.Count)
3484                        if (m_TestPlan[m_CurrentPlanIndex].GetType().Name ==
                                "TestIterator")
3485                        {
3486                            ((TestIterator)m_TestPlan[m_CurrentPlanIndex]).
                                    LoopSize = LoopSize;
3487                        if (listViewTestPlan.SelectedIndices.Count > 0)
```

```
3488                                listViewTestPlan.Items[listViewTestPlan.
                                        SelectedIndices[0]].Text = ((TestIterator)
                                        m_TestPlan[m_CurrentPlanIndex]).ToString
                                        ();
3489                            }
3490                        }
3491                    }
3492
3493                    private void pictureBox1_MouseDown(object sender,
                            MouseEventArgs e)
3494                    {
3495                        if (m_GuiLocked)
3496                            return;
3497
3498                        if (e.Button == System.Windows.Forms.MouseButtons.Left)
3499                        {
3500                            BitmapSelectionXStart = e.X;
3501                            BitmapSelectionYStart = e.Y;
3502                        }
3503                    }
3504
3505                    private void pictureBox1_MouseUp(object sender, MouseEventArgs e
                            )
3506                    {
3507                        if (m_GuiLocked)
3508                            return;
3509
3510                        if (e.Button == System.Windows.Forms.MouseButtons.Left)
3511                        {
3512                            // Create a rectangle object which corresponds to the selected
                                    region of the background image
3513                            Rectangle selectionRect = new Rectangle(Math.Min(
                                    BitmapSelectionXStart, e.X), Math.Min(
                                    BitmapSelectionYStart, e.Y), Math.Abs(e.X -
                                    BitmapSelectionXStart), Math.Abs(e.Y -
                                    BitmapSelectionYStart));
3514
3515                            // Do not accept a region which intersects other regions
3516                            if (!SelectionIntersect(selectionRect))
3517                            {
```

```
3518                    // When the mouse click is released, draw the selection
                            rectangle permanently on
3519                    // the Bitmap otherwise the selected region will be lost
                            when selecting
3520                    // the next region
3521                    m_PictureGraphics.DrawRectangle(selectionPen,
                            selectionRect);
3522
3523                    // Add the selection rectangle to a list of selected regions
                            of the background image
3524                    selectionList.Add(selectionRect);
3525                }
3526
3527                pictureBox1.Refresh();
3528            }
3529        }
3530
3531        private void pictureBox1_MouseMove(object sender,
                MouseEventArgs e)
3532        {
3533            if (m_GuiLocked)
3534                return;
3535
3536            // If the left button on the mouse is clicked, then the
3537            // user is selecting a region of the image
3538            if (e.Button == System.Windows.Forms.MouseButtons.Left)
3539            {
3540                imageSelectionRect = new Rectangle(Math.Min(
                        BitmapSelectionXStart, e.X), Math.Min(
                        BitmapSelectionYStart, e.Y), Math.Abs(
                        BitmapSelectionXStart - e.X), Math.Abs(
                        BitmapSelectionYStart - e.Y));
3541                Invalidate(imageSelectionRect);
3542
3543                pictureBox1.Refresh();
3544            }
3545
3546            // Make the toolStripMenuItem to remove the selected area not
3547            // visible by default (contextMenu)
3548            toolStripMenuRemoveSelection.Visible = false;
3549
```

```
3550            // By default no region is selected
3551            m_CurrentSelectedRegion = -1;
3552
3553            // If the mouse position is within one of the selected regions, then
3554            // make the toolStripMenuItem visible in the contextMenu
3555            for (int i = 0; i < selectionList.Count; i++)
3556                if ((e.X > selectionList[i].X && e.X < selectionList[i].X +
                        selectionList[i].Width) && (e.Y > selectionList[i].Y && e
                        .Y < selectionList[i].Y + selectionList[i].Height))
3557                {
3558                    m_CurrentSelectedRegion = i;
3559                    toolStripMenuRemoveSelection.Visible = true;
3560                }
3561        }
3562
3563        private void pictureBox1_Paint(object sender, PaintEventArgs e)
3564        {
3565            if (m_GuiLocked)
3566                return;
3567
3568            if (pictureBox1.Image != null)
3569            {
3570                // Keep drawing the selected region of interest while the mouse
                        is clicked
3571                e.Graphics.DrawRectangle(selectionPen, imageSelectionRect);
3572                // Once the selected region of interest has been drawn, do not
                        keep
3573                // drawing it again and again !
3574                imageSelectionRect.Width = 0;
3575                imageSelectionRect.Height = 0;
3576            }
3577        }
3578
3579        // When the picturebox is resized the selection should be cleared
3580        private void pictureBox1_Resize(object sender, EventArgs e)
3581        {
3582            if (m_GuiLocked)
3583                return;
3584
3585            // Create a new foreground image to hold the regions of interest
3586            if (m_ForeImage != null)
```

```
3587                m_ForeImage.Dispose();
3588            if (m_PictureGraphics != null)
3589                m_PictureGraphics.Dispose();
3590            m_ForeImage = new Bitmap(pictureBox1.Width, pictureBox1.
                    Height, System.Drawing.Imaging.PixelFormat.
                    Format32bppArgb);
3591            m_PictureGraphics = Graphics.FromImage(m_ForeImage);
3592
3593            // Display the new foreground image in the picturebox
3594            pictureBox1.Image = m_ForeImage;
3595
3596            if (selectionList.Count > 0)
3597            {
3598                ConsoleAppendText("Image␣selection␣is␣lost␣after␣window
                        ␣resize...\r\n");
3599                selectionList.Clear();
3600            }
3601
3602            if (scaledselectionList.Count > 0)
3603                scaledselectionList.Clear();
3604
3605            pictureBox1.Refresh();
3606        }
3607
3608        private void toolStripMenuRemoveSelection_Click(object sender,
                EventArgs e)
3609        {
3610            if (m_GuiLocked)
3611                return;
3612
3613            if (m_CurrentSelectedRegion >= 0 && m_CurrentSelectedRegion
                    < selectionList.Count)
3614            {
3615                // Remove the selected region from the list of regions of interest
3616                selectionList.RemoveAt(m_CurrentSelectedRegion);
3617
3618                m_CurrentSelectedRegion = -1;
3619                toolStripMenuRemoveSelection.Visible = false;
3620
3621                // Create a new foreground image to hold the regions of
                        interest
```

```
3622                if (m_ForeImage != null)
3623                    m_ForeImage.Dispose();
3624                if (m_PictureGraphics != null)
3625                    m_PictureGraphics.Dispose();
3626                m_ForeImage = new Bitmap(pictureBox1.Width, pictureBox1.
                        Height, System.Drawing.Imaging.PixelFormat.
                        Format32bppArgb);
3627                m_PictureGraphics = Graphics.FromImage(m_ForeImage);
3628
3629                // Display the new foreground image in the picturebox
3630                pictureBox1.Image = m_ForeImage;
3631
3632                // Redraw the regions of interest
3633                for (int i = 0; i < selectionList.Count; i++)
3634                    m_PictureGraphics.DrawRectangle(selectionPen,
                            selectionList[i]);
3635
3636                pictureBox1.Refresh();
3637            }
3638        }
3639
3640        // Determine whether a given Rectangle intesects any of the currently
                selected
3641        // region of interests (Rectangle items in the selectionList list)
3642        private bool SelectionIntersect(Rectangle selectionRect)
3643        {
3644            bool intersectionfound = false;
3645
3646            for (int i = 0; i < selectionList.Count; i++)
3647            {
3648                Region region = new Region(selectionRect);
3649                region.Intersect(selectionList[i]);
3650                intersectionfound = !region.IsEmpty(m_PictureGraphics);
3651                if (region != null)
3652                    region.Dispose();
3653                if (intersectionfound)
3654                    break;
3655            }
3656
3657            return (intersectionfound);
3658        }
```

```
3659
3660            // The following event is triggered when a new image has been captured
                    by the
3661            // framegrabber.
3662            private void Framegrabber_ImageChanged(object sender, EventArgs e
                    )
3663            {
3664                if (m_GuiLocked)
3665                    return;
3666
3667    #if (FRAMEGRABBER) // The Framegrabber object is not defined when not
            using a framegrabber
3668                SetBackgroundImage(Framegrabber.image);
3669    #endif
3670
3671                pictureBox1.BackgroundImage = (System.Drawing.Image)
                        m_BackImage.Clone();
3672            }
3673
3674            private void SetBackgroundImage(Bitmap image)
3675            {
3676                if (image != null)
3677                    m_BackImage = (System.Drawing.Bitmap)image.Clone();
3678                else
3679                {
3680                    try
3681                    {
3682                        m_BackImage = new Bitmap("SETP_image.png");
3683                    }
3684                    catch (ArgumentException)
3685                    {
3686                        m_BackImage = null;
3687                    }
3688                }
3689            }
3690
3691            private void pictureBox1_BackgroundImageChanged(object sender,
                    EventArgs e)
3692            {
3693                if (m_GuiLocked)
3694                    return;
```

```
3695
3696            // This call needs to be asynchronous otherwise it causes a
                    deadlock when the execution
3697            // thread is stopped from the main thread.
3698            if (this.pictureBox1.InvokeRequired)
3699            {
3700                GenericEventCallback callback = new GenericEventCallback(
                        pictureBox1_BackgroundImageChanged);
3701                this.BeginInvoke(callback, new object[] { sender, e });
3702            }
3703            else
3704            {
3705                pictureBox1.Refresh();
3706            }
3707        }
3708
3709        private void testDesignMenu_Click(object sender, EventArgs e)
3710        {
3711            if (m_GuiLocked)
3712                return;
3713
3714            // Enable/disable the test case save menu item
3715            saveTestMenu.Enabled = m_TestCase.NeedToSave;
3716        }
3717
3718        private void testPlanMenu_Click(object sender, EventArgs e)
3719        {
3720            if (m_GuiLocked)
3721                return;
3722
3723            // If a test plan is loaded or if there is at least one test case in the
3724            // listview, enable the reload item in the toolstrip menu
3725            if ((m_TestPlan.FileName != null && m_TestPlan.FileName !=
                    String.Empty) || m_TestPlan.Count > 0)
3726                reloadPlanMenu.Enabled = true;
3727            else
3728                reloadPlanMenu.Enabled = false;
3729
3730            // Enable/disable the test plan save menu item
3731            savePlanMenu.Enabled = m_TestPlan.NeedToSave;
3732
```

```
3733            // Enable/disable the test plan save as menu item
3734            saveAsPlanMenu.Enabled = m_TestPlan.Count > 0;
3735        }
3736
3737        private void contextMenuTestPlan_Opening(object sender,
                    CancelEventArgs e)
3738        {
3739            if (m_GuiLocked)
3740                return;
3741
3742            // Enable or disable the loop configuration
3743            if (m_CurrentPlanIndex >= 0 && m_CurrentPlanIndex <
                    m_TestPlan.Count)
3744            {
3745                menuConfigLoopIterations.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3746                menuConfigLoopSize.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3747                toolStripSeparatorLoopConfiguration.Visible = (m_TestPlan[
                        m_CurrentPlanIndex].GetType().Name == "TestIterator");
3748            }
3749            else
3750            {
3751                menuConfigLoopIterations.Visible = false;
3752                menuConfigLoopSize.Visible = false;
3753                toolStripSeparatorLoopConfiguration.Visible = false;
3754            }
3755
3756            // If there are no test cases, then hide the "Remove" option from the
3757            // contextual menu in panelPlan.
3758            removeTestMenu.Visible = m_TestPlan.Count > 0;
3759
3760            // If there are less than two test cases, then hide the "Move Up", "
                    Move Down",
3761            // "Move to Top" and "Move to Bottom" options from the contextual
                     menu in
3762            // panelPlan.
3763            moveUpTestMenu.Visible = m_TestPlan.Count > 1;
3764            moveDownTestMenu.Visible = m_TestPlan.Count > 1;
3765            moveToTopTestMenu.Visible = m_TestPlan.Count > 1;
3766            moveToBottomTestMenu.Visible = m_TestPlan.Count > 1;
```

```
3767            }
3768
3769            private void contextMenuStepList_Opening(object sender,
                    CancelEventArgs e)
3770            {
3771                bool stepSelected;
3772                int index;
3773
3774                if (m_GuiLocked)
3775                    return;
3776
3777                if (m_DesignerView)
3778                {
3779                    index = -1;
3780                    if (listViewTestSteps.SelectedIndices.Count > 0)
3781                        index = listViewTestSteps.SelectedIndices[0];
3782                    stepSelected = (index >= 0 && index < listViewTestSteps.
                            Items.Count);
3783                    removeStepMenu.Visible = stepSelected;
3784                    insertStepMenu.Visible = stepSelected;
3785                    appendStepMenu.Visible = true;
3786                }
3787                else
3788                {
3789                    removeStepMenu.Visible = false;
3790                    insertStepMenu.Visible = false;
3791                    appendStepMenu.Visible = false;
3792                }
3793            }
3794        }
3795    }
```

Appendice D

Codice sorgente per il modello dei dati

```
1  ï»¿using System;
2  using System.Data;
3  using System.Collections;
4  using ScanEngineTestProgram;
5  using System.IO;
6
7  // A delegate type for hooking up test step and test case result change
       notifications.
8  public delegate void ResultChangedEventHandler(object sender, EventArgs e);
9
10 public class HexString
11 {
12     private string m_String = null;
13
14     public bool SetString(string currentString)
15     {
16         char[] chArray = currentString.ToCharArray();
17
18         // length is odd, append "0"
19         if (((currentString.Length) & 1) != 0)
20             currentString = "0" + currentString;
21
22         foreach (char ch in chArray)
23         {
24             if (((ch >= '0') && (ch <= '9')) ||
```

```
25                      ((ch >= 'A') && (ch <= 'F')) ||
26                      ((ch >= 'a') && (ch <= 'f')))
27                  {
28                      // acceptable character, continue
29                      continue;
30                  }
31                  else
32                  {
33                      // wrong character
34                      return false;
35                  }
36              }
37
38              m_String = currentString;
39
40              return (true);
41          }
42
43          public string GetString()
44          {
45              return m_String;
46          }
47
48          public byte[] GetBytes()
49          {
50              if (m_String == null) return(null);
51              if (m_String.Length == 0) return (null);
52
53              byte[] tempBytes = new byte[m_String.Length / 2];
54              for (int i = 0; i < m_String.Length; i+=2)
55              {
56                  tempBytes[i / 2] = ConvertToByte(m_String, i);
57              }
58              return tempBytes;
59          }
60
61          private byte ConvertToByte(string s, int index)
62          {
63              char ch1 = s[index];
64              char ch2 = s[index+1];
```

```
65              int val = ConvertHexCharToInt(ch1) * 16 + ConvertHexCharToInt(ch2)
                    ;
66              return (byte)val;
67          }
68
69          private int ConvertHexCharToInt(char ch)
70          {
71              switch (ch)
72              {
73                  case '0': return 0;
74                  case '1': return 1;
75                  case '2': return 2;
76                  case '3': return 3;
77                  case '4': return 4;
78                  case '5': return 5;
79                  case '6': return 6;
80                  case '7': return 7;
81                  case '8': return 8;
82                  case '9': return 9;
83                  case 'a':
84                  case 'A': return 10;
85                  case 'b':
86                  case 'B': return 11;
87                  case 'c':
88                  case 'C': return 12;
89                  case 'd':
90                  case 'D': return 13;
91                  case 'e':
92                  case 'E': return 14;
93                  case 'f':
94                  case 'F': return 15;
95                  default: return 0;
96              }
97          }
98      }
99
100     // "NoCheck" means that it will always succeed
101     public enum StatusResponse { Ack = 0, Nack, ChecksumError, Watchdog,
            NoCheck };
102
103     public enum ImageAnalysis {
```

```
104         Brightness = 0,
105         BrightnessLoss,
106         BrightnessDistribution,
107         Contrast,
108         ContrastBalance,
109         PixelNoise,
110         Snr,
111         InterFrameNoise,
112         InterFrameBrightnessStability,
113         BrightSaturation,
114         DarkSaturation,
115         Blur,
116         AimVisibility };
117
118 public class HexMask
119 {
120     private string m_Mask = null;
121
122     public bool SetMask(string currentMask)
123     {
124         char[] chArray = currentMask.ToCharArray();
125
126         if (((currentMask.Length) & 1) != 0)
127         {
128             // length is odd, do not accept
129             return false;
130         }
131
132         for (int i = 0; i < currentMask.Length; i++)
133         {
134             char ch = chArray[i];
135
136             if (((ch >= '0') && (ch <= '9')) ||
137                 ((ch >= 'A') && (ch <= 'F')) ||
138                 ((ch >= 'a') && (ch <= 'f')) || (ch == 'X'))
139             {
140                 if (ch == 'X')
141                 {
142                     if ((i & 1) == 0)
143                     {
144                         // even index
```

```
145                         if (chArray[i + 1] == 'X')
146                         {
147                             // ok we can accept it
148                             continue;
149                         }
150                         else
151                         {
152                             // not an X pair
153                             return (false);
154                         }
155                     }
156                     else
157                     {
158                         // odd index
159                         if (chArray[i − 1] == 'X')
160                         {
161                             // ok we can accept it
162                             continue;
163                         }
164                         else
165                         {
166                             // not an X pair
167                             return (false);
168                         }
169                     }
170                 }
171                 else
172                 {
173                     continue;
174                 }
175             }
176             else
177             {
178                 // character wrong
179                 return false;
180             }
181         }
182         m_Mask = currentMask;
183         return (true);
184     }
185
```

```
186     public byte[] GetMask()
187     {
188         if (m_Mask == null) return (null);
189         if (m_Mask.Length == 0) return (null);
190         byte[] tempBytes = new byte[m_Mask.Length / 2];
191         for (int i = 0; i < m_Mask.Length; i += 2)
192         {
193             tempBytes[i / 2] = ConvertToByte(m_Mask, i);
194
195         }
196         return tempBytes;
197     }
198
199     public string GetString()
200     {
201         return (m_Mask);
202     }
203
204     private byte ConvertToByte(string s, int index)
205     {
206         char ch1 = s[index];
207         char ch2 = s[index + 1];
208         int val = ConvertHexMaskToInt(ch1) * 16 + ConvertHexMaskToInt(
                ch2);
209         return (byte)val;
210     }
211
212     private int ConvertHexMaskToInt(char ch)
213     {
214         switch (ch)
215         {
216             case '0': return 0;
217             case '1': return 1;
218             case '2': return 2;
219             case '3': return 3;
220             case '4': return 4;
221             case '5': return 5;
222             case '6': return 6;
223             case '7': return 7;
224             case '8': return 8;
225             case '9': return 9;
```

```
226                 case 'a':
227                 case 'A': return 10;
228                 case 'b':
229                 case 'B': return 11;
230                 case 'c':
231                 case 'C': return 12;
232                 case 'd':
233                 case 'D': return 13;
234                 case 'e':
235                 case 'E': return 14;
236                 case 'f':
237                 case 'F': return 15;
238                 case 'X': return -1;
239                 default: return 0;
240             }
241         }
242     }
243
244     public class TestStep
245     {
246         public string m_Description { get; set; }
247
248         public event ResultChangedEventHandler ResultChanged;
249
250         // Invoke the ResultChanged event; called whenever the test step result
                changes
251         protected virtual void OnResultChanged(EventArgs e)
252         {
253             if (ResultChanged != null)
254                 ResultChanged(this, e);
255         }
256
257         int m_Result;
258
259         bool result_changed;
260
261         public int Result {
262             get
263             {
264                 return m_Result;
265             }
```

```
266             set
267             {
268                 if (value == −1 || value == 0 || value == 1)
269                 {
270                     result_changed = false;
271                     if (m_Result != value)
272                         result_changed = true;
273
274                     m_Result = value;
275
276                     if (result_changed)
277                         OnResultChanged(EventArgs.Empty);
278                 }
279             }
280         } // The outcome of each step execution (not used in Designer View).
281
282         public string failure_reason { get; set; }
283
284         public TestStep()
285         {
286             m_Description = null;
287             m_Result = −1; // Not executed yet.
288             failure_reason = null;
289         }
290
291         virtual public void ConvertToDataRow(ref DataRow dataRow)
292         {
293             dataRow["StepType"] = this.GetType().Name;
294             dataRow["Description"] = m_Description;
295             return;
296         }
297
298         virtual public void ConvertFromDataRow(DataRow dataRow)
299         {
300             if (dataRow["Description"].GetType().Name != "DBNull")
301             {
302                 m_Description = (string)dataRow["Description"];
303             }
304             else
305             {
306                 m_Description = null;
```

```
307             }
308         }
309     }
310
311     public class Sleep : TestStep
312     {
313         int m_Duration;
314
315         public int Duration
316         {
317             get
318             {
319                 return m_Duration;
320             }
321             set
322             {
323                 if (value >= 0 && value <= 10000)
324                     m_Duration = value;
325             }
326         }
327
328         public Sleep()
329         {
330             m_Duration = 0;
331         }
332
333         public override string ToString()
334         {
335             if (m_Duration == 0)
336                 return "No␣Operation";
337             else
338                 return "Sleep␣" + m_Duration.ToString() + "␣ms";
339         }
340
341         public override void ConvertToDataRow(ref DataRow dataRow)
342         {
343             dataRow["StepType"] = this.GetType().Name;
344             dataRow["SleepDuration"] = m_Duration;
345
346             return;
347         }
```

```
348
349         public override void ConvertFromDataRow(DataRow dataRow)
350         {
351             // Get Duration
352             if (dataRow["SleepDuration"].GetType().Name != "DBNull")
353             {
354                 m_Duration = ((int)dataRow["SleepDuration"]);
355             }
356             else
357             {
358                 m_Duration = 0;
359             }
360         }
361     }
362
363     public class SendCommand : TestStep
364     {
365         HexString m_hexCommand; // The command as a hex byte
366         HexString m_hexParameters; // The command parameters as a hex string
367
368         // Lunghezza sempre 2 caratteri (1 byte)
369         HexString m_I2CAddress;
370         static HexString m_I2CDefaultAddress = new HexString();
371
372         // Expected answer from device
373         StatusResponse m_expStatus;
374
375         public static void SetI2CDefaultAddress(string i2CDefaultAddress)
376         {
377             m_I2CDefaultAddress.SetString(i2CDefaultAddress);
378         }
379
380         public SendCommand()
381         {
382             m_hexCommand = new HexString();
383             m_hexParameters = new HexString();
384             m_I2CAddress = new HexString();
385             m_I2CAddress.SetString(m_I2CDefaultAddress.GetString());
386             m_expStatus = StatusResponse.Ack;
387         }
388
```

```
389     public override string ToString()
390     {
391         if ((m_Description == null) || (m_Description == String.Empty))
392         {
393             return "Send Command";
394         }
395         else
396         {
397             return m_Description;
398         }
399     }
400
401     public void SetHexCommand(String hexCommand)
402     {
403         m_hexCommand.SetString(hexCommand);
404     }
405
406     public String GetHexCommand()
407     {
408         if (m_hexCommand != null)
409         {
410             return m_hexCommand.GetString();
411         }
412         else
413         {
414             return (null);
415         }
416     }
417
418     public void SetHexParameters(String hexParameters)
419     {
420         m_hexParameters.SetString(hexParameters);
421     }
422
423     public String GetHexParameters()
424     {
425         if (m_hexParameters != null)
426         {
427             return m_hexParameters.GetString();
428         }
429         else
```

```
430             {
431                 return (null);
432             }
433         }
434
435         public void SetI2CAddress(string i2cAddress)
436         {
437             m_I2CAddress.SetString(i2cAddress);
438         }
439
440         public string GetI2CAddress()
441         {
442             return m_I2CAddress.GetString();
443         }
444
445         public void SetExpectedStatus(StatusResponse expStatus)
446         {
447             m_expStatus = expStatus;
448         }
449
450         public StatusResponse GetExpectedStatus()
451         {
452             return m_expStatus;
453         }
454
455         public override void ConvertToDataRow(ref DataRow dataRow)
456         {
457             dataRow["StepType"] = this.GetType().Name;
458             dataRow["Description"] = m_Description;
459             dataRow["HexCommand"] = m_hexCommand.GetString();
460             dataRow["HexParameters"] = m_hexParameters.GetString();
461             dataRow["I2CAddress"] = m_I2CAddress.GetString();
462             dataRow["ExpectedStatus"] = m_expStatus;
463
464             return;
465         }
466
467         public override void ConvertFromDataRow(DataRow dataRow)
468         {
469             // Get Description
470             if (dataRow["Description"].GetType().Name != "DBNull")
```

```
471             {
472                 m_Description = (string)dataRow["Description"];
473             }
474             else
475             {
476                 m_Description = null;
477             }
478
479             // Get HexCommand
480             if (dataRow["HexCommand"].GetType().Name != "DBNull")
481             {
482                 m_hexCommand.SetString((string)dataRow["HexCommand"]);
483             }
484             else
485             {
486                 m_hexCommand.SetString(String.Empty);
487             }
488
489             // Get HexParameters
490             if (dataRow["HexParameters"].GetType().Name != "DBNull")
491             {
492                 m_hexParameters.SetString((string)dataRow["HexParameters"]);
493             }
494             else
495             {
496                 m_hexParameters.SetString(String.Empty);
497             }
498
499             // Get I2C Address
500             if (dataRow["I2CAddress"].GetType().Name != "DBNull")
501             {
502                 m_I2CAddress.SetString((string)dataRow["I2CAddress"]);
503             }
504             else
505             {
506                 m_I2CAddress.SetString(String.Empty);
507             }
508
509             // Get the Status
510             if (dataRow["ExpectedStatus"].GetType().Name != "DBNull")
511             {
```

```
512                 string s = (string)dataRow["ExpectedStatus"];
513
514                 if (s == "Ack")
515                 {
516                     m_expStatus = StatusResponse.Ack;
517                 }
518                 else if (s == "Nack")
519                 {
520                     m_expStatus = StatusResponse.Nack;
521                 }
522                 else if (s == "ChecksumError")
523                 {
524                     m_expStatus = StatusResponse.ChecksumError;
525                 }
526                 else if (s == "Watchdog")
527                 {
528                     m_expStatus = StatusResponse.Watchdog;
529                 }
530                 else if (s == "NoCheck")
531                 {
532                     m_expStatus = StatusResponse.NoCheck;
533                 }
534             }
535             else
536             {
537                 m_expStatus = StatusResponse.Ack;
538             }
539         }
540 }
541
542 public class LoadImage : TestStep
543 {
544     string m_FileName;
545
546     public string FileName
547     {
548         get { return m_FileName; }
549         set { m_FileName = value; }
550     }
551
552     bool m_SecondaryImage;
```

```
553
554         public bool SecondaryImage
555         {
556             get { return m_SecondaryImage; }
557             set { m_SecondaryImage = value; }
558         }
559
560         public LoadImage()
561         {
562             m_FileName = null;
563             m_SecondaryImage = false;
564         }
565
566         public override string ToString()
567         {
568             string secondaryImage;
569
570             if (m_SecondaryImage == true)
571                 secondaryImage = "Secondary␣";
572             else
573                 secondaryImage = "";
574
575             if ((m_Description == null) || (m_Description == String.Empty))
576                 if (m_FileName == null || m_FileName == String.Empty)
577                     return "Load␣" + secondaryImage + "Image␣From␣File␣(
                            unconfigured)";
578                 else
579                     return "Load␣" + secondaryImage + "Image␣From␣File";
580             else
581                 return m_Description;
582         }
583
584         public override void ConvertToDataRow(ref DataRow dataRow)
585         {
586             dataRow["StepType"] = this.GetType().Name;
587             dataRow["Description"] = m_Description;
588             dataRow["ImageFileName"] = m_FileName;
589             dataRow["SecondaryImage"] = m_SecondaryImage;
590
591             return;
592         }
```

```
593
594         public override void ConvertFromDataRow(DataRow dataRow)
595         {
596             // Get Description
597             if (dataRow["Description"].GetType().Name != "DBNull")
598             {
599                 m_Description = (string)dataRow["Description"];
600             }
601             else
602             {
603                 m_Description = null;
604             }
605
606             // Get Image File Name
607             if (dataRow["ImageFileName"].GetType().Name != "DBNull")
608             {
609                 m_FileName = ((string)dataRow["ImageFileName"]);
610             }
611             else
612             {
613                 m_FileName = (String.Empty);
614             }
615
616             // Get Secondary Image Flag
617             if (dataRow["SecondaryImage"].GetType().Name != "DBNull")
618             {
619                 m_SecondaryImage = ((bool)dataRow["SecondaryImage"]);
620             }
621             else
622             {
623                 m_SecondaryImage = false;
624             }
625         }
626 }
627
628 public class CaptureImage : TestStep
629 {
630         bool m_SaveImage;
631         string m_SaveFolder;
632         string m_FileName;
633         int m_frames;
```

```
634
635         public CaptureImage()
636         {
637             m_SaveImage = false;
638             m_SaveFolder = null;
639             m_FileName = null;
640             m_frames = 1;
641         }
642
643         public bool SaveImage
644         {
645             get { return m_SaveImage; }
646             set { m_SaveImage = value; }
647         }
648
649         public string SaveFolder
650         {
651             get { return m_SaveFolder; }
652             set { m_SaveFolder = value; }
653         }
654
655         public string FileName
656         {
657             get { return m_FileName; }
658             set { m_FileName = value; }
659         }
660
661         public int Frames
662         {
663             get
664             {
665                 if (SaveImage)
666                     return m_frames;
667                 else
668                     return 1;
669             }
670             set
671             {
672                 if (value > 0)
673                     m_frames = value;
674             }
```

```
675         }
676
677         public override string ToString()
678         {
679             if ((m_Description == null) || (m_Description == String.Empty))
680             {
681                 if (m_frames == 1 || !m_SaveImage)
682                     return "Capture␣Image";
683                 else
684                     return "Capture␣Image␣(" + m_frames.ToString() + "␣frames
                        )";
685             }
686             else
687             {
688                 return m_Description;
689             }
690         }
691
692         public override void ConvertToDataRow(ref DataRow dataRow)
693         {
694             dataRow["StepType"] = this.GetType().Name;
695             dataRow["Description"] = m_Description;
696             dataRow["CheckSaveImage"] = m_SaveImage;
697             dataRow["SaveImageFolder"] = m_SaveFolder;
698             dataRow["ImageFileName"] = m_FileName;
699             dataRow["FramesNumber"] = m_frames;
700
701             return;
702         }
703
704         public override void ConvertFromDataRow(DataRow dataRow)
705         {
706             // Get Description
707             if (dataRow["Description"].GetType().Name != "DBNull")
708             {
709                 m_Description = (string)dataRow["Description"];
710             }
711             else
712             {
713                 m_Description = null;
714             }
```

```
715
716             // Get Image Saving Preference
717             if (dataRow["CheckSaveImage"].GetType().Name != "DBNull")
718             {
719                 m_SaveImage = ((bool)dataRow["CheckSaveImage"]);
720             }
721             else
722             {
723                 m_SaveImage = false;
724             }
725
726             // Get Save Image Folder
727             if (dataRow["SaveImageFolder"].GetType().Name != "DBNull")
728             {
729                 m_SaveFolder = ((string)dataRow["SaveImageFolder"]);
730             }
731             else
732             {
733                 m_SaveFolder = (String.Empty);
734             }
735
736             // Get File Name
737             if (dataRow["ImageFileName"].GetType().Name != "DBNull")
738             {
739                 m_FileName = ((string)dataRow["ImageFileName"]);
740             }
741             else
742             {
743                 m_FileName = (String.Empty);
744             }
745
746             // Get Frames Number
747             if (dataRow["FramesNumber"].GetType().Name != "DBNull")
748             {
749                 m_frames = ((int)dataRow["FramesNumber"]);
750             }
751             else
752             {
753                 m_frames = 1;
754             }
755         }
```

```
756 }
757
758 public class AnalyzeImage : TestStep
759 {
760     ImageAnalysis m_ImageAnalysis;
761     double m_ValueTarget;
762     bool m_Operator;
763
764     public AnalyzeImage()
765     {
766         m_ImageAnalysis = ImageAnalysis.Brightness;
767         m_ValueTarget = 0;
768         m_Operator = true;
769     }
770
771     public void SetImageAnalysis (ImageAnalysis imageResolution)
772     {
773         m_ImageAnalysis = imageResolution;
774     }
775
776     public ImageAnalysis GetImageAnalysis()
777     {
778         return m_ImageAnalysis;
779     }
780
781     public void SetValueTarget (double numericValue)
782     {
783         m_ValueTarget = numericValue;
784     }
785
786     public double GetValueTarget()
787     {
788         return m_ValueTarget;
789     }
790
791     public void SetOperator(string majorOrMinor)
792     {
793         if (majorOrMinor == ">=")
794         {
795             m_Operator = true;
796         }
```

```
797             else
798             {
799                 m_Operator = false;
800             }
801         }
802
803         public bool GetOperator()
804         {
805             return m_Operator;
806         }
807
808         public override string ToString()
809         {
810             if ((m_Description == null) || (m_Description == String.Empty))
811             {
812                 return "Analyze␣Image";
813             }
814             else
815             {
816                 return m_Description;
817             }
818         }
819
820         public override void ConvertToDataRow(ref DataRow dataRow)
821         {
822             dataRow["StepType"] = this.GetType().Name;
823             dataRow["Description"] = m_Description;
824             dataRow["ImageAnalysis"] = m_ImageAnalysis;
825             dataRow["ValueTarget"] = m_ValueTarget;
826             dataRow["Operator"] = m_Operator;
827
828             return;
829         }
830
831         public override void ConvertFromDataRow(DataRow dataRow)
832         {
833             // Get Description
834             if (dataRow["Description"].GetType().Name != "DBNull")
835             {
836                 m_Description = (string)dataRow["Description"];
837             }
```

```
838             else
839             {
840                 m_Description = null;
841             }
842
843             // Get ImageAnalysis
844             if (dataRow["ImageAnalysis"].GetType().Name != "DBNull")
845             {
846                 string s = (string)dataRow["ImageAnalysis"];
847                 if (s == "Brightness")
848                 {
849                     m_ImageAnalysis = ImageAnalysis.Brightness;
850                 }
851                 else if (s == "BrightnessDistribution")
852                 {
853                     m_ImageAnalysis = ImageAnalysis.BrightnessDistribution;
854                 }
855                 else if (s == "BrightnessLoss")
856                 {
857                     m_ImageAnalysis = ImageAnalysis.BrightnessLoss;
858                 }
859                 else if (s == "BrightSaturation")
860                 {
861                     m_ImageAnalysis = ImageAnalysis.BrightSaturation;
862                 }
863                 else if (s == "Contrast")
864                 {
865                     m_ImageAnalysis = ImageAnalysis.Contrast;
866                 }
867                 else if (s == "ContrastBalance")
868                 {
869                     m_ImageAnalysis = ImageAnalysis.ContrastBalance;
870                 }
871                 else if (s == "DarkSaturation")
872                 {
873                     m_ImageAnalysis = ImageAnalysis.DarkSaturation;
874                 }
875                 else if (s == "InterFrameBrightnessStability")
876                 {
877                     m_ImageAnalysis = ImageAnalysis.
                            InterFrameBrightnessStability;
```

```
878                 }
879                 else if (s == "InterFrameNoise")
880                 {
881                     m_ImageAnalysis = ImageAnalysis.InterFrameNoise;
882                 }
883                 else if (s == "PixelNoise")
884                 {
885                     m_ImageAnalysis = ImageAnalysis.PixelNoise;
886                 }
887                 else if (s == "SNR")
888                 {
889                     m_ImageAnalysis = ImageAnalysis.Snr;
890                 }
891                 else if (s == "Blur")
892                 {
893                     m_ImageAnalysis = ImageAnalysis.Blur;
894                 }
895                 else if (s == "AimVisibility")
896                 {
897                     m_ImageAnalysis = ImageAnalysis.AimVisibility;
898                 }
899             }
900             else
901             {
902                 m_ImageAnalysis = ImageAnalysis.Brightness;
903             }
904 
905             // Get ValueTarget
906             if (dataRow["ValueTarget"].GetType().Name != "DBNull")
907             {
908                 m_ValueTarget = ((double)dataRow["ValueTarget"]);
909             }
910             else
911             {
912                 m_ValueTarget = 0.0;
913             }
914 
915             // Get Operator
916             if (dataRow["Operator"].GetType().Name != "DBNull")
917             {
918                 m_Operator = ((bool)dataRow["Operator"]);
```

```
919             }
920             else
921             {
922                 m_Operator = true;
923             }
924         }
925     }
926
927     public class UserMessage : TestStep
928     {
929         string m_userText;
930
931         string m_imageFileName;
932
933         public void SetUserMessage(string userMessage)
934         {
935             m_userText = userMessage;
936         }
937
938         public string GetUserMessage()
939         {
940             return m_userText;
941         }
942
943         public string ImageFileName
944         {
945             get { return m_imageFileName; }
946             set { m_imageFileName = value; }
947         }
948
949         public UserMessage()
950         {
951             m_userText = null;
952             m_imageFileName = null;
953         }
954
955         public override string ToString()
956         {
957             if ((m_Description == null) || (m_Description == String.Empty))
958             {
959                 return "User␣Message";
```

```
960             }
961             else
962             {
963                 return m_Description;
964             }
965         }
966
967         public override void ConvertToDataRow(ref DataRow dataRow)
968         {
969             dataRow["StepType"] = this.GetType().Name;
970             dataRow["Description"] = m_Description;
971             dataRow["UserText"] = m_userText;
972             dataRow["ImageFileName"] = m_imageFileName;
973
974             return;
975         }
976
977         public override void ConvertFromDataRow(DataRow dataRow)
978         {
979             // Get Description
980             if (dataRow["Description"].GetType().Name != "DBNull")
981             {
982                 m_Description = (string)dataRow["Description"];
983             }
984             else
985             {
986                 m_Description = null;
987             }
988
989             // Get User Text
990             if (dataRow["UserText"].GetType().Name != "DBNull")
991             {
992                 m_userText = ((string)dataRow["UserText"]);
993             }
994             else
995             {
996                 m_userText = (String.Empty);
997             }
998
999             // Get Image File Name
1000            if (dataRow["ImageFileName"].GetType().Name != "DBNull")
```

```
1001            {
1002                m_imageFileName = ((string)dataRow["ImageFileName"]);
1003            }
1004            else
1005            {
1006                m_imageFileName = (String.Empty);
1007            }
1008        }
1009    }
1010
1011    public class UserFeedback : TestStep
1012    {
1013        bool m_ExpYesAnswer;
1014        string m_Message;
1015
1016        public string Message
1017        {
1018            get { return m_Message; }
1019            set { m_Message = value; }
1020        }
1021
1022        public UserFeedback()
1023        {
1024            m_ExpYesAnswer = true;
1025        }
1026
1027        public override string ToString()
1028        {
1029            if ((m_Description == null) || (m_Description == String.Empty))
1030            {
1031                return "User␣Feedback";
1032            }
1033            else
1034            {
1035                return m_Description;
1036            }
1037        }
1038
1039        public void SetFeedback(bool expAnswer)
1040        {
1041            m_ExpYesAnswer = expAnswer;
```

```
1042        }
1043
1044        public bool GetFeedback()
1045        {
1046            return m_ExpYesAnswer;
1047        }
1048
1049        public override void ConvertToDataRow(ref DataRow dataRow)
1050        {
1051            dataRow["StepType"] = this.GetType().Name;
1052            dataRow["Description"] = m_Description;
1053            dataRow["UserText"] = m_Message;
1054            dataRow["ExpectedAnswer"] = m_ExpYesAnswer;
1055
1056            return;
1057        }
1058
1059        public override void ConvertFromDataRow(DataRow dataRow)
1060        {
1061            // Get Description
1062            if (dataRow["Description"].GetType().Name != "DBNull")
1063            {
1064                m_Description = (string)dataRow["Description"];
1065            }
1066            else
1067            {
1068                m_Description = null;
1069            }
1070
1071            // Get UserText
1072            if (dataRow["UserText"].GetType().Name != "DBNull")
1073            {
1074                m_Message = ((string)dataRow["UserText"]);
1075            }
1076            else
1077            {
1078                m_Message = (String.Empty);
1079            }
1080
1081            // Get ExpectedAnswer
1082            if (dataRow["ExpectedAnswer"].GetType().Name != "DBNull")
```

```
1083            {
1084                m_ExpYesAnswer = ((bool)dataRow["ExpectedAnswer"]);
1085            }
1086            else
1087            {
1088                m_ExpYesAnswer = true;
1089            }
1090        }
1091    }
1092
1093    public class TestCase : ArrayList
1094    {
1095        string m_Name;
1096        public string Name
1097        {
1098            get { return m_Name; }
1099            set { m_Name = value; }
1100        }
1101
1102        string m_FileName;
1103        public string FileName
1104        {
1105            get { return m_FileName; }
1106            set { m_FileName = value; }
1107        }
1108
1109        bool m_NeedToSave;
1110        public bool NeedToSave
1111        {
1112            get { return m_NeedToSave; }
1113            set { m_NeedToSave = value; }
1114        }
1115
1116        public event ResultChangedEventHandler ResultChanged;
1117
1118        // Invoke the ResultChanged event; called whenever the test case result
                changes
1119        protected virtual void OnResultChanged(EventArgs e)
1120        {
1121            if (ResultChanged != null)
1122                ResultChanged(this, e);
```

```
1123        }
1124
1125        int m_Result; // The outcome of the test case execution (not used in
                Designer View).
1126
1127        bool result_changed;
1128
1129        public int Result
1130        {
1131            get
1132            {
1133                return m_Result;
1134            }
1135            set
1136            {
1137                if (value == −1 || value == 0 || value == 1)
1138                {
1139                    result_changed = false;
1140                    if (m_Result != value)
1141                        result_changed = true;
1142
1143                    m_Result = value;
1144
1145                    if (result_changed)
1146                        OnResultChanged(EventArgs.Empty);
1147                }
1148            }
1149        } // The outcome of each test case execution (not used in Designer View).
1150
1151        public TestCase()
1152        {
1153            Reset();
1154        }
1155
1156        public void Reset()
1157        {
1158            m_Name = null;
1159            m_FileName = null;
1160            m_NeedToSave = false;
1161            m_Result = −1; // Not executed yet.
1162            Clear();
```

```
1163        }
1164
1165        public override string ToString()
1166        {
1167            if (m_Name != null && m_Name != String.Empty)
1168            {
1169                return (m_Name);
1170            }
1171
1172            if (m_FileName != null && m_FileName != String.Empty)
1173                return(Path.GetFileNameWithoutExtension(m_FileName));
1174
1175            return "{Empty␣Test␣Case␣Name}";
1176        }
1177
1178        DataSetTestCase m_DataSetTestCase = null;
1179
1180        public void WriteXml(string fileName)
1181        {
1182            ConvertToDataSet();
1183            m_DataSetTestCase.WriteXml(fileName);
1184            m_FileName = fileName;
1185        }
1186
1187        public void ReadXml(string fileName)
1188        {
1189            if (fileName != String.Empty) // Avoids ArgumentException.
1190            {
1191                Clear();
1192                m_DataSetTestCase = new DataSetTestCase();
1193                m_DataSetTestCase.ReadXml(fileName);
1194                ConvertFromDataSet();
1195                m_FileName = fileName;
1196                m_NeedToSave = false;
1197            }
1198        }
1199
1200        private void ConvertToDataSet()
1201        {
1202            m_DataSetTestCase = new DataSetTestCase();
1203
```

```
1204            // Save the name
1205            DataRow dataRow = m_DataSetTestCase.DataTableTestCase.NewRow
                    ();
1206            dataRow["Name"] = m_Name;
1207            m_DataSetTestCase.DataTableTestCase.Rows.Add(dataRow);
1208
1209            // Save all steps
1210            foreach (TestStep testStep in this)
1211            {
1212                dataRow = m_DataSetTestCase.DataTableStep.NewRow();
1213                testStep.ConvertToDataRow(ref dataRow);
1214                m_DataSetTestCase.DataTableStep.Rows.Add(dataRow);
1215            }
1216            return;
1217        }
1218
1219        private void ConvertFromDataSet()
1220        {
1221            // Load the name
1222            m_Name = null;
1223            if (m_DataSetTestCase.DataTableTestCase.Rows.Count > 0)
1224            {
1225                DataRow dataRow = m_DataSetTestCase.DataTableTestCase.Rows
                        [0];
1226                if (dataRow["Name"].GetType().Name != "DBNull")
1227                {
1228                    m_Name = (string)dataRow["Name"];
1229                }
1230            }
1231
1232            // Load all steps
1233            for (int i = 0; i < m_DataSetTestCase.DataTableStep.Rows.Count; i++ )
1234            {
1235                DataRow dataRow = m_DataSetTestCase.DataTableStep.Rows[i];
1236
1237                // Create a step of the appropriate type
1238                TestStep newStep = null;
1239                string stepType = (string)dataRow["StepType"];
1240                if (stepType == "Sleep")
1241                {
1242                    newStep = new Sleep();
```

```
1243                }
1244                else if (stepType == "SendCommand")
1245                {
1246                    newStep = new SendCommand();
1247                }
1248                else if (stepType == "LoadImage")
1249                {
1250                    newStep = new LoadImage();
1251                }
1252                else if (stepType == "CaptureImage")
1253                {
1254                    newStep = new CaptureImage();
1255                }
1256                else if (stepType == "AnalyzeImage")
1257                {
1258                    newStep = new AnalyzeImage();
1259                }
1260                else if (stepType == "UserMessage")
1261                {
1262                    newStep = new UserMessage();
1263                }
1264                else if (stepType == "UserFeedback")
1265                {
1266                    newStep = new UserFeedback();
1267                }
1268
1269                // Fill the step and add it to the test case
1270                if (newStep != null)
1271                {
1272                    newStep.ConvertFromDataRow(dataRow);
1273                    Add(newStep);
1274                }
1275            }
1276            return;
1277        }
1278
1279        public int CalculateResult()
1280        {
1281            int result = 1;
1282            foreach (TestStep testStep in this)
1283            {
```

```
1284                if (testStep.Result == 0)
1285                {
1286                    result = 0;
1287                    break;
1288                }
1289                else if (testStep.Result == -1)
1290                {
1291                    result = -1;
1292                    break;
1293                }
1294            }
1295            Result = result;
1296            return result;
1297        }
1298    }
1299
1300    // It can be further generalized as TestDirective, if needed to include other types
        of
1301    // execution directives
1302    public class TestIterator
1303    {
1304        int m_LoopIterations;
1305
1306        public int LoopIterations
1307        {
1308            get
1309            {
1310                return m_LoopIterations;
1311            }
1312            set
1313            {
1314                if (value > 0)
1315                    m_LoopIterations = value;
1316            }
1317        }
1318
1319        int m_LoopSize;
1320
1321        public int LoopSize {
1322            get
1323            {
```

```
1324                return m_LoopSize;
1325            }
1326            set
1327            {
1328                if (value > 0)
1329                    m_LoopSize = value;
1330            }
1331        }
1332
1333        public TestIterator()
1334        {
1335            Reset();
1336        }
1337
1338        public void Reset()
1339        {
1340            m_LoopIterations = 1;
1341            m_LoopSize = 1;
1342        }
1343
1344        public override string ToString()
1345        {
1346            return "---␣Iterate␣next␣" + LoopSize + "␣elements␣" +
                        LoopIterations + "␣times␣---";
1347        }
1348    }
1349
1350    public class TestPlan : ArrayList
1351    {
1352        string[] m_FileNames = null;
1353
1354        // TODO separare il nome del test case dal nome del file
1355        string m_Name;
1356        public string Name
1357        {
1358            get { return m_Name; }
1359            set { m_Name = value; }
1360        }
1361
1362        DataSetTestPlan m_DataSetTestPlan = null;
1363
```

```
1364        string m_FileName;
1365        public string FileName
1366        {
1367            get { return m_FileName; }
1368            set { m_FileName = value; }
1369        }
1370
1371        bool m_NeedToSave;
1372        public bool NeedToSave
1373        {
1374            get { return m_NeedToSave; }
1375            set { m_NeedToSave = value; }
1376        }
1377
1378        int m_Result;
1379
1380        public int Result
1381        {
1382            get
1383            {
1384                return m_Result;
1385            }
1386            set
1387            {
1388                if (value == -1 || value == 0 || value == 1)
1389                {
1390                    m_Result = value;
1391                }
1392            }
1393        } // The outcome of each test plan execution (not used in Designer View).
1394
1395        public TestPlan()
1396        {
1397            Reset();
1398        }
1399
1400        public void Reset()
1401        {
1402            m_Name = null;
1403            m_FileName = null;
1404            m_NeedToSave = false;
```

```
1405            m_Result = -1; // Not executed yet.
1406            Clear();
1407        }
1408
1409        public override string ToString()
1410        {
1411            if (m_Name != null && m_Name != String.Empty)
1412            {
1413                return (m_Name);
1414            }
1415
1416            if (m_FileName != null && m_FileName != String.Empty)
1417                return (Path.GetFileNameWithoutExtension(m_FileName));
1418
1419            return "{Empty␣Test␣Plan␣Name}";
1420        }
1421
1422        public void WriteXml(string fileName)
1423        {
1424            m_FileName = fileName;
1425            ConvertToDataSet();
1426            if (fileName != String.Empty)
1427                m_DataSetTestPlan.WriteXml(fileName);
1428        }
1429
1430        virtual public void ConvertToDataSet()
1431        {
1432            m_DataSetTestPlan = new DataSetTestPlan();
1433
1434            // Save the name and the filename
1435            DataRow dataRow = m_DataSetTestPlan.DataTableTestPlan.NewRow()
                    ;
1436            dataRow["Name"] = m_Name;
1437
1438            dataRow["FileName"] = m_FileName;
1439            m_DataSetTestPlan.DataTableTestPlan.Rows.Add(dataRow);
1440
1441            for (int i = 0; i < this.Count; i++)
1442            {
1443                if (this[i].GetType().Name == "TestCase")
1444                {
```

```
1445                    // Save the test case filename
1446                    TestCase testCase = ((TestCase)this[i]);
1447                    string filename = testCase.FileName;
1448                    dataRow = m_DataSetTestPlan.DataTablePlanList.NewRow();
1449                    dataRow["TestCaseFileName"] = filename;
1450                    m_DataSetTestPlan.DataTablePlanList.Rows.Add(dataRow);
1451                }
1452                else if (this[i].GetType().Name == "TestIterator")
1453                {
1454                    TestIterator testIterator = ((TestIterator)this[i]);
1455                    dataRow = m_DataSetTestPlan.DataTablePlanList.NewRow();
1456                    dataRow["LoopIterations"] = testIterator.LoopIterations;
1457                    dataRow["LoopSize"] = testIterator.LoopSize;
1458                    m_DataSetTestPlan.DataTablePlanList.Rows.Add(dataRow);
1459                }
1460            }
1461
1462            return;
1463        }
1464
1465        virtual public void ConvertFromDataSet()
1466        {
1467            int number_of_test_cases, j;
1468
1469            // Load the name
1470            m_Name = null;
1471            if (m_DataSetTestPlan.DataTableTestPlan.Rows.Count > 0)
1472            {
1473                DataRow dataRow = m_DataSetTestPlan.DataTableTestPlan.Rows
                        [0];
1474                if (dataRow["Name"].GetType().Name != "DBNull")
1475                {
1476                    m_Name = (string)dataRow["Name"];
1477                }
1478            }
1479
1480            // Determine the total number of test cases
1481            number_of_test_cases = 0;
1482            for (int i = 0; i < m_DataSetTestPlan.DataTablePlanList.Rows.Count; i
                    ++)
1483            {
```

```
1484            DataRow dataRow = m_DataSetTestPlan.DataTablePlanList.Rows[
                    i];
1485            if (dataRow["TestCaseFileName"].GetType().Name != "DBNull")
1486                number_of_test_cases++;
1487        }
1488
1489        m_FileNames = new string[number_of_test_cases];
1490
1491        // Load all elements (test cases or iteration directives)
1492        j = 0;
1493        for (int i = 0; i < m_DataSetTestPlan.DataTablePlanList.Rows.Count; i
                ++)
1494        {
1495            DataRow dataRow = m_DataSetTestPlan.DataTablePlanList.Rows[
                    i];
1496
1497            if (dataRow["LoopIterations"].GetType().Name != "DBNull")
1498            {
1499                TestIterator testIterator = new TestIterator();
1500                testIterator.LoopIterations = (int)dataRow["LoopIterations"];
1501                testIterator.LoopSize = 1;
1502                if (dataRow["LoopSize"].GetType().Name != "DBNull")
1503                {
1504                    testIterator.LoopSize = (int)dataRow["LoopSize"];
1505                }
1506                Add(testIterator);
1507            }
1508            else if (dataRow["TestCaseFileName"].GetType().Name != "
                    DBNull")
1509            {
1510                m_FileNames[j] = (string)dataRow["TestCaseFileName"];
1511
1512                if (File.Exists(m_FileNames[j]))
1513                {
1514                    // Load the test case
1515                    TestCase testCase = new TestCase();
1516                    testCase.ReadXml(m_FileNames[j]);
1517                    Add(testCase);
1518                    j++;
1519                }
1520            }
```

```
1521            }
1522
1523            return;
1524        }
1525
1526        public void ReadXml(string fileName)
1527        {
1528            if (fileName != String.Empty) // Avoids ArgumentException.
1529            {
1530                Clear();
1531                m_DataSetTestPlan = new DataSetTestPlan();
1532                m_DataSetTestPlan.ReadXml(fileName);
1533                ConvertFromDataSet();
1534                m_FileName = fileName;
1535                m_NeedToSave = false;
1536            }
1537        }
1538
1539        public int CalculateResult()
1540        {
1541            int result = 1;
1542
1543            for (int i = 0; i < this.Count; i++)
1544            {
1545                if (this[i].GetType().Name == "TestCase")
1546                {
1547                    TestCase testCase = ((TestCase)this[i]);
1548                    if (testCase.Result == 0)
1549                    {
1550                        result = 0;
1551                        break;
1552                    }
1553                    else if (testCase.Result == -1)
1554                    {
1555                        result = -1;
1556                        break;
1557                    }
1558                }
1559            }
1560
1561            Result = result;
```

```
1562                return result;
1563            }
1564    }
```

Appendice E

Codice sorgente per il motore di esecuzione

```
1  ï»¿using System;
2  using System.Globalization;
3  using System.IO;
4  using System.Reflection;
5  using System.Runtime.InteropServices; // for the DllImport attribute
6  using System.Text;
7  using System.Threading;
8  using System.Windows.Forms;
9
10 namespace ScanEngineTestProgram
11 {
12    public class ExecutionThread
13    {
14       // TODO: Should be able to select a specific device, when multiple devices
         are available.
15       const string portName = "AARDVARK"; // Select Aardvark ("AARDVARK
         ") instead of Pleora ("PLEORA").
16
17       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
18       static extern int comPortInit(string portName, uint Bitrate, uint Timeout,
         byte addr);
19
20       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
21       static extern int comPortClose();
22
```

```
23      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
24      static extern int CMD_CameraStart(int argB);
25
26      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
27      static extern int CMD_SetCameraMode(int argB);
28
29      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
30      static extern int CMD_SetSensorOperatingMode(int argB);
31
32      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
33      static extern int CMD_get_camera_param(ushort param, byte[] buffer);
34
35      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
36      static extern int CMD_set_camera_param(ushort param, byte[] buffer);
37
38      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
39      static extern int CMD_getClockGeneratorRegister(byte regName, byte[]
        valueLSB, byte[] valueMSB);
40
41      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
42      static extern int CMD_setClockGeneratorRegister(byte regName, byte
        valueLSB, byte valueMSB);
43
44      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
45      static extern int CMD_getImageSensorRegister(byte regName, out byte
        valueLSB, out byte valueMSB);
46
47      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
48      static extern int CMD_setImageSensorRegister(byte regName, byte
        valueLSB, byte valueMSB);
49
50      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
51      static extern int CMD_IlluminationEnable(int argB);
52
53      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
54      static extern int CMD_AimControl(int arg);
55
56      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
57      static extern int CMD_enableAimingControlLine(int onOff);
58
59      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
```

```
60      static extern int CMD_AimBlinkRate(byte onTime, byte offTime);
61
62      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
63      static extern int CMD_AimDuringExposure(int arg);
64
65      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
66      static extern int CMD_AimPower(byte duration);
67
68      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
69      static extern int CMD_SetAutoPowerLevel(int argB);
70
71      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
72      static extern int CMD_SetAutoPowerTimeout(byte tOut);
73
74      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
75      static extern int CMD_get_digipot_value(byte[] digiLevel);
76
77      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
78      static extern int CMD_set_digipot_value(byte level);
79
80      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
81      static extern int CMD_store_digipot_value(byte[] digiLevel);
82
83      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
84      static extern int CMD_runCommandList(byte listNum);
85
86      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
87      static extern int CMD_SetCommandList(byte listNum, byte[] buf, ushort
        listCmdLen);
88
89      [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
90      static extern int CMD_Set_FPS(int argB);
91
92      // Parameters "Size" and "Pack" are probably not needed
93      // The use of the attribute MarshalAsAttribute is probably not needed
94      [StructLayout(LayoutKind.Sequential, Pack=1, Size=4)]
95      public struct reportArg_t
96      {
97          [MarshalAsAttribute(UnmanagedType.U1)]
98          byte aimPower;
99
```

```
100         [MarshalAsAttribute(UnmanagedType.U1)]
101         byte thermalWarning;
102
103         [MarshalAsAttribute(UnmanagedType.U1)]
104         byte spare1;
105
106         [MarshalAsAttribute(UnmanagedType.U1)]
107         byte spare2;
108       };
109
110       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
111       static extern int CMD_get_StatusReport(ref reportArg_t report);
112
113       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
114       static extern int CMD_SensorSetROI(int edge, ushort pixel);
115
116       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
117       static extern int CMD_SensorSetBinning(int argB);
118
119       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
120       static extern int CMD_SetPowerLevel(int argB);
121
122       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
123       static extern int CMD_CameraReset(int argB);
124
125       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
126       static extern int CMD_GetTemperature(byte[] buffer);
127
128       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
129       static extern int CMD_ControlInternalWatchdog(int argB);
130
131       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
132       static extern int CMD_SetOscillatorPowerLevel(int argB);
133
134       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
135       static extern int CMD_DitheringEnable(int argB);
136
137       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
138       static extern int CMD_setGpioPortLev(byte port, byte pin, byte level);
139
140       [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
```

```
141         static extern int CMD_setGpioPortDir(byte port, byte pin, byte dir);
142
143         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
144         static extern int CMD_checkGpioPortPin(byte port, byte pin, byte[] level)
            ;
145
146         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
147         static extern int CMD_enterBootloader(byte[] argBoot);
148
149         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
150         static extern int CMD_startBootloader(int argB);
151
152         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
153         static extern int CMD_templateCmd(byte cmd, byte[] argCmdB, byte
            argCmdBLen, byte[] argResB, byte argResBLen);
154
155         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
156         static extern int CMD_toggleTestMode(byte timeout);
157
158         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
159         static extern int CMD_enableLvdsMode(int mode);
160
161         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
162         static extern int CMD_Crypto_getPubKey(byte[] pubKeyLen, byte[]
            pubKey, byte[] bigNumLen, byte[] bigNum);
163
164         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
165         static extern int CMD_Crypto_Autenticate(byte[] data);
166
167         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
168         static extern int CMD_Restore_FactoryDefaults(byte argB);
169
170         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
171         static extern int CMD_Illumination_Delay(uint argB);
172
173         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
174         static extern int CMD_Illumination_Duration(uint argB);
175
176         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
177         static extern int CMD_Picklist_Mode(byte argB);
178
```

```
179         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
180         static extern int CMD_Led_Drive(byte argB);
181
182         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
183         static extern int CMD_SetAimVcc(byte argB);
184
185         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
186         static extern byte Oscillator_GetRegister(byte reg, byte[] parLSB, byte[]
            parMSB);
187
188         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
189         static extern byte Oscillator_SetRegister(byte reg, byte parLSB, byte
            parMSB);
190
191         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
192         static extern int CMD_HardwarReset(byte gpioIn, uint delay_ms);
193
194         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
195         static extern int CMD_EnablePowerSupply(byte gpioIn, int level);
196
197         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
198         static extern int CMD_CPLDPROG_SendAndCheckAck(byte[] data);
199
200         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
201         static extern int CMD_getFakeCameraParam(byte param, byte[] buffer);
202
203         [DllImport("device.dll", CallingConvention = CallingConvention.Cdecl)]
204         static extern int CMD_Generic(byte cmd, byte[] argCmdB, byte
            argCmdBLen);
205
206         const int MAX_ANSWER_LENGTH = 40; // The maximum length of an
            answer from the scan engine (buffer length)
207
208         public ushort defaultAddr; // the default I2C address for the slave device
209         public ushort slaveAddr; // the current I2C address for the slave device
210         public ushort Bitrate; // kHz, the bitrate to configure for the I2C bus
211         public ushort Timeout; // ms, the I2C bus lock timeout
212
213         // Enumeration imported from C++ header file halogen1Lib.h
214         enum ExitStatus {    CMD_OK=0, CMD_CKS_ERR=-1, CMD_NAK
            =-2, CMD_WRONG_ANSW=-3, CMD_WRONG_ARG=-4,
```

```
        CMD_WRONG_CMD=-5,
215                             CMD_UNABLE_TO_CLOSE=-8,
        CMD_INVALID_HANDLE=-9, CMD_CONFIG_ERROR=-10,
        CMD_CMD_NOT_SUPORTED=-11,
216                             CMD_CMD_NOT_IMPLEMENTED=-12,
        CMD_TARGET_NOT_RES=-13, CMD_WD_RESET=-14,
217                             CMD_I2C_NOT_AVAILABLE=-100,
        CMD_I2C_NOT_ENABLED=-101,
218                             CMD_I2C_READ_ERROR=-102,
        CMD_I2C_WRITE_ERROR=-103
219             };
220
221     MainForm mainForm;
222
223     MainForm.report_level ReportLevel; // The level of the report to be
        generated during and after execution
224
225     string TesterName = null;
226
227     MainForm.on_failure OnFailure;
228
229     DateTime startTime, endTime;
230     TimeSpan execution_duration;
231
232     public Thread workerThread;
233     public bool paused; // Indicates (does not control) paused state
234     public bool debug; // Indicates (does not control) debug mode
235
236     public ExecutionThread(MainForm mainForm, TestPlan m_TestPlan,
        MainForm.report_level ReportLevel, string TesterName, MainForm.
        on_failure OnFailure)
237     {
238         // Load the settings from the general configuration
239         defaultAddr = UInt16.Parse(Properties.Settings.Default.I2CAddress,
        NumberStyles.AllowHexSpecifier);
240         slaveAddr = defaultAddr;
241         Bitrate = Properties.Settings.Default.I2CBitrate;
242         Timeout = Properties.Settings.Default.I2CTimeout;
243
244         startTime = DateTime.Now;
245
```

```
246         this.mainForm = mainForm;
247
248         mainForm.ConsoleAppendText("Execution␣thread␣instantiated.\r\n");
249
250         this.ReportLevel = ReportLevel;
251         this.TesterName = TesterName;
252         this.OnFailure = OnFailure;
253
254         workerThread = new Thread(this.Run);
255
256         try
257         {
258            workerThread.SetApartmentState(ApartmentState.STA);
259         }
260         catch (InvalidOperationException)
261         {
262            mainForm.ConsoleAppendText("Cannot␣set␣the␣thread's␣apartment
       ␣state...\r\n");
263         }
264
265         workerThread.Start(m_TestPlan);
266       }
267
268       void Run(object TPlan)
269       {
270          // I2C bitrate for version 3.x Aardvark hardware should be within 1kHz
       and 800kHz.
271          // The default power-on bitrate is 100kHz. Only certain discrete bitrates
       are
272          // supported by the Aardvark. As such, the actual bitrate set will be less
       than
273          // or equal to the requested bitrate.
274          //
275          // The Atmel microcontroller only officially supports 100kHz and 400kHz.
276          const int MIN_I2C_BITRATE = MainForm.MIN_I2C_BITRATE;
277          const int MAX_I2C_BITRATE = MainForm.MAX_I2C_BITRATE;
278
279          const int DEFAULT_I2C_BITRATE = MAX_I2C_BITRATE;
280
281          bool port_opened = false;
282
```

```
283         int status; // The status after an I2C command or after a configuration of
        the I2C host adapter.
284
285         int total_test_cases, relative_test_cases, test_case_failures,
        test_case_success, test_cases_executed;
286         float percentage_test_case_failure, percentage_test_case_success,
        percentage_test_case_executed;
287         int total_test_steps, test_step_failures, test_step_success,
        test_steps_executed;
288         float percentage_test_step_failure, percentage_test_step_success,
        percentage_test_step_executed;
289
290         int number_of_iterations, iteration_size, relative_position; // Used for
        looping functionality
291
292         ushort new_I2CAddress; // The slave address configured for a specific
        command
293
294         TestPlan testPlan = (TestPlan)TPlan;
295         TestCase testCase;
296         TestStep testStep;
297
298         byte[] buffer = new byte[MAX_ANSWER_LENGTH];
299
300         string report_filename = null;
301         StreamWriter reportWriter = null;
302
303         mainForm.ConsoleAppendText("Execution␣thread␣started.\r\n");
304
305         // Reset the test results from previous executions
306         for (int i = 0; i < testPlan.Count; i++)
307         {
308             if (testPlan[i].GetType().Name == "TestCase")
309             {
310                 testCase = (TestCase)testPlan[i];
311                 for (int j = 0; j < testCase.Count; j++)
312                 {
313                     // Loop until paused
314                     while (_shouldPause)
315                     {
316                         paused = true;
```

```
317                     if (_shouldStop)
318                         break;
319                 }
320
321                 paused = false;
322
323                 if (_shouldStop)
324                     break;
325
326                 testStep = (TestStep)testCase[j];
327                 testStep.Result = -1;
328             }
329             testCase.Result = -1;
330         }
331     }
332     testPlan.Result = -1;
333
334     // Initialize execution statistics variables
335     total_test_cases = 0;
336     relative_test_cases = 0;
337     total_test_steps = 0;
338     test_case_failures = 0;
339     test_case_success = 0;
340     test_cases_executed = 0;
341     test_step_failures = 0;
342     test_step_success = 0;
343     test_steps_executed = 0;
344
345     // Calculate the total number of test cases and test steps
346     for (int i = 0; i < testPlan.Count; i++)
347     {
348         number_of_iterations = 1;
349         iteration_size = 1;
350
351         if (testPlan[i].GetType().Name == "TestIterator")
352         {
353             if (((TestIterator)testPlan[i]).LoopIterations > 0)
354                 number_of_iterations = ((TestIterator)testPlan[i]).LoopIterations;
355             if (((TestIterator)testPlan[i]).LoopSize > 0)
356                 iteration_size = ((TestIterator)testPlan[i]).LoopSize;
357             if (i < testPlan.Count - 1)
```

```
358                 i++; // Skip to next test plan element
359              else
360                 break;
361           }
362
363           while (number_of_iterations > 0)
364           {
365              number_of_iterations−−;
366
367              for (int k = 0; k < iteration_size; k++)
368              {
369                 if (i + k >= testPlan.Count)
370                    break;
371
372                 if (testPlan[i + k].GetType().Name == "TestCase")
373                 {
374                    total_test_cases++;
375                    total_test_steps += ((TestCase)testPlan[i + k]).Count;
376                 }
377              }
378           }
379
380           if (iteration_size > 0)
381              i += iteration_size − 1;
382        }
383
384        // Calculate the relative number of test cases (not considering iterations)
385        for (int i = 0; i < testPlan.Count; i++)
386        {
387           if (testPlan[i].GetType().Name == "TestCase")
388              relative_test_cases++;
389        }
390
391        paused = false;
392        while (!_shouldStop)
393        {
394           int count;
395
396           if (Bitrate < MIN_I2C_BITRATE || Bitrate > MAX_I2C_BITRATE)
397           {
```

```
398             mainForm.ConsoleAppendText("Error␣while␣trying␣to␣configure
    ␣the␣bitrate:␣invalid␣rate␣" + Bitrate + ".␣Using␣default␣bitrate␣" +
    DEFAULT_I2C_BITRATE + ".\r\n");
399             Bitrate = DEFAULT_I2C_BITRATE;
400         }
401
402         status = -1;
403         try
404         {
405             status = comPortInit(portName, (uint)Bitrate, (uint)Timeout, (byte)
    slaveAddr); // Open and configure the I2C host adapter
406         }
407         catch (DllNotFoundException)
408         {
409             mainForm.ConsoleAppendText("Error␣while␣trying␣to␣load␣
    dynamic-link␣libraries:␣device.dll␣not␣found␣!\r\n");
410             return;
411         }
412         catch (EntryPointNotFoundException)
413         {
414             EntryPointNotFound_Exit();
415         }
416         if (status < 0)
417         {
418             mainForm.ConsoleAppendText("Unable␣to␣open␣the␣I2C␣host␣
    adapter␣(status:␣" + status + ").\r\n");
419             port_opened = false;
420         }
421         else
422         {
423             mainForm.ConsoleAppendText("I2C␣host␣adapter␣opened␣
    successfully␣(bitrate␣=␣" + Bitrate + "␣kHz,␣timeout␣=␣" + Timeout + "␣
    ms).\r\n");
424             port_opened = true;
425         }
426
427         if (testPlan.FileName != null && testPlan.FileName != String.Empty
    && ReportLevel != MainForm.report_level.Disabled)
428         {
429             report_filename = testPlan.FileName.Substring(0, testPlan.
    FileName.Length - 3) + ".log";
```

```
430             try
431             {
432                 reportWriter = new StreamWriter(report_filename, false); // Do
    not append, overwrite
433             }
434             catch (IOException)
435             {
436                 mainForm.ConsoleAppendText("Cannot␣write␣to␣report␣file␣!\
    r\n");
437                 report_filename = null;
438                 reportWriter = null;
439             }
440         }
441         if (reportWriter != null)
442         {
443             reportWriter.WriteLine("Scan␣Engine␣Test␣Program␣version␣" +
    Assembly.GetExecutingAssembly().GetName().Version);
444             reportWriter.WriteLine("
    ----------------------------------------" +
445             "
    ----------------------------------------"
    );
446             reportWriter.WriteLine("Execution␣date␣and␣time:␣" + startTime)
    ;
447             if (TesterName != null && TesterName != String.Empty)
448                 reportWriter.WriteLine("Tester␣Name:␣" + TesterName);
449             else
450                 reportWriter.WriteLine("Tester␣Name:␣" + Environment.
    UserName);
451         }
452
453         byte[] serialnumber = new byte[16];
454         status = CMD_get_camera_param(0x0001, serialnumber);
455         if (status == 0)
456         {
457             string serialnumber_string = Encoding.UTF8.GetString(
    serialnumber);
458             if (serialnumber_string.Length > 0)
459             {
```

```
460                 PrintAndLogMessage(reportWriter, "Serial␣number:␣" +
    serialnumber_string);
461                 mainForm.ConsoleAppendText("\r\n");
462               }
463             }
464
465             byte[] fwversionreport = new byte[12];
466             // CMD_get_camera_param(0x000a, fwversionreport, 12);
467             status = CMD_get_camera_param(0x000a, fwversionreport);
468             if (status == 0)
469             {
470               string fwversionreport_string = Encoding.UTF8.GetString(
    fwversionreport);
471               if (fwversionreport_string.Length > 0)
472               {
473                 PrintAndLogMessage(reportWriter, "Firmware␣version:␣" +
    fwversionreport_string);
474                 mainForm.ConsoleAppendText("\r\n");
475               }
476             }
477
478             byte[] familyid = new byte[8];
479             // CMD_get_camera_param(0x03f7, familyid, 8);
480             status = CMD_get_camera_param(0x03f7, familyid);
481             if (status == 0)
482             {
483               string familyid_string = Encoding.UTF8.GetString(familyid);
484               if (familyid_string.Length > 0)
485               {
486                 PrintAndLogMessage(reportWriter, "Family␣ID:␣" +
    familyid_string);
487                 mainForm.ConsoleAppendText("\r\n");
488               }
489             }
490
491             if (reportWriter != null)
492             {
493               reportWriter.WriteLine("
    ----------------------------------------------------------------
    " +
```

```
494             "
        ----------------------------------------"
        );
495             if (testPlan.Name != null && testPlan.Name != String.Empty)
496                 reportWriter.WriteLine("Test␣Plan␣Name:␣" + testPlan.Name);
497             reportWriter.WriteLine("Test␣Plan␣Filename:␣" + testPlan.
        FileName);
498             reportWriter.WriteLine("
        ----------------------------------------
        " +
499             "
        ----------------------------------------"
        );
500         }
501
502         // Run the actual test execution
503         relative_position = 0;
504         for (int i = 0; i < testPlan.Count; i++)
505         {
506             number_of_iterations = 1;
507             iteration_size = 1;
508             if (testPlan[i].GetType().Name == "TestIterator")
509             {
510                 number_of_iterations = ((TestIterator)testPlan[i]).LoopIterations;
511                 iteration_size = ((TestIterator)testPlan[i]).LoopSize;
512                 if (i < testPlan.Count - 1)
513                     i++; // Skip to next test plan element
514                 else
515                     break;
516             }
517
518             if (testPlan[i].GetType().Name == "TestCase")
519                 relative_position++;
520
521             while (number_of_iterations > 0)
522             {
523                 number_of_iterations--;
524
525                 for (int k = 0; k < iteration_size; k++)
526                 {
527                     if (i + k >= testPlan.Count)
```

```
528                 {
529                     PrintAndLogMessage(reportWriter, "Warning: size of
        iteration beginning at position index " + (i − 1) + " goes beyond
        actual test plan length ! Breaking loop at last test case.");
530                     break;
531                 }
532
533                 if (testPlan[i + k].GetType().Name != "TestCase")
534                 {
535                     PrintAndLogMessage(reportWriter, "Error at test plan
        element " + (i + k) + ": expected test case but found another element
        type !");
536                     PrintAndLogMessage(reportWriter, "Cannot proceed.
        Aborting execution...");
537                     RequestStop();
538                 }
539
540                 if (_shouldStop)
541                     break;
542
543                 testCase = (TestCase)testPlan[i + k];
544
545                 for (int j = 0; j < testCase.Count; j++)
546                 {
547                     // If step−by−step execution is enabled, pause...
548                     if (_stepbystep)
549                         _shouldPause = true;
550
551                     // Loop until paused
552                     while (_shouldPause)
553                     {
554                         paused = true;
555                         if (_shouldStop)
556                             break;
557                     }
558
559                     paused = false;
560
561                     if (_shouldStop)
562                         break;
563
```

```
564                     testStep = (TestStep)testCase[j];
565
566                     // After each step execution, the property testStep.Result
    should be set to 1 (pass) or 0 (fail) or −1 (not executed)
567
568                     string operationname = testStep.GetType().Name;
569                     string messaboxcaption;
570                     switch (operationname)
571                     {
572                        case "Sleep":
573                           Sleep sleep = (Sleep)testStep;
574                           Thread.Sleep(sleep.Duration);
575                           testStep.Result = 1; // Always succeeds
576                           break;
577                        case "SendCommand":
578                           //if (!port_opened)
579                           //{
580                           //   testStep.Result = 0;
581                           //   testStep.failure_reason = "Cannot send a command
    without an open I2C port !";
582                           //   CheckOnFailureCondition();
583                           //   break;
584                           //}
585                           SendCommand sendCommand = (SendCommand)
    testStep;
586                           StatusResponse expected_status;
587                           ExitStatus exp_status;
588                           string hexcommand = sendCommand.
    GetHexCommand();
589                           string hexparameters = sendCommand.
    GetHexParameters();
590                           string lsb_string;
591                           string msb_string;
592                           string address_string;
593                           StringBuilder buffer_to_stringbuilder = new
    StringBuilder();
594                           byte lsb_out;
595                           byte msb_out;
596                           byte data;
597                           byte address;
598                           byte length_bytes = (byte)(hexparameters.Length / 2);
```

```
599                         byte[] argCmdB = new byte[length_bytes];
600
601                         new_I2CAddress = UInt16.Parse(sendCommand.
    GetI2CAddress(), NumberStyles.AllowHexSpecifier);
602                         port_opened = HandleI2CAddressChange(
    new_I2CAddress);
603
604                         expected_status = sendCommand.GetExpectedStatus();
605
606                         switch (hexcommand)
607                         {
608                             case "00": // Custom command
609                                 if (hexparameters == String.Empty)
610                                 {
611                                     testStep.Result = 1;
612                                     break;
613                                 }
614                                 if (hexparameters.Length % 2 != 0)
615                                 {
616                                     testStep.Result = 0;
617                                     testStep.failure_reason = "Wrong␣custom␣
    command␣format␣(odd␣parameter␣length).";
618                                     CheckOnFailureCondition();
619                                     break;
620                                 }
621                                 hexcommand = hexparameters.Substring(0, 2);
622                                 for (int l = 0; l < length_bytes - 1; l++)
623                                     argCmdB[l] = byte.Parse(hexparameters.
    Substring(l * 2 + 2, 2), System.Globalization.NumberStyles.HexNumber);
624
625                                 try
626                                 {
627                                     status = CMD_Generic(byte.Parse(
    hexcommand, System.Globalization.NumberStyles.HexNumber), argCmdB,
     (byte)(length_bytes - 1));
628                                 }
629                                 catch (EntryPointNotFoundException)
630                                 {
631                                     EntryPointNotFound_Exit();
632                                 }
633                                 break;
```

```
634                             case "23":
635                                 if (hexparameters == String.Empty ||
        hexparameters.Length != 2)
636                                 {
637                                     testStep.Result = 0;
638                                     testStep.failure_reason = "Wrong␣command␣
        parameter␣format.";
639                                     CheckOnFailureCondition();
640                                     break;
641                                 }
642                                 try
643                                 {
644                                     status = CMD_Restore_FactoryDefaults(byte.
        Parse(hexparameters, System.Globalization.NumberStyles.HexNumber));
645                                 }
646                                 catch (EntryPointNotFoundException)
647                                 {
648                                     EntryPointNotFound_Exit();
649                                 }
650                                 break;
651                             case "30":
652                                 if (hexparameters == String.Empty ||
        hexparameters.Length != 6)
653                                 {
654                                     testStep.Result = 0;
655                                     testStep.failure_reason = "Wrong␣command␣
        parameter␣format.";
656                                     CheckOnFailureCondition();
657                                     break;
658                                 }
659                                 lsb_string = hexparameters.Substring(4, 2);
660                                 msb_string = hexparameters.Substring(2, 2);
661                                 address_string = hexparameters.Substring(0, 2);
662                                 lsb_out = byte.Parse(lsb_string, System.
        Globalization.NumberStyles.HexNumber);
663                                 msb_out = byte.Parse(msb_string, System.
        Globalization.NumberStyles.HexNumber);
664                                 address = byte.Parse(address_string, System.
        Globalization.NumberStyles.HexNumber);
665                                 try
666                                 {
```

```
667                                 status = CMD_setImageSensorRegister(address,
        lsb_out, msb_out);
668                             }
669                             catch (EntryPointNotFoundException)
670                             {
671                                 EntryPointNotFound_Exit();
672                             }
673                             break;
674                         case "31":
675                             if (hexparameters == String.Empty ||
        hexparameters.Length != 2)
676                             {
677                                 testStep.Result = 0;
678                                 testStep.failure_reason = "Wrong␣command␣
        parameter␣format.";
679                                 CheckOnFailureCondition();
680                                 break;
681                             }
682                             address_string = hexparameters.Substring(0, 2);
683                             address = byte.Parse(address_string, System.
        Globalization.NumberStyles.HexNumber);
684                             lsb_out = msb_out = 0;
685                             try
686                             {
687                                 status = CMD_getImageSensorRegister(address,
        out lsb_out, out msb_out);
688                             }
689                             catch (EntryPointNotFoundException)
690                             {
691                                 EntryPointNotFound_Exit();
692                             }
693                             if (status == 0)
694                                 mainForm.ConsoleAppendText("
        CMD_getImageSensorRegister:␣lsb␣=␣" + lsb_out + ",␣msb␣=␣" +
        msb_out + "\r\n");
695                             break;
696                         case "34":
697                             if (hexparameters == String.Empty ||
        hexparameters.Length != 2)
698                             {
699                                 testStep.Result = 0;
```

```
700                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
701                                 CheckOnFailureCondition();
702                                 break;
703                             }
704                             try
705                             {
706                                 status = CMD_Illumination_Delay(uint.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
707                             }
708                             catch (EntryPointNotFoundException)
709                             {
710                                 EntryPointNotFound_Exit();
711                             }
712                             break;
713                         case "35":
714                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
715                             {
716                                 testStep.Result = 0;
717                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
718                                 CheckOnFailureCondition();
719                                 break;
720                             }
721                             try
722                             {
723                                 status = CMD_AimControl(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
724                             }
725                             catch (EntryPointNotFoundException)
726                             {
727                                 EntryPointNotFound_Exit();
728                             }
729                             break;
730                         case "37":
731                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
732                             {
733                                 testStep.Result = 0;
```

```
734                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
735                                 CheckOnFailureCondition();
736                                 break;
737                             }
738                             try
739                             {
740                                 status = CMD_CameraReset(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
741                             }
742                             catch (EntryPointNotFoundException)
743                             {
744                                 EntryPointNotFound_Exit();
745                             }
746                             break;
747                         case "38":
748                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
749                             {
750                                 testStep.Result = 0;
751                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
752                                 CheckOnFailureCondition();
753                                 break;
754                             }
755                             try
756                             {
757                                 status = CMD_CameraStart(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
758                             }
759                             catch (EntryPointNotFoundException)
760                             {
761                                 EntryPointNotFound_Exit();
762                             }
763                             break;
764                         case "39":
765                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
766                             {
767                                 testStep.Result = 0;
```

```
768                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
769                                 CheckOnFailureCondition();
770                                 break;
771                             }
772                             try
773                             {
774                                 status = CMD_IlluminationEnable(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
775                             }
776                             catch (EntryPointNotFoundException)
777                             {
778                                 EntryPointNotFound_Exit();
779                             }
780                             break;
781                         case "3B":
782                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
783                             {
784                                 testStep.Result = 0;
785                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
786                                 CheckOnFailureCondition();
787                                 break;
788                             }
789                             try
790                             {
791                                 status = CMD_SetCameraMode(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
792                             }
793                             catch (EntryPointNotFoundException)
794                             {
795                                 EntryPointNotFound_Exit();
796                             }
797                             break;
798                         case "3C":
799                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
800                             {
801                                 testStep.Result = 0;
```

```
802                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
803                                 CheckOnFailureCondition();
804                                 break;
805                             }
806                             try
807                             {
808                                 status = CMD_SensorSetBinning(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
809                             }
810                             catch (EntryPointNotFoundException)
811                             {
812                                 EntryPointNotFound_Exit();
813                             }
814                             break;
815                         case "3F":
816                             if (hexparameters == String.Empty ||
    hexparameters.Length != 2)
817                             {
818                                 testStep.Result = 0;
819                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
820                                 CheckOnFailureCondition();
821                                 break;
822                             }
823                             try
824                             {
825                                 status = CMD_SetPowerLevel(int.Parse(
    hexparameters, System.Globalization.NumberStyles.HexNumber));
826                             }
827                             catch (EntryPointNotFoundException)
828                             {
829                                 EntryPointNotFound_Exit();
830                             }
831                             break;
832                         case "40":
833                             if (hexparameters == String.Empty ||
    hexparameters.Length != 4)
834                             {
835                                 testStep.Result = 0;
```

```
836                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
837                                 CheckOnFailureCondition();
838                                 break;
839                             }
840                             try
841                             {
842                                 status = CMD_get_camera_param(ushort.Parse
    (hexparameters, System.Globalization.NumberStyles.HexNumber), buffer);
843                             }
844                             catch (EntryPointNotFoundException)
845                             {
846                                 EntryPointNotFound_Exit();
847                             }
848                             if (status == 0)
849                             {
850                                 count = 0;
851                                 while (buffer[count] != 0 && count <
    MAX_ANSWER_LENGTH)
852                                 {
853                                     buffer_to_stringbuilder.Append((char)buffer[
    count]);
854                                     count++;
855                                 }
856                                 mainForm.ConsoleAppendText("Receive␣
    buffer:␣" + buffer_to_stringbuilder + "\r\n");
857                             }
858                             break;
859                         case "41":
860                             if (hexparameters == String.Empty ||
    hexparameters.Length < 6)
861                             {
862                                 testStep.Result = 0;
863                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
864                                 CheckOnFailureCondition();
865                                 break;
866                             }
867                             count = hexparameters.Length − 4;
868                             do
869                             {
```

```
870                                 data = byte.Parse(hexparameters.Substring(
    hexparameters.Length − count, 2), System.Globalization.NumberStyles.
    HexNumber);
871                                 buffer[(hexparameters.Length − 4 − count) / 2]
    = data;
872                                 count = count − 2;
873                             } while (count > 0);
874                             try
875                             {
876                                 status = CMD_set_camera_param(ushort.Parse(
    hexparameters.Substring(0, 4), System.Globalization.NumberStyles.
    HexNumber), buffer);
877                             }
878                             catch (EntryPointNotFoundException)
879                             {
880                                 EntryPointNotFound_Exit();
881                             }
882                             break;
883                         case "42":
884                             if (hexparameters == String.Empty ||
    hexparameters.Length != 6)
885                             {
886                                 testStep.Result = 0;
887                                 testStep.failure_reason = "Wrong␣command␣
    parameter␣format.";
888                                 CheckOnFailureCondition();
889                                 break;
890                             }
891                             buffer[0] = byte.Parse(hexparameters.Substring(0,
    2), System.Globalization.NumberStyles.HexNumber);
892                             buffer[1] = byte.Parse(hexparameters.Substring(2,
    2), System.Globalization.NumberStyles.HexNumber);
893                             buffer[2] = byte.Parse(hexparameters.Substring(4,
    2), System.Globalization.NumberStyles.HexNumber);
894                             try
895                             {
896                                 status = CMD_enterBootloader(buffer);
897                             }
898                             catch (EntryPointNotFoundException)
899                             {
900                                 EntryPointNotFound_Exit();
```

```
901                                 }
902                                 break;
903                             default:
904                                 if (hexparameters.Length % 2 != 0)
905                                 {
906                                     testStep.Result = 0;
907                                     testStep.failure_reason = "Wrong␣custom␣
    command␣format␣(odd␣parameter␣length).";
908                                     CheckOnFailureCondition();
909                                     break;
910                                 }
911                                 for (int l = 0; l < length_bytes; l++)
912                                     argCmdB[l] = byte.Parse(hexparameters.
    Substring(l * 2, 2), System.Globalization.NumberStyles.HexNumber);
913 
914                                 try
915                                 {
916                                     status = CMD_Generic(byte.Parse(
    hexcommand, System.Globalization.NumberStyles.HexNumber), argCmdB,
     length_bytes);
917                                 }
918                                 catch (EntryPointNotFoundException)
919                                 {
920                                     EntryPointNotFound_Exit();
921                                 }
922                                 break;
923                         } // switch(hexcommand)
924 
925                         if (testStep.Result == -1) // Test result has not been
    determined yet
926                         {
927                             if (expected_status == StatusResponse.NoCheck)
928                             {
929                                 testStep.Result = 1;
930                             }
931                             else
932                             {
933                                 switch (expected_status)
934                                 {
935                                     case StatusResponse.Ack:
936                                         exp_status = ExitStatus.CMD_OK;
```

```
937                                     break;
938                                 case StatusResponse.Nack:
939                                     exp_status = ExitStatus.CMD_NAK;
940                                     break;
941                                 case StatusResponse.ChecksumError:
942                                     exp_status = ExitStatus.CMD_CKS_ERR;
943                                     break;
944                                 case StatusResponse.Watchdog:
945                                     exp_status = ExitStatus.CMD_WD_RESET;
946                                     break;
947                                 default:
948                                     exp_status = ExitStatus.CMD_OK;
949                                     break;
950                             }
951
952                             // Check if the actual answer to the command is the
    same as the expected status
953                             if ((ExitStatus)status == exp_status)
954                                 testStep.Result = 1;
955                             else
956                             {
957                                 testStep.Result = 0;
958                                 testStep.failure_reason = "Unexpected␣answer␣
    from␣device␣(" + Interpret_CMD_Status(status) + ").";
959                                 CheckOnFailureCondition();
960                             }
961                         }
962                     }
963
964                     port_opened = HandleI2CAddressChange(defaultAddr)
    ;
965                     break;
966                 case "LoadImage":
967                     LoadImage loadImage = (LoadImage)testStep;
968                     mainForm.LoadImage(ref loadImage);
969                     if (loadImage.Result == 0)
970                         CheckOnFailureCondition();
971                     break;
972                 case "CaptureImage":
973                     CaptureImage captureImage = (CaptureImage)testStep;
974                     mainForm.CaptureImage(ref captureImage);
```

```
975                             if (captureImage.Result == 0)
976                                 CheckOnFailureCondition();
977                             break;
978                         case "AnalyzeImage":
979                             AnalyzeImage analyzeImage = (AnalyzeImage)testStep
    ;
980                             mainForm.AnalyzeImage(ref analyzeImage);
981                             if (analyzeImage.Result == 0)
982                                 CheckOnFailureCondition();
983                             break;
984                         case "UserMessage":
985                             UserMessage userMessage = (UserMessage)testStep;
986                             if (userMessage.ImageFileName != null &&
    userMessage.ImageFileName != String.Empty)
987                                 mainForm.ShowUserMessageImage(userMessage.
    ImageFileName);
988                             if (userMessage.m_Description != null &&
    userMessage.m_Description != String.Empty)
989                                 messaboxcaption = "User␣Message:␣" +
    userMessage.m_Description;
990                             else
991                                 messaboxcaption = "User␣Message";
992                             MessageBox.Show(userMessage.GetUserMessage(),
    messaboxcaption, MessageBoxButtons.OK);
993                             testStep.Result = 1;
994                             break;
995                         case "UserFeedback":
996                             UserFeedback userFeedback = (UserFeedback)testStep;
997                             if (userFeedback.m_Description != null &&
    userFeedback.m_Description != String.Empty)
998                                 messaboxcaption = "User␣Feedback␣request:␣" +
    userFeedback.m_Description;
999                             else
1000                                messaboxcaption = "User␣Feedback␣request";
1001                            DialogResult dialogResult = MessageBox.Show(
    userFeedback.Message, messaboxcaption, MessageBoxButtons.YesNo);
1002                            if (dialogResult == DialogResult.Yes)
1003                                testStep.Result = userFeedback.GetFeedback() ? 1 :
    0;
1004                            else if (dialogResult == DialogResult.No)
```

```
1005                                testStep.Result = userFeedback.GetFeedback() ? 0 :
        1;
1006                            if (testStep.Result == 0)
1007                                CheckOnFailureCondition();
1008                            break;
1009                        default:
1010                            mainForm.ConsoleAppendText("Error:␣Trying␣to␣
        execute␣an␣unknown␣operation␣!\r\n");
1011                            testStep.Result = 0;
1012                            testStep.failure_reason = "Unknown␣operation␣
        requested.";
1013                            CheckOnFailureCondition();
1014                            break;
1015                    } // switch(operationname)
1016
1017                    Thread.Sleep(1);
1018
1019                    if (testStep.Result == 1)
1020                        test_step_success++;
1021                    else if (testStep.Result == 0)
1022                    {
1023                        if (testStep.failure_reason != null && testStep.
        failure_reason.Length > 0)
1024                            PrintAndLogMessage(reportWriter, "test␣step[" + (j +
        1) + "/" + testCase.Count + "]␣failed:␣" + testStep.failure_reason);
1025                        else
1026                            PrintAndLogMessage(reportWriter, "test␣step[" + (j +
        1) + "/" + testCase.Count + "]␣failed");
1027                        test_step_failures++;
1028                    }
1029                } // for (j)
1030
1031                testCase.CalculateResult();
1032                if (testCase.Result == 1)
1033                {
1034                    test_case_success++;
1035                    PrintAndLogMessage(reportWriter, "test␣case[" + (
        relative_position + k) + "/" + relative_test_cases + "]:␣success");
1036                }
1037                else if (testCase.Result == 0)
1038                {
```

```
1039                        test_case_failures++;
1040                        PrintAndLogMessage(reportWriter, "test␣case[" + (
        relative_position + k) + "/" + relative_test_cases + "]:␣failure");
1041                    }
1042                    else if (testCase.Result == -1)
1043                        PrintAndLogMessage(reportWriter, "test␣case[" + (
        relative_position + k) + "/" + relative_test_cases + "]:␣not␣executed");
1044                } // for (k)
1045
1046                if (_shouldStop)
1047                    break;
1048            }
1049
1050            if (iteration_size > 0)
1051            {
1052                i += iteration_size - 1;
1053                relative_position += iteration_size - 1;
1054            }
1055
1056            if (_shouldStop)
1057                break;
1058        } // for (i)
1059
1060        testPlan.CalculateResult();
1061
1062        RequestStop();
1063    }
1064
1065    // Print the execution statistics and log them to the report file
1066    if (reportWriter != null)
1067        reportWriter.WriteLine("
        ----------------------------------------
        " +
1068            "
        ----------------------------------------"
        );
1069    test_cases_executed = test_case_success + test_case_failures;
1070    if (total_test_cases > 0)
1071    {
1072        percentage_test_case_executed = ((float)test_cases_executed /
        total_test_cases) * 100;
```

```
1073            percentage_test_case_failure = ((float)test_case_failures /
        total_test_cases) * 100;
1074            percentage_test_case_success = ((float)test_case_success /
        total_test_cases) * 100;
1075        }
1076        else
1077        {
1078            percentage_test_case_executed = 100;
1079            percentage_test_case_failure = 0;
1080            percentage_test_case_success = 100;
1081        }
1082        test_steps_executed = test_step_success + test_step_failures;
1083        if (total_test_steps > 0)
1084        {
1085            percentage_test_step_executed = ((float)test_steps_executed /
        total_test_steps) * 100;
1086            percentage_test_step_failure = ((float)test_step_failures /
        total_test_steps) * 100;
1087            percentage_test_step_success = ((float)test_step_success /
        total_test_steps) * 100;
1088        }
1089        else
1090        {
1091            percentage_test_step_executed = 100;
1092            percentage_test_step_failure = 0;
1093            percentage_test_step_success = 100;
1094        }
1095        if (reportWriter != null)
1096            reportWriter.WriteLine("                                        
        Execution Summary");
1097        PrintAndLogMessage(reportWriter, "Executed " + test_cases_executed
        + " out of " + total_test_cases + " test cases (" +
        percentage_test_case_executed + "%).");
1098        PrintAndLogMessage(reportWriter, "- Successful test cases: " +
        test_case_success + " out of " + total_test_cases + " (" +
        percentage_test_case_success + "%).");
1099        PrintAndLogMessage(reportWriter, "- Unsuccessful test cases: " +
        test_case_failures + " out of " + total_test_cases + " (" +
        percentage_test_case_failure + "%).");
1100        PrintAndLogMessage(reportWriter, "- Undetermined test cases: " + (
        total_test_cases - test_cases_executed) + " out of " + total_test_cases + "
```

```
␣(" + (100 − percentage_test_case_executed) + "%).");
    PrintAndLogMessage(reportWriter, "Executed␣" + test_steps_executed
+ "␣out␣of␣" + total_test_steps + "␣test␣steps␣(" +
percentage_test_step_executed + "%).");
    PrintAndLogMessage(reportWriter, "−␣Successful␣test␣steps:␣" +
test_step_success + "␣out␣of␣" + total_test_steps + "␣(" +
percentage_test_step_success + "%).");
    PrintAndLogMessage(reportWriter, "−␣Unsuccessful␣test␣steps:␣" +
test_step_failures + "␣out␣of␣" + total_test_steps + "␣(" +
percentage_test_step_failure + "%).");
    PrintAndLogMessage(reportWriter, "−␣Undetermined␣test␣steps:␣" + (
total_test_steps − test_steps_executed) + "␣out␣of␣" + total_test_steps + "␣
(" + (100 − percentage_test_step_executed) + "%).");
    if (reportWriter != null)
       reportWriter.WriteLine("
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
"+
          "
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−"
);

    if (test_steps_executed == total_test_steps && test_step_failures == 0)
       PrintAndLogMessage(reportWriter, "Test␣Plan␣result:␣success");

    if (test_steps_executed < total_test_steps && test_step_failures == 0)
       PrintAndLogMessage(reportWriter, "Test␣Plan␣result:␣undetermined
");

    if (test_step_failures > 0)
       PrintAndLogMessage(reportWriter, "Test␣Plan␣result:␣failure");

    if (port_opened)
       if (comPortClose() <= 0)
          mainForm.ConsoleAppendText("Error␣closing␣the␣I2C␣host␣
adapter␣!\r\n");
       else
          mainForm.ConsoleAppendText("I2C␣host␣adapter␣closed␣
successfully.\r\n");

    // Calculate total execution duration
    endTime = DateTime.Now;
```

```
1126            execution_duration = endTime − startTime;
1127
1128            // Print the total execution duration
1129            PrintAndLogMessage(reportWriter, "Total␣execution␣duration:␣" +
        execution_duration);
1130
1131            // Close the report file
1132            if (reportWriter != null)
1133                reportWriter.Close();
1134
1135            mainForm.ConsoleAppendText("\r\nExecution␣thread:␣terminating␣
        gracefully.\r\n");
1136         } // Run()
1137
1138         public void RequestPause()
1139         {
1140            if (!_stepbystep)
1141            {
1142               _shouldPause = !_shouldPause;
1143
1144               if (_shouldPause)
1145                  mainForm.ConsoleAppendText("Execution␣thread:␣pausing...\r\n")
        ;
1146               else
1147                  mainForm.ConsoleAppendText("Execution␣thread:␣resuming...\r\n
        ");
1148            }
1149         }
1150
1151         public void RequestStop()
1152         {
1153            if (!_shouldStop)
1154               mainForm.ConsoleAppendText("Execution␣thread:␣stopping...\r\n");
1155
1156            _shouldStop = true;
1157         }
1158
1159         public void RequestDebug()
1160         {
1161            if (_stepbystep == false)
```

```
1162            mainForm.ConsoleAppendText("Execution␣thread:␣Enabling␣debug
        ␣mode␣(step−by−step␣execution)...\r\n");
1163        else
1164            mainForm.ConsoleAppendText("Execution␣thread:␣Stepping␣
        execution...\r\n");
1165
1166        _stepbystep = debug = true;
1167
1168        if (paused)
1169            _shouldPause = false;
1170    }
1171
1172    public void StopDebug()
1173    {
1174        if (_stepbystep == true)
1175            mainForm.ConsoleAppendText("Execution␣thread:␣Disabling␣debug
        ␣mode␣(step−by−step␣execution)...\r\n");
1176
1177        _stepbystep = debug = false;
1178
1179        if (paused)
1180            _shouldPause = false;
1181    }
1182
1183    // Volatile is used as hint to the compiler that this data
1184    // members will be accessed by multiple threads.
1185    private volatile bool _shouldPause;
1186    private volatile bool _shouldStop;
1187    private volatile bool _stepbystep;
1188
1189    private void EntryPointNotFound_Exit()
1190    {
1191        mainForm.ConsoleAppendText("Error␣while␣trying␣to␣load␣dynamic
        −link␣libraries:␣wrong␣version␣of␣the␣device.dll␣library␣!\r\n");
1192        return;
1193    }
1194
1195    private bool HandleI2CAddressChange(ushort new_I2CAddress)
1196    {
1197        int status;
1198
```

```
1199        if (new_I2CAddress == slaveAddr)
1200            return true;
1201        else
1202            mainForm.ConsoleAppendText("I2C␣address␣changed.\r\n");
1203
1204        if (comPortClose() <= 0)
1205            mainForm.ConsoleAppendText("Error␣closing␣the␣I2C␣host␣
        adapter.\r\n");
1206        else
1207            mainForm.ConsoleAppendText("I2C␣host␣adapter␣closed␣
        successfully.\r\n");
1208
1209        status = comPortInit(portName, (uint)Bitrate, (uint)Timeout, (byte)
        new_I2CAddress); // Reopen and reconfigure the I2C host adapter
1210        if (status < 0)
1211        {
1212            mainForm.ConsoleAppendText("Unable␣to␣reopen␣the␣I2C␣host␣
        adapter␣(status:␣" + status + ").\r\n");
1213            return false;
1214        }
1215        else
1216        {
1217            mainForm.ConsoleAppendText("I2C␣host␣adapter␣reopened␣
        successfully.\r\n");
1218            slaveAddr = new_I2CAddress;
1219            return true;
1220        }
1221    }
1222
1223    private string Interpret_CMD_Status(int status)
1224    {
1225        string answer;
1226
1227        switch (status)
1228        {
1229            case 0:
1230                answer = "Acknowledge␣[ACK]";
1231                break;
1232            case -1:
1233                answer = "Checksum␣error";
1234                break;
```

```
1235            case −2:
1236              answer = "Not␣Acknowledge␣[NACK]";
1237              break;
1238            case −3:
1239              answer = "Wrong␣answer␣received␣from␣slave␣device";
1240              break;
1241            case −4:
1242              answer = "Wrong␣command␣parameter";
1243              break;
1244            case −5:
1245              answer = "Wrong␣command␣issued";
1246              break;
1247            case −8:
1248              answer = "Unable␣to␣close␣device";
1249              break;
1250            case −9:
1251              answer = "Invalid␣device␣handle";
1252              break;
1253            case −10:
1254              answer = "Configuration␣error";
1255              break;
1256            case −13:
1257              answer = "Slave␣device␣not␣responding";
1258              break;
1259            case −14:
1260              answer = "Slave␣device␣thermal␣failure";
1261              break;
1262            case −100:
1263              answer = "I2C␣feature␣not␣available";
1264              break;
1265            case −101:
1266              answer = "I2C␣not␣enabled";
1267              break;
1268            case −102:
1269              answer = "I2C␣read␣error";
1270              break;
1271            case −103:
1272              answer = "I2C␣write␣error";
1273              break;
1274            default:
1275              answer = "";
```

```
1276                 break;
1277             }
1278
1279             return answer;
1280         }
1281
1282         private void BreakOrContinue()
1283         {
1284             DialogResult dialogResult = MessageBox.Show("Test␣step␣failed.␣
                Press␣Yes␣to␣Break␣or␣No␣to␣Continue.", "Break␣or␣Continue␣?",
                MessageBoxButtons.YesNo);
1285             if (dialogResult == DialogResult.Yes)
1286             {
1287                 RequestStop();
1288             }
1289         }
1290
1291         private void CheckOnFailureCondition()
1292         {
1293             if (OnFailure == MainForm.on_failure.Ask)
1294                 BreakOrContinue();
1295             else if (OnFailure == MainForm.on_failure.Break)
1296                 RequestStop();
1297         }
1298
1299         private void PrintAndLogMessage(StreamWriter reportWriter, string
                message)
1300         {
1301             if (reportWriter != null)
1302                 reportWriter.WriteLine(message);
1303             mainForm.ConsoleAppendText(message + "\r\n");
1304         }
1305     }
1306 }
```

Appendice F

Codice sorgente per la libreria Framegrabber

```
1  ï»¿/*
2   * Frame−grabber library for the Scan Engine Test Program.
3   *
4   * Libreria di incapsulamento delle funzioni di acquisizione immagini
5   * tramite scheda Pleora (SDK API versione 3.1.10 e 4.0.6).
6   *
7   * developed by Guido Trentalancia for Datalogic S.p.A.
8   *
9   */
10
11 #if (!PLEORA_API_V4)
12 #define PLEORA_API_V3
13 #endif
14
15 using System;
16 using System.Collections.Generic;
17 using System.Drawing; // for the Bitmap class
18 using System.Linq;
19 using System.Net; // to detect mismatch between framegrabber IP address and
       available NICs
20 using System.Net.NetworkInformation; // to detect mismatch between
       framegrabber IP address and available NICs
21 using System.Net.Sockets; // to detect mismatch between framegrabber IP
       address and available NICs
22 using System.Text;
```

```
23  using System.Windows.Forms; // for DialogResult
24
25  #if (PLEORA_API_V3) || (PLEORA_API_V4)
26  using PvDotNet;
27  using PvGUIDotNet; // for PvDeviceFinderForm (Pleora device selection
        window)
28  #endif
29
30  namespace FramegrabberLibrary
31  {
32     // A delegate type for hooking up captured image change notifications.
33     public delegate void ImageChangedEventHandler(object sender, EventArgs
         e);
34
35     public class Framegrabber
36     {
37        public const int DefaultFrameGrabberPixelFormat = 17301505; // Mode8
          =17301505
38        public const long DefaultFrameGrabberAcquisitionMode = 0; //
          AcquisitionMode=Continuous -> 0
39
40        public const long DefaultFrameGrabberWidth = 752;
41        public const long DefaultFrameGrabberHeight = 480;
42
43        long m_FrameGrabberWidth = DefaultFrameGrabberWidth;
44        long m_FrameGrabberHeight = DefaultFrameGrabberHeight;
45
46        public long FrameGrabberWidth
47        {
48           get
49           {
50              return m_FrameGrabberWidth;
51           }
52           set
53           {
54              if (value > 0)
55                 m_FrameGrabberWidth = value;
56           }
57        }
58
59        public long FrameGrabberHeight
```

```
60          {
61              get
62              {
63                  return m_FrameGrabberHeight;
64              }
65              set
66              {
67                  if (value > 0)
68                      m_FrameGrabberHeight = value;
69              }
70          }
71
72          bool m_NeedsReconnect;
73
74          private Bitmap m_BackImage = null;
75
76          public Bitmap image
77          {
78              get
79              {
80                  return m_BackImage;
81              }
82          }
83
84          public event ImageChangedEventHandler ImageChanged;
85
86          // Invoke the ImageChanged event; called whenever the captured image
        changes
87          protected virtual void OnImageChanged(EventArgs e)
88          {
89              if (ImageChanged != null)
90                  ImageChanged(this, e);
91          }
92
93          PvDeviceFinderForm m_Finder = null;
94
95      #if (PLEORA_API_V4)
96          private PvDeviceInfo m_DeviceInfo = null;
97      #endif
98
99          // Main application objects: device, stream, pipeline
```

```
100 #if (PLEORA_API_V3)
101         private PvDevice m_Device = new PvDevice();
102         private PvStream m_Stream = new PvStream();
103 #endif
104 #if (PLEORA_API_V4)
105         private PvDevice m_Device = null;
106         private PvStream m_Stream = null;
107 #endif
108
109         private PvPipeline m_Pipeline = null;
110
111         // Acquisition state manager
112         private PvAcquisitionStateManager m_AcquisitionManager = null;
113
114         public Framegrabber()
115         {
116             m_NeedsReconnect = false;
117
118 #if (PLEORA_API_V3)
119             // Create a Pleora pipeline - requires a Pleora stream
120             m_Pipeline = new PvPipeline(m_Stream);
121 #endif
122         }
123
124         public int Select()
125         {
126             // Create a Pleora device finder window object that can be used
127             // to open the device selection dialog
128             m_Finder = new PvDeviceFinderForm();
129
130             // Show the Pleora device finder
131             if ((m_Finder.ShowDialog() != DialogResult.OK) ||
132                 (m_Finder.Selected == null))
133             {
134                 if (m_Finder != null)
135                     m_Finder.Dispose();
136                 m_Finder = null;
137
138                 return -1; // "Capture Image: Framegrabber device has not been
        selected. Cannot connect to a framegrabber device !\r\n"
139             }
```

```
140 #if (PLEORA_API_V4)
141         else
142         {
143            try
144            {
145               m_DeviceInfo = m_Finder.Selected as PvDeviceInfo;
146            }
147            catch (PvException)
148            {
149               Disconnect();
150
151               if (m_Finder != null)
152                  m_Finder.Dispose();
153               m_Finder = null;
154
155               return -2; // "Capture Image: Cannot get framegrabber device
       information.\r\n"
156            }
157         }
158 #endif
159
160         // Connect to the Pleora device
161 #if (PLEORA_API_V3)
162         if (!Connect(m_Finder.Selected))
163 #endif
164 #if (PLEORA_API_V4)
165         if (m_DeviceInfo != null)
166            if (!Connect(m_DeviceInfo))
167 #endif
168         {
169            if (m_Finder != null)
170               m_Finder.Dispose();
171            m_Finder = null;
172
173            return -3; // "Capture Image: Cannot connect to the selected
       framegrabber device !\r\n"
174         }
175
176         if (m_Finder != null)
177            m_Finder.Dispose();
178         m_Finder = null;
```

```
179
180             return 0;
181         }
182
183         private static IPAddress CalculateNetwork(UnicastIPAddressInformation
            addr)
184         {
185             // The mask will be null in some scenarios, like a dhcp address 169.254.x.
            x
186             if (addr.IPv4Mask == null)
187                 return null;
188
189             var ip = addr.Address.GetAddressBytes();
190             var mask = addr.IPv4Mask.GetAddressBytes();
191             var result = new Byte[4];
192             for (int i = 0; i < 4; ++i)
193             {
194                 result[i] = (Byte)(ip[i] & mask[i]);
195             }
196
197             return new IPAddress(result);
198         }
199
200         private static IPAddress CalculateNetwork(byte[] ip, byte[] mask)
201         {
202             var result = new Byte[4];
203             for (int i = 0; i < 4; ++i)
204             {
205                 result[i] = (Byte)(ip[i] & mask[i]);
206             }
207
208             return new IPAddress(result);
209         }
210
211         // Determine whether or not the local IP address used to receive images
            from
212         // the framegrabber actually belongs to a NIC (Network Interface Card).
213         // It provides the netmask of the matching NIC as an output parameter.
214         private bool LocalIPAddressBelongsToNIC(out
            UnicastIPAddressInformation NICaddr)
215         {
```

```
216             NICaddr = null;
217
218     #if (PLEORA_API_V3)
219             if (m_Stream != null && m_Stream.IsOpened)
220     #endif
221     #if (PLEORA_API_V4)
222             if (m_Stream != null && m_Stream.IsOpen)
223     #endif
224             {
225                 IPAddress[] ips;
226
227                 ips = Dns.GetHostAddresses(m_Stream.LocalIPAddress);
228
229                 var nics = NetworkInterface.GetAllNetworkInterfaces();
230                 foreach (var nic in nics)
231                 {
232                     var ipProps = nic.GetIPProperties();
233
234                     // We're only interested in IPv4 addresses.
235                     var ipv4Addrs = ipProps.UnicastAddresses.Where(addr => addr.
            Address.AddressFamily == AddressFamily.InterNetwork);
236
237                     foreach (var addr in ipv4Addrs)
238                     {
239                         foreach (IPAddress ip in ips)
240                             if (ip.Equals(addr.Address))
241                             {
242                                 NICaddr = addr;
243
244                                 return true;
245                             }
246                     }
247                 }
248             }
249
250             return false;
251         }
252
253         // Check that transmitter and receiver are both on the same network
254         private bool IsDeviceOnSameNetwork(UnicastIPAddressInformation
            NICaddr)
```

```
255         {
256 #if (PLEORA_API_V3)
257             if (m_Stream != null && m_Stream.IsOpened && NICaddr != null &&
        NICaddr.IPv4Mask != null)
258 #endif
259 #if (PLEORA_API_V4)
260             if (m_Stream != null && m_Stream.IsOpen && NICaddr != null)
261 #endif
262             {
263                 PvGenInteger mDeviceIPAddress;
264                 IPAddress rxip, txip;
265                 IPAddress rxnetwork = null, txnetwork = null;
266                 IPAddress[] rxips, txips;
267 
268                 try
269                 {
270                     mDeviceIPAddress = m_Stream.Parameters.GetInteger("
        DeviceIPAddress");
271                 }
272                 catch (PvException)
273                 {
274                     mDeviceIPAddress = null;
275                 }
276 
277                 // If the transmitter IP address parameter is not available for some
        reason,
278                 // then assume the transmitter is on the same network as the receiver (
        NIC).
279                 if (mDeviceIPAddress == null)
280                     return true;
281 
282                 rxips = Dns.GetHostAddresses(m_Stream.LocalIPAddress);
283                 txips = Dns.GetHostAddresses(mDeviceIPAddress.ToString());
284 
285                 if (rxips.Length > 0)
286                 {
287                     rxip = rxips[rxips.Length - 1];
288                     rxnetwork = CalculateNetwork(rxip.GetAddressBytes(), NICaddr.
        IPv4Mask.GetAddressBytes());
289                 }
290 
```

```
291             if (txips.Length > 0)
292             {
293                 txip = txips[txips.Length − 1];
294                 txnetwork = CalculateNetwork(txip.GetAddressBytes(), NICaddr.
        IPv4Mask.GetAddressBytes());
295             }
296
297             if (rxips.Length > 0 && txips.Length > 0)
298                 if (rxnetwork.Equals(txnetwork))
299                     return true;
300         }
301
302         return false;
303     }
304
305     public int Capture(long width, long height, int frames, bool saveImage,
        string filename, string folder)
306     {
307         bool keepgoing = true;
308         bool localipaddressmatchesnic;
309         bool deviceonsamenetwork;
310         int configurationresult;
311         int imagescaptured;
312
313         UnicastIPAddressInformation NICaddr;
314
315         FrameGrabberWidth = width;
316         FrameGrabberHeight = height;
317
318         // Check address mismatch between stream object and available network
        interface cards (NICs).
319         // If a mismatch is detected, then the IP address of the framegrabber must
         be configured
320         // again through m_Finder.ShowDialog().
321         localipaddressmatchesnic = LocalIPAddressBelongsToNIC(out NICaddr)
        ;
322
323         // Check that transmitter (framegrabber device) and receiver (host
        computer NIC) are both
324         // on the same network.
325         deviceonsamenetwork = IsDeviceOnSameNetwork(NICaddr);
```

```
326
327         // Connect to the Pleora device, if not already connected or if IP address
        does not match
328         // any of the currently available NIC addresses or if transmitter and
        receiver are not
329         // on the same network.
330 #if (PLEORA_API_V3)
331         if (m_Device == null || (m_Device != null && !m_Device.IsConnected)
        || m_Stream == null || (m_Stream != null && !m_Stream.IsOpened) || !
        localipaddressmatchesnic || !deviceonsamenetwork || m_NeedsReconnect)
332 #endif
333 #if (PLEORA_API_V4)
334         if (m_Device == null || (m_Device != null && !m_Device.IsConnected)
        || m_Stream == null || (m_Stream != null && !m_Stream.IsOpen) || !
        localipaddressmatchesnetwork || !deviceonsamenetwork ||
        m_NeedsReconnect)
335 #endif
336         {
337 #if (PLEORA_FAIL_IF_NOT_SELECTED)
338             return -1; // "Capture Image: Framegrabber device has not been
        selected. Cannot connect to a framegrabber device !\r\n"
339         }
340 #else
341             // Create a Pleora device finder window object that can be used
342             // to open the device selection dialog
343             m_Finder = new PvDeviceFinderForm();
344
345             // Show the Pleora device finder
346 #if (PLEORA_API_V3)
347             if ((m_Finder.ShowDialog() != DialogResult.OK) ||
348                 (m_Finder.Selected == null))
349             {
350                 if (m_Finder != null)
351                     m_Finder.Dispose();
352                 m_Finder = null;
353
354                 return -1; // "Capture Image: Framegrabber device has not been
        selected. Cannot connect to a framegrabber device !\r\n"
355             }
356 #endif
357 #if (PLEORA_API_V4)
```

```
358             DialogResult dialogResult = DialogResult.None;
359             try
360             {
361                 if (m_Finder != null)
362                     dialogResult = m_Finder.ShowDialog();
363             }
364             catch (PvException)
365             {
366                 if (m_Finder != null)
367                     m_Finder.Dispose();
368                 m_Finder = null;
369 
370                 return -2; // "Capture Image: Cannot select a framegrabber
    device. Try disabling the firewall.\r\n"
371             }
372             if (dialogResult == DialogResult.OK && m_Finder.Selected != null
    )
373             {
374                 try
375                 {
376                     m_DeviceInfo = m_Finder.Selected as PvDeviceInfo;
377                 }
378                 catch (PvException)
379                 {
380                     if (m_Finder != null)
381                         m_Finder.Dispose();
382                     m_Finder = null;
383 
384                     return -3; // "Capture Image: Cannot get framegrabber
    device information.\r\n"
385                 }
386             }
387             else
388             {
389                 if (m_Finder != null)
390                     m_Finder.Dispose();
391                 m_Finder = null;
392 
393                 return -1; // "Capture Image: Framegrabber device has not been
     selected. Cannot connect to a framegrabber device !\r\n"
394             }
```

```
395 #endif
396
397             // Connect to the Pleora device
398 #if (PLEORA_API_V3)
399             keepgoing = Connect(m_Finder.Selected);
400 #endif
401 #if (PLEORA_API_V4)
402             if (m_DeviceInfo != null)
403                 keepgoing = Connect(m_DeviceInfo);
404 #endif
405
406             if (m_Finder != null)
407                 m_Finder.Dispose();
408             m_Finder = null;
409         }
410 #endif
411
412         if (m_Device == null || !keepgoing)
413         {
414             return -4; // "Capture Image: Cannot connect to selected
        framegrabber device !\r\n"
415         }
416
417         // Set the resolution, pixel format (Mono8) and acquisition mode (
        Continuous)
418         configurationresult = SetParameters();
419         keepgoing = (configurationresult == 0);
420
421         if (!keepgoing)
422             return -8; // "Capture Image: Framegrabber configuration failed !\r\
        n"
423
424         keepgoing = StartStreaming();
425
426         if (!keepgoing)
427             return -5; // "Capture Image: Cannot enable streaming from
        framegrabber device !\r\n"
428
429         imagescaptured = StartAcquisition(frames, saveImage, filename, folder);
430         keepgoing = (imagescaptured > 0);
431
```

```
432             StopAcquisition();
433
434             StopStreaming();
435
436             if (!keepgoing)
437             {
438                 if (imagescaptured == -1)
439                     return -7; // "Capture Image: Image acquisition unsuccessful:
        could not start acquisition manager !\r\n"
440                 else // imagescaptured == 0
441                     return -6; // "Capture Image: Image acquisition unsuccessful !\r\n
        "
442             }
443
444             return imagescaptured;
445         }
446
447         // Connect to the selected Pleora device
448         private bool Connect(PvDeviceInfo aDI)
449         {
450             // Just in case we came here still connected...
451             Disconnect();
452
453             try
454             {
455                 // Connect to device using device info
456 #if (PLEORA_API_V3)
457                 m_Device.Connect(aDI);
458 #endif
459 #if (PLEORA_API_V4)
460                 m_Device = PvDevice.CreateAndConnect(aDI);
461 #endif
462             }
463             catch (PvException)
464             {
465                 Disconnect();
466
467                 return false;
468             }
469
470             if (m_Device == null)
```

```
471             return false;
472
473         try
474         {
475             // Open stream using device IP address
476 #if (PLEORA_API_V3)
477             m_Stream.Open(aDI.IPAddress);
478 #endif
479 #if (PLEORA_API_V4)
480             m_Stream = PvStream.CreateAndOpen(aDI.ConnectionID);
481 #endif
482         }
483         catch (PvException)
484         {
485             Disconnect();
486
487             return false;
488         }
489
490         if (m_Stream == null)
491             return false;
492
493         try
494         {
495 #if (PLEORA_API_V3)
496             // Negotiate packet size
497             m_Device.NegotiatePacketSize();
498
499             // Set stream destination in our stream object
500             m_Device.SetStreamDestination(m_Stream.LocalIPAddress,
        m_Stream.LocalPort);
501 #endif
502 #if (PLEORA_API_V4)
503             PvDeviceGEV lDGEV = m_Device as PvDeviceGEV;
504             if (lDGEV != null)
505             {
506                 // Negotiate packet size
507                 lDGEV.NegotiatePacketSize();
508                 // Set stream destination.
509                 PvStreamGEV lSGEV = m_Stream as PvStreamGEV;
```

```
510                 lDGEV.SetStreamDestination(lSGEV.LocalIPAddress, lSGEV.
        LocalPort);
511             }
512 #endif
513         }
514         catch (PvException)
515         {
516             // Failure at some point, abort.
517             Disconnect();
518 
519             return false;
520         }
521 
522         if (m_Device != null && m_Device.IsConnected)
523         {
524             // Connect link disconnection handler
525             m_Device.OnLinkDisconnected += new OnLinkDisconnectedHandler
        (OnLinkDisconnected);
526         }
527 
528 #if (PLEORA_API_V3)
529         if (m_Device.IsConnected && m_Stream.IsOpened)
530         {
531 #endif
532 #if (PLEORA_API_V4)
533         if (m_Device.IsConnected && m_Stream.IsOpen)
534         {
535             // Create a Pleora pipeline − requires a Pleora stream
536             m_Pipeline = new PvPipeline(m_Stream);
537 #endif
538             m_NeedsReconnect = false;
539 
540             return true;
541         }
542         else
543             return false;
544       }
545 
546       public void Disconnect()
547       {
548         // If streaming, stop streaming
```

```
549 #if (PLEORA_API_V3)
550             if (m_Stream != null && m_Stream.IsOpened)
551 #endif
552 #if (PLEORA_API_V4)
553             if (m_Stream != null && m_Stream.IsOpen)
554 #endif
555             {
556                 StopStreaming();
557                 m_Stream.Close();
558 #if (PLEORA_API_V4)
559                 m_Stream = null;
560 #endif
561             }
562 
563             if (m_Device != null && m_Device.IsConnected)
564             {
565                 // Disconnect events
566                 m_Device.OnLinkDisconnected -= new OnLinkDisconnectedHandler
        (OnLinkDisconnected);
567 
568                 m_Device.Disconnect();
569 #if (PLEORA_API_V4)
570                 m_Device = null;
571 #endif
572             }
573         }
574 
575         // Direct device disconnect handler.
576         private void OnLinkDisconnected(PvDevice aDevice)
577         {
578             m_NeedsReconnect = true;
579         }
580 
581         private int SetParameters()
582         {
583             bool cannotaccessresolution = false;
584             bool invalidsetting = false;
585             int result = 0;
586             PvGenInteger mWidth, mHeight;
587             PvGenEnum mPixelFormat, mAcquisitionMode;
588 
```

```
589             if (m_Device != null)
590             {
591                 try
592                 {
593 #if (PLEORA_API_V3)
594                     mWidth = m_Device.GenParameters.GetInteger("Width");
595 #endif
596 #if (PLEORA_API_V4)
597                     mWidth = m_Device.Parameters.GetInteger("Width");
598 #endif
599                 }
600                 catch (PvException)
601                 {
602                     mWidth = null;
603 
604                     cannotaccessresolution = true;
605                 }
606                 try
607                 {
608 #if (PLEORA_API_V3)
609                     mHeight = m_Device.GenParameters.GetInteger("Height");
610 #endif
611 #if (PLEORA_API_V4)
612                     mHeight = m_Device.Parameters.GetInteger("Height");
613 #endif
614                 }
615                 catch
616                 {
617                     mHeight = null;
618 
619                     cannotaccessresolution = true;
620                 }
621 
622                 if (cannotaccessresolution)
623                     result += 128; // "Capture Image: Cannot access framegrabber
        register for resolution.\r\n"
624 
625                 try
626                 {
627 #if (PLEORA_API_V3)
```

```
628                 mPixelFormat = m_Device.GenParameters.GetEnum("PixelFormat"
        );
629 #endif
630 #if (PLEORA_API_V4)
631                 mPixelFormat = m_Device.Parameters.GetEnum("PixelFormat");
632 #endif
633             }
634             catch (PvException)
635             {
636                 mPixelFormat = null;
637 
638                 result += 64; // "Capture Image: Cannot access framegrabber
        register for pixel format.\r\n"
639             }
640             try
641             {
642 #if (PLEORA_API_V3)
643                 mAcquisitionMode = m_Device.GenParameters.GetEnum("
        AcquisitionMode");
644 #endif
645 #if (PLEORA_API_V4)
646                 mAcquisitionMode = m_Device.Parameters.GetEnum("
        AcquisitionMode");
647 #endif
648             }
649             catch (PvException)
650             {
651                 mAcquisitionMode = null;
652 
653                 result += 32; // "Capture Image: Cannot access framegrabber
        register for acquisition mode.\r\n"
654             }
655 
656             // Set the Pleora device resolution
657             if (mWidth != null)
658             {
659                 try
660                 {
661                     if (FrameGrabberWidth >= mWidth.Min &&
        FrameGrabberWidth <= mWidth.Max)
662                         mWidth.Value = FrameGrabberWidth;
```

```
663             }
664             catch (PvException)
665             {
666                 invalidsetting = true;
667 
668                 result += 16; // "Capture Image: Cannot set framegrabber
    resolution parameter (width).\r\n"
669             }
670         }
671 
672         if (mHeight != null)
673         {
674             try
675             {
676                 if (FrameGrabberHeight >= mHeight.Min &&
    FrameGrabberHeight <= mHeight.Max)
677                     mHeight.Value = FrameGrabberHeight;
678             }
679             catch (PvException)
680             {
681                 invalidsetting = true;
682 
683                 result += 8; // "Capture Image: Cannot set framegrabber
    resolution parameter (height).\r\n"
684             }
685         }
686 
687         if (mPixelFormat != null)
688         {
689             try
690             {
691                 // TODO: Allow different values for PixelFormat
692                 mPixelFormat.ValueInt = DefaultFrameGrabberPixelFormat;
693             }
694             catch (PvException)
695             {
696                 invalidsetting = true;
697 
698                 result += 4; // "Capture Image: Cannot set default framegrabber
    pixel format.\r\n"
699             }
```

```
700             }
701
702             if (mAcquisitionMode != null)
703             {
704                 try
705                 {
706                     // TODO: Allow different values for AcquisitionMode
707                     mAcquisitionMode.ValueInt =
        DefaultFrameGrabberAcquisitionMode;
708                 }
709                 catch (PvException)
710                 {
711                     invalidsetting = true;
712
713                     result += 2; // "Capture Image: Cannot set default framegrabber
        acquisition mode.\r\n"
714                 }
715             }
716
717             if (invalidsetting && mWidth != null && mHeight != null)
718             {
719                 try
720                 {
721                     mWidth.Value = DefaultFrameGrabberWidth;
722                     mHeight.Value = DefaultFrameGrabberHeight;
723                 }
724                 catch (PvException)
725                 {
726                     result += 1; // "Capture Image: Cannot set default framegrabber
        resolution.\r\n"
727                 }
728             }
729         }
730
731         return result;
732     }
733
734     // Setups streaming. After calling this method the application is ready to
        receive data.
735     // StartAcquisition will instruct the device to actually start sending data.
736     private bool StartStreaming()
```

```
737         {
738             if (m_Pipeline != null && m_Pipeline.IsStarted)
739             {
740                 StopStreaming();
741             }
742 
743             // Configure acquisition state manager
744             if (m_Device != null && m_Stream != null)
745                 m_AcquisitionManager = new PvAcquisitionStateManager(m_Device,
        m_Stream);
746 
747             // Start pipeline
748             if (m_Pipeline != null && !m_Pipeline.IsStarted)
749                 m_Pipeline.Start();
750             else if (m_Pipeline != null && m_Pipeline.IsStarted)
751             {
752                 return false;
753             }
754             else if (m_Pipeline == null)
755             {
756                 return false;
757             }
758 
759 #if (PLEORA_API_V4)
760             // Enables streaming before sending the AcquisitionStart command.
761             if (m_Device != null)
762                 m_Device.StreamEnable();
763 #endif
764 
765             return (m_Pipeline.IsStarted);
766         }
767 
768         // Stops streaming. After calling this method the application is no longer
        armed or ready
769         // to receive data.
770         public void StopStreaming()
771         {
772             // Release acquisition manager
773             if (m_AcquisitionManager != null)
774             {
775                 m_AcquisitionManager.Dispose();
```

```
776             m_AcquisitionManager = null;
777         }
778
779 #if (PLEORA_API_V4)
780         // Disable streaming after sending the AcquisitionStop command.
781         if (m_Device != null)
782             m_Device.StreamDisable();
783 #endif
784
785         // Stop the pipeline
786         if (m_Pipeline != null && m_Pipeline.IsStarted)
787         {
788             m_Pipeline.Stop();
789         }
790     }
791
792     // Starts acquisition from the Pleora device.
793     private int StartAcquisition(int frames, bool saveImage, string filename,
        string folder)
794     {
795         int framecounter = frames;
796         int imagecaptured = 0;
797         bool folderexists;
798         string fullpathfilename = String.Empty;
799         string fileextension = String.Empty;
800         string filenamewithoutextension = String.Empty;
801
802         PvBuffer aBuffer = null;
803         PvResult result;
804
805         UInt32 lPayloadSize;
806
807         // Get payload size
808         try
809         {
810             lPayloadSize = PayloadSize;
811         }
812         catch (PvException)
813         {
814             lPayloadSize = 0;
815         }
```

```
816         if (lPayloadSize > 0)
817         {
818            // Propagate to pipeline to make sure buffers are big enough
819            m_Pipeline.BufferSize = lPayloadSize;
820         }
821
822         // Reset pipeline
823         if (m_Pipeline != null)
824            m_Pipeline.Reset();
825
826         // Reset stream statistics
827         PvGenCommand lResetStats = m_Stream.Parameters.GetCommand("
        Reset");
828         lResetStats.Execute();
829
830         try
831         {
832            // Use acquisition manager to send the acquisition start command to
        the device.
833            // It issues the following two commands (exact order):
834            // //m_Device.GenParameters.SetIntegerValue("TLParamsLocked", 1);
835            // //m_Device.GenParameters.ExecuteCommand("AcquisitionStart");
836            if (m_AcquisitionManager != null) // && m_AcquisitionManager.
        State == PvAcquisitionState.Unlocked)
837               m_AcquisitionManager.Start();
838         }
839         catch (PvException)
840         {
841            return -1;
842         }
843
844         folderexists = System.IO.Directory.Exists(folder);
845
846         do
847         {
848            framecounter--;
849
850            // Use a 1 second timeout, DO NOT BLOCK !
851            result = m_Pipeline.RetrieveNextBuffer(ref aBuffer, 1000);
852
853            if (result.IsOK)
```

```
854             {
855                 if (aBuffer != null && aBuffer.OperationResult.IsOK && aBuffer.
        Image.Width > 0 && aBuffer.Image.Height > 0)
856                 {
857                     if (m_BackImage != null)
858                         m_BackImage.Dispose();
859 
860                     m_BackImage = new Bitmap((int)aBuffer.Image.Width, (int)
        aBuffer.Image.Height);
861 
862                     aBuffer.Image.CopyToBitmap(m_BackImage);
863 
864                     imagecaptured++;
865 
866                     if (saveImage)
867                     {
868                         if (folderexists)
869                         {
870                             if (frames == 1)
871                                 fullpathfilename = folder + "\\" + filename;
872                             else
873                             {
874                                 filenamewithoutextension = System.IO.Path.
        GetFileNameWithoutExtension(filename);
875                                 fileextension = System.IO.Path.GetExtension(filename);
876                                 fullpathfilename = folder + "\\" +
        filenamewithoutextension + (frames - framecounter) + fileextension;
877                             }
878                         }
879 
880                         if (folderexists)
881                             m_BackImage.Save(fullpathfilename, System.Drawing.
        Imaging.ImageFormat.Bmp);
882                     }
883 
884                     OnImageChanged(EventArgs.Empty);
885                 }
886 
887                 if (aBuffer != null)
888                     m_Pipeline.ReleaseBuffer(aBuffer);
889             }
```

```
890             } while (framecounter > 0);
891
892             return imagecaptured;
893         }
894
895         // Stops acquisition from the Pleora device.
896         private void StopAcquisition()
897         {
898             try
899             {
900                 // Use acquisition manager to send the acquisition stop command to
        the device.
901                 // It issues the following two commands (exact order):
902                 // //m_Device.GenParameters.ExecuteCommand("AcquisitionStop");
903                 // //m_Device.GenParameters.SetIntegerValue("TLParamsLocked", 0);
904                 if (m_AcquisitionManager != null) // && m_AcquisitionManager.
        State == PvAcquisitionState.Locked)
905                     m_AcquisitionManager.Stop();
906             }
907             catch (PvException)
908             {
909                 return;
910             }
911         }
912
913         // Retrieve or guess the payload size
914         private UInt32 PayloadSize
915         {
916             get
917             {
918                 // Get parameters required
919 #if (PLEORA_API_V3)
920                 PvGenInteger lPayloadSize = m_Device.GenParameters.GetInteger("
        PayloadSize");
921                 PvGenInteger lWidth = m_Device.GenParameters.GetInteger("Width")
        ;
922                 PvGenInteger lHeight = m_Device.GenParameters.GetInteger("Height
        ");
923                 PvGenEnum lPixelFormat = m_Device.GenParameters.GetEnum("
        PixelFormat");
924 #endif
```

```
925 #if (PLEORA_API_V4)
926         PvGenInteger lPayloadSize = m_Device.Parameters.GetInteger("
    PayloadSize");
927         PvGenInteger lWidth = m_Device.Parameters.GetInteger("Width");
928         PvGenInteger lHeight = m_Device.Parameters.GetInteger("Height");
929         PvGenEnum lPixelFormat = m_Device.Parameters.GetEnum("
    PixelFormat");
930 #endif
931
932         // Try getting the payload size from the PayloadSize mandatory
    parameter
933         Int64 lPayloadSizeValue = 0;
934         if (lPayloadSize != null)
935         {
936            try
937            {
938               lPayloadSizeValue = lPayloadSize.Value;
939            }
940            catch (PvException)
941            {
942               return 0;
943            }
944         }
945
946         // Compute poor man's payload size - for devices not maintaining
    PayloadSize properly
947         Int64 lPoorMansPayloadSize = 0;
948         if ((lWidth != null) && (lHeight != null) && (lPixelFormat != null))
949         {
950            Int64 lWidthValue = lWidth.Value;
951            Int64 lHeightValue = lHeight.Value;
952
953            Int64 lPixelFormatValue = lPixelFormat.ValueInt;
954            Int64 lPixelSizeInBits = PvImage.GetPixelBitCount((PvPixelType)
    lPixelFormatValue);
955
956            lPoorMansPayloadSize = (lWidthValue * lHeightValue *
    lPixelSizeInBits) / 8;
957         }
958
959         // Take max
```

```
960             Int64 lBestPayloadSize = Math.Max(lPayloadSizeValue,
        lPoorMansPayloadSize);
961             if ((lBestPayloadSize > 0) && (lBestPayloadSize < UInt32.MaxValue)
        )
962             {
963                 // Round up to make it mod 32 (works around an issue with some
        devices)
964                 if ((lBestPayloadSize % 32) != 0)
965                 {
966                     lBestPayloadSize = ((lBestPayloadSize / 32) + 1) * 32;
967                 }
968
969                 return (UInt32)lBestPayloadSize;
970             }
971
972             // Could not compute/retrieve payload size...
973             return 0;
974         }
975       }
976     }
977   }
```

Bibliografia

[1] Theuwissen A. J. P., *CMOS or CCD image sensors for digital still applications?*, Proceedings of the 25th European Solid-State Circuits Conference (ESSCIRC '99), Pag. 28, 1999.

[2] Theuwissen A. J. P., *CCD or CMOS image sensors for consumer digital still photography?*, Proceedings of Technical Papers from the International Symposium on VLSI Technology, Systems and Applications, Pag. 168-171, 2001, ISBN 0-7803-6412-0.

[3] Carlson B. S., *Comparison of modern CCD and CMOS image sensor technologies and systems for low resolution imaging*, Proceedings of IEEE Sensors, Vol. 1, Pag. 171-176, 2002, ISBN 0-7803-7454-1.

[4] D. Litwiller, *CCD vs. CMOS: Facts and Fiction*, Photonics Spectra, Laurin Publishing Company Inc., January 2001.

[5] Documentazione interna Datalogic, *DE2011-DL Integration Guide* (Revisione A), Maggio 2014, codice a barre 820061590.

[6] ECMA International, *C# Language Specification*, ECMA-334, quarta edizione, Giugno 2006, http://www.ecma-international.org/publications/standards/Ecma-334.htm.

[7] ECMA International, *Common Language Infrastructure (CLI) - Partitions I to VI*, ECMA-335,

sesta edizione, Giugno 2012, http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[8] Herbert Schildt, *C# 4.0: the complete reference*, McGraw-Hill, 2010, ISBN 978-0-07-174116-3.

[9] James Rumbaugh, Ivar Jacobson, Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, seconda edizione, 2005.

[10] Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, seconda edizione, 2005.

[11] Jim Arlow, Ila Neustadt, *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley, seconda edizione, 2005.

[12] Ernani Carrada, *L'affidabilità per l'elettronica*, La Goliardica Editrice, 1975.