

|  |  |
| --- | --- |
|  | **imotion** |
|  |  |
|  | AbuDhabi Project |
|  |  |
|  | Documentation |
|  |  |
|  | Technical Description |
|  |  |
|  | Version: 0.2 |
|  | State: Draft |
|  | Classification: Internal use only |
|  | Author: COS |
|  | Creation date: 2012-08-02 |
|  | Repository: $/Gorba/Main/Motion/Protran/Documents/TD\_ProtranDocumentation/TD\_AbuDhabiProject.docx |
|  | Gorba AG  Sandackerstrasse  9245 Oberbüren  Switzerland |

**Table of contents**

[1 Introduction 4](#_Toc335049403)

[1.1 Protran Version 4](#_Toc335049404)

[1.2 Purpose 4](#_Toc335049405)

[1.3 Intended Audience and Reading Suggestions 4](#_Toc335049406)

[1.4 Remarks 4](#_Toc335049407)

[2 System Overview 4](#_Toc335049408)

[3 Startup Sequence 5](#_Toc335049409)

[3.1 AbuDhabi DLL Startup 6](#_Toc335049410)

[3.1.1 ISI Communication 6](#_Toc335049411)

[3.1.2 FTP Communication 7](#_Toc335049412)

[3.1.3 CTU Communication 7](#_Toc335049413)

[3.2 Ibis DLL Startup 7](#_Toc335049414)

[4 IBIS and ISI Protocols Integration 8](#_Toc335049415)

[4.1 The Fallback Algorithm 9](#_Toc335049416)

[4.2 XIMPLE Cache 10](#_Toc335049417)

[5 AbuDhabi State Machines 11](#_Toc335049418)

[5.1 AbuDhabi Cycles 11](#_Toc335049419)

[5.2 ISI State Machine 14](#_Toc335049420)

[5.3 IBIS State Machine 14](#_Toc335049421)

[5.4 State Machine Architecture 15](#_Toc335049422)

[6 Update 16](#_Toc335049423)

[6.1 TopBox Update 16](#_Toc335049424)

[6.2 CU5 Update 20](#_Toc335049425)

**Modification management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Modifications** | **State** |
| 0.1 | 2012-08-02 | COS | SW | Initial version | draft |
| 0.2 | 2012-09-10 | COS | SW | First review | draft |

**Review**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Remarks** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Release**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Remarks** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Introduction

## Protran Version

This document covers the Protran’s features starting from its version **1.4.1231** and implemented for the AbuDhabi project.

## Purpose

The goal to this document is to describe the Protran’s behaviors and architecture for the AbuDhabi project.

At the end of this document, the reader will have all the details to understand how Protran works and how perform its tasks for the AbuDhabi project. .

## Intended Audience and Reading Suggestions

This document is addressed to software developers due to its technical contents.

## Remarks

This document doesn’t cover the Protran’s configuration. To have notion about it, please read the document: $/Gorba/Main/Motion/Protran/Documents/TD\_ProtranDocumentation/TD\_ProtranDocumentation.docx

# System Overview

Firstly will be given a brief overview about the system used for the AbuDhabi project:

INIT Board Computer

Gorba’s TopBox

CU5

TCP/IP

TCP/IP

UDP

IBIS

Figure ‎2‑1 AbuDhabi System Overview

The TopBox is connected to two external entities and the transports used are Ethernet or serial ports .

The system is composed by several applications. All of them are shown in the following scheme:

* Copilot Simulator
* INIT FTP server
* Protran.exe
* MediServer.exe
* InfoMedia.exe
* FileZilla Server.exe
* Update.exe
* SystemManager.exe
* uTasker
* FTP Client

CU5

INIT Board Computer

Gorba’s TopBox

Figure ‎2‑2 AbuDhabi’s Applications Set

In the INIT Board Computer, the *main* application is “Copilot Simulator”: this is the application tasked to send to Protran all the information for the passengers and so on. Also, the INIT Board Computer contains an FTP server from which Protran downloads the files required to update the CU5 (the process about this update will be explained later) and the TopBox itself (for example the InfoMedia’s layout, newer software etc…).

In the Gorba’s TopBox there is our suite of applications all connected to each other. Basically, Protran is the application tasked to gather from the sources the information, the application tasked to display them on the screen is InfoMedia and MediServer is the application responsible to connect Protran to InfoMedia and viceversa.

In the CU5 device, the *main* “application” is uTasker: it sends information for the passengers using the IBIS protocol. There’s also an FTP client tasked to download files from the TopBox only after some command over UDP ordered by Protran.

In the next chapters will be explained the most important parts of the AbuDhabi project. They are:

* the startup sequence
* the integration with the IBIS protocol
* the fallback mechanism
* the cycle state machines
* the update procedures

For the rest of this document it will be intended that Protran is configured to support the AbuDhabi project.

# Startup Sequence

Currently the Protran’s solution is made by the projects shown in the following picture:

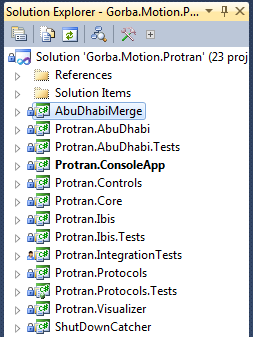


Figure ‎3‑1 Protran’s Solution

At the Protran’s startup, the **ConsoleApp** loads the DLL **Protran.AbuDhabi**; this last loads the DLL **Protran.Ibis** (in a separate thread). The first DLL is responsible to manage completely all the tasks with the INIT board computer, and the second one manages all the stuffs related to the IBIS protocol (this kind of telegrams are coming from the CU5).

## AbuDhabi DLL Startup

In this DLL is entirely contained the logic about the communication (bidirectional) with the INIT remote computer and the logic about the communication (also this is bidirectional) with the CU5. The first type of communication follows the rules specified by the **ISI** protocol (TCP/IP); the second follows the rules of the **CTU** protocol (UDP). In this DLL there’s also the logic about the process of downloading files from the INIT board computer using the **FTP** protocol.

Before starting the **ISI** communication, Protran tries to contact the CU5 sending to it a **CTU** datagram and waiting for a response. Once the answer is received, Protran starts the **ISI** communication. If the CU5 is not reachable (due to problems with the cables, IP address not rightly configured, etc…) after a certain time expires a timeout and after this event Protran starts anyway the **ISI** communication. This behavior is to avoid a paralysis in the communication.

From the point of view of the source code, the protocols mentioned above are managed by the following objects:

|  |  |
| --- | --- |
| **Protocol** | **Object manager** |
| ISI | Gorba.Motion.Protran.AbuDhabi.Isi.IsiClient |
| FTP | Gorba.Motion.Protran.AbuDhabi.Ism.IsmClient |
| CTU | Gorba.Motion.Protran.AbuDhabi.Ctu.CtuClient |

Table ‎3‑1 AbuDhabi DLL Protocols

In the next paragraphs will be described the behaviors of all of them.

### ISI Communication

The INIT board computer expects to receive from Protran (periodically and when a change occurs) the following set of information:

* The Protran’s application name (hard coded to “INFOTAINMENT”)
* The Protran’s serial number (hard coded to “9,1234”)
* The CU5 status
* The CU5 serial number
* The CU5 software version
* The CU5 data version
* The Protran’s status about its FTP activity

Protran expects to receive from the INIT board computer (periodically and when a change occurs) information mainly about the trip that has to be played by InfoMedia.exe both in English and in Arabic language (the set of this information is huge and therefore is here omitted. It can be found in the file AbuDhabi.xml).

As the name IsiClient suggests, Protran is not an ISI based TCP/IP server but rather a client. In this are client obviously implemented the reconnection and the keep alive mechanisms.

An extremely important function within IsiClient is:

private void RaiseXimpleCreated(XimpleEventArgs e)

It’s invoked whenever the client has produced a XIMPLE ready to be forwarded to the Protran’s core. The typical information encapsulated in those XIMPLEs are referred to route currently performed by the bus.

In order to receive all the XIMPLEs produced by IsiClient, you have to register to its event public event EventHandler<XimpleEventArgs> XimpleCreated.

### FTP Communication

As mentioned before, in the INIT board computer is installed an FTP server to which Protran has to connect and download files, whenever possible. This point is extremely important for the AbuDhabi project because involves the update of the TopBox and the CU5 software and/or the data and therefore will be detailed later in a separate chapter.

### CTU Communication

CTU is the protocol used by Protran and the CU5, is UDP based and is required to forward some information from the INIT board computer to the CU5 and to rule the CU5 update process.

During its startup, Protran sends periodically CTU telegrams to the CU5 having information about its status and expects to receive periodically the CU5 status as well. If there’s silence from the CU5 for too much time, Protran restarts its UDP client and then retries. In case of active communication, Protran sends to the CU5 the following information:

* The line number
* The direction number
* The destination number
* The destination text in English language
* The destination text in Arabic language

Those information are required by the CU5 for its internal purposes and Protran acts as a pure forwarder.

This object doesn’t produce any XIMPLE, in fact in it is absent the function

private void RaiseXimpleCreated(XimpleEventArgs e)

and the EventHandler

public event EventHandler<XimpleEventArgs> XimpleCreated.

## Ibis DLL Startup

For the AbuDhabi project is important to recognize also all the telegrams coming from the IBIS serial port. Usually, for the projects where is required to gather data only from the serial port, the **Protran.Ibis.dll** is of course the only one loaded, but for the AbuDhabi project is mandatory the coexistence with **Protran.AbuDhabi.dll**.

During the “boot” of **Protran.Ibis.dll**, are initialized and opened the two serial ports COM1 and COM2. The first is the one from which are sent/received the IBIS telegrams, the second is used just only to detect changes on the CTS and RTS lines.

|  |  |
| --- | --- |
| **COM1** | **COM2** |
| * Read IBIS telegrams * Write IBIS telegrams | * Detect changes on the CTS line * Detect changes on the RTS line |

Table ‎3‑2 Protran’s Serial Ports

The other thing done by Protran during the startup of this DLL, is the configuration and initialization of the IBIS recorder (an object tasked to store in a file all the IBIS telegrams received in order to simulate later the session in case of debug).

Inside this DLL there are sophisticated logics about the parsing and the handling of the IBIS telegrams (points that will not be covered in this document) but here is important to mention that all the XIMPLEs produced in this DLL will arrive to the object Gorba.Motion.Protran.Ibis.IbisProtocol in its function

private void RaiseXimpleCreated(XimpleEventArgs e)

Gorba.Motion.Protran.Ibis.IbisProtocol is the main object in the **Protran.Ibis.dll**. Its tasks are:

* To create and initialize the channels for the IBIS telegram (here, mainly the creation and the initialization of the serial port connected to the IBIS master)
* To create and initialize the handlers of all the IBIS telegrams that must be managed (parsed, transformed and translated in XIMPLEs)
* To start the process about the synchronization of the system’s time
* To manage all the XIMPLEs coming from other applications connected to MediServer
* To forward to the Protran’s core all the XIMPLEs produced by the IBIS handlers

An UML schema whit the IbisProtocol’s skeleton is provided with the following image:

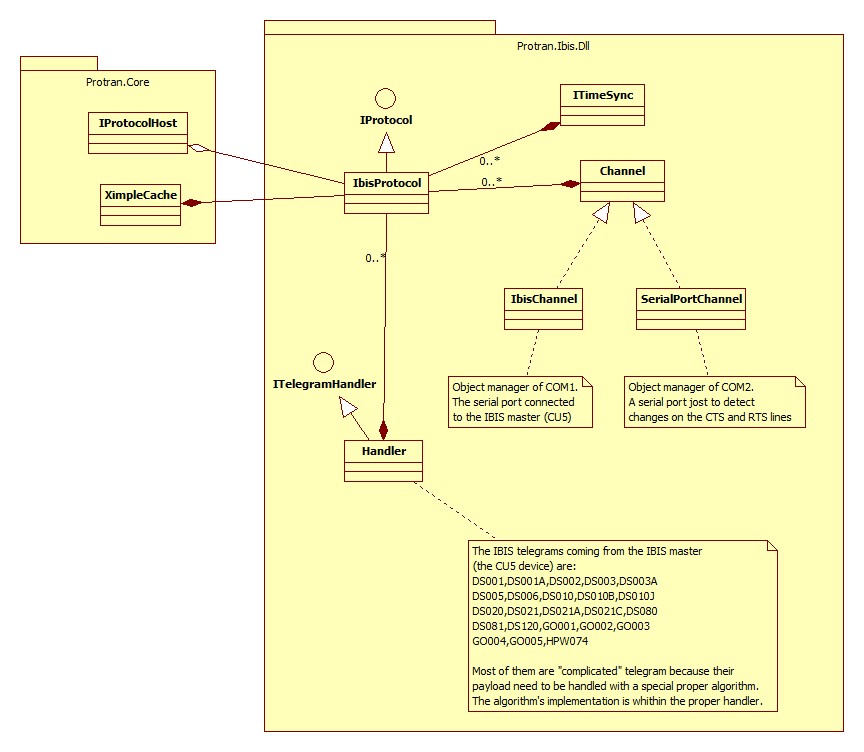


Figure ‎3‑2 IbisProtocol’s overview

After this presentation about the logics distributed in the Protran.AbuDhabi.dll and Protran.Ibis.dll is now the time to see in more details how they are interrelated.

# IBIS and ISI Protocols Integration

Protran can accept information from two sources:

INIT board computer

Protran

CU5

Screen

ISI protocol

IBIS protocol

XIMPLE

Figure ‎4‑1 Protocols carrying passengers information

Obviously, not both of them can be *selected* at the same time because they might send different information causing confusion to the passengers. This is the reason why Protran has to implement an algorithm for the source selection. This algorithm is explained in the next paragraph.

## The Fallback Algorithm

By default, Protran starts selecting the ISI source and switches automatically to the IBIS one if the first doesn’t send anything within a period of two minutes (more precisely 125 seconds).

Also, Protran switches to the IBIS source (and then back again to ISI source) whenever the INIT board computer sends to it the specific **ISI** command **GorbaSystemFallbackActive** with a specific value. The logic is shown with the flow chart below:

Select the **ISI** source

Wait for ISI information

Received ISI information?

Waited too much time?

Received **GorbaSystemFallbackActive** with value 1?

Select the **IBIS** source

Reception of **GorbaSystemFallbackActive** with value 0

yes

no

no

yes

yes

no

Do stuffs with the **ISI** information just received

Figure ‎4‑2 Fallback Algorithm

The switch policy is done completely by the object:

Gorba.Motion.Protran.AbuDhabi.Multiplexing.XimpleArbiter

that resides (as its namespace suggest) in the AbuDhabi DLL.

The only callable function to this object and having the capability to switch the source is the property:

public IXimpleSource CurrentSource

As you can see, a source is recognized to be valid if it implements the interface IXimpleSource. This decouples completely the source switch policy from the source type itself.

XimpleArbiter has two instances of sources:

private readonly IXimpleSource primary;

private readonly IXimpleSource secondary;

The first is the ISI source and the second is the IBIS one.

Only one source is set to be currently the active and only the XIMPLEs produced by it are forwarded to the player. It’s extremely important to understand that even if a source is not currently selected, this doesn’t mean that it is ignored by Protran. Protran does the parsing, the handling and the translations also to the information coming from the not selected source **but** blocks all the referring produced XIMPLEs into a cache. Those cells are blocked until their source becomes the currently active, and once this occurs they are forwarded immediately to the player (clearing in this way the cache and refreshing the player with the new information).

## XIMPLE Cache

The object Gorba.Motion.Protran.Core.Utils.XimpleCache is a pure container of XIMPLE cells (attention, not XIMPLEs but cells). It is used by the DLL **Protran.AbuDhabi** and **Protran.Ibis.**

A XIMPLE’s cell is uniquely identified by the 3D-coordinate {table, column, row} and that’s why the XIMPLE cache uses a .NET Dictionary instantiated as follows to store them:

private readonly Dictionary<string, Cell> cells = new Dictionary<string, Cell>();

where a dictionary key is composed by:

string key = string.Format("{0}#{1}#{2}", cell.Table, cell.Column, cell.Row);

Sometimes, internally in Protran are produced XIMPLEs having just only one cell. These kinds of XIMPLEs have to be passed to MediServer as well even if their content is extremely small. Also, it may happen that cells with the same coordinates and values have to be sent with different XIMPLEs after really short intervals of time. In order to avoid a “waste” of XIMPLEs (and therefore TCP/IP bandwidth) a cache is required.

For the AbuDhabi project, the cache is extremely important during the switch between a source and the other (fallback mechanism). For example, during the “ISI mode” all the XIMPLEs received from the IBIS source (the CU5 device) are stored into the cache (instead of being discarded) and whenever the source changes from “ISI mode” to “IBIS mode” all the cache’s content is sent through MediServer.

# AbuDhabi State Machines

As mentioned in the previous chapters, for the AbuDhabi project are available two sources that send to Protran information for the passengers. This information (after some manipulations) are displayed on the screen by the application InfoMedia. Basically, the information to be shown are a set of stops (usually 5 or 6 stops one above the other), the destination, a text or a picture that say that a passenger has requested the stop, a text or a picture that say that the doors are open/close and so on. According to the functional specifications defined for the AbuDhabi project, depending on the information received from the source, on the display must be shown a specific **set of information**. We can call “**cycle**[[1]](#footnote-1)” each specific set of information. For example, during a normal journey is continuously shown the stops list and the journey’s destination. 100 m before the arrival to the next stop, Protran receives from the source the information about the approach to the next stop this kind of information: the result has to be a cycle change from the one previously shown to the new one depicted for this specific state. And obviously, when the bus stop is over, on the screen must be shown the initial cycle with the updated stops list. The way in which Protran “pilots” the InfoMedia’s cycles is realizing a state machine.

## AbuDhabi Cycles

The cycles defined for the AbuDhabi project are 4:

* Main cycle
* Stop requested cycle
* Stop approaching cycle
* Stop requested AND approaching cycle

The first, “Main cycle” shows the so called “Perlschnur” on the screen:

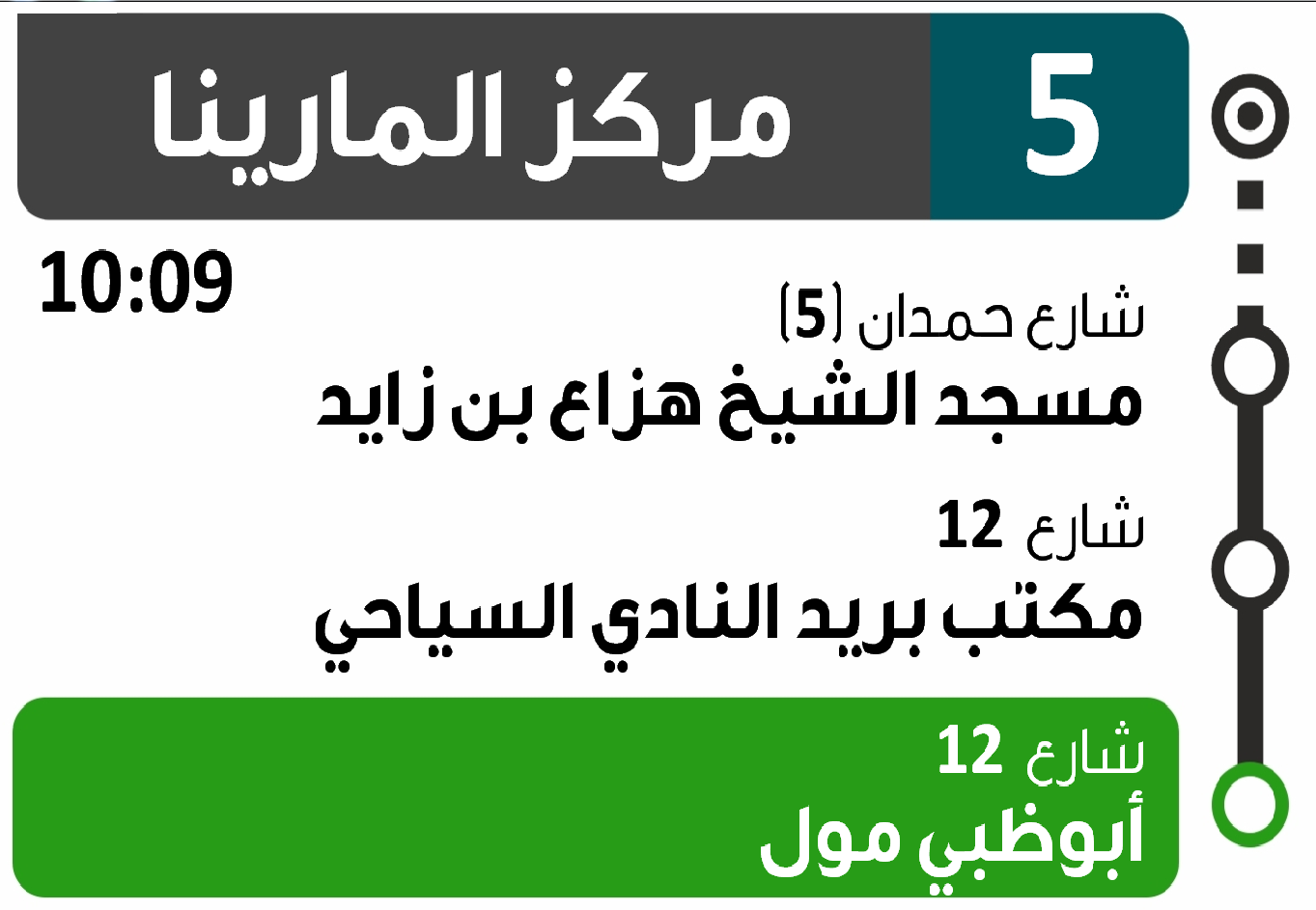


Figure ‎5‑1 the typical Main Cycle

The second cycle, “Stop requested” is as shown below:



Figure ‎5‑2 the typical Stop requested cycle

The third cycle, “Stop approaching” is as shown below:

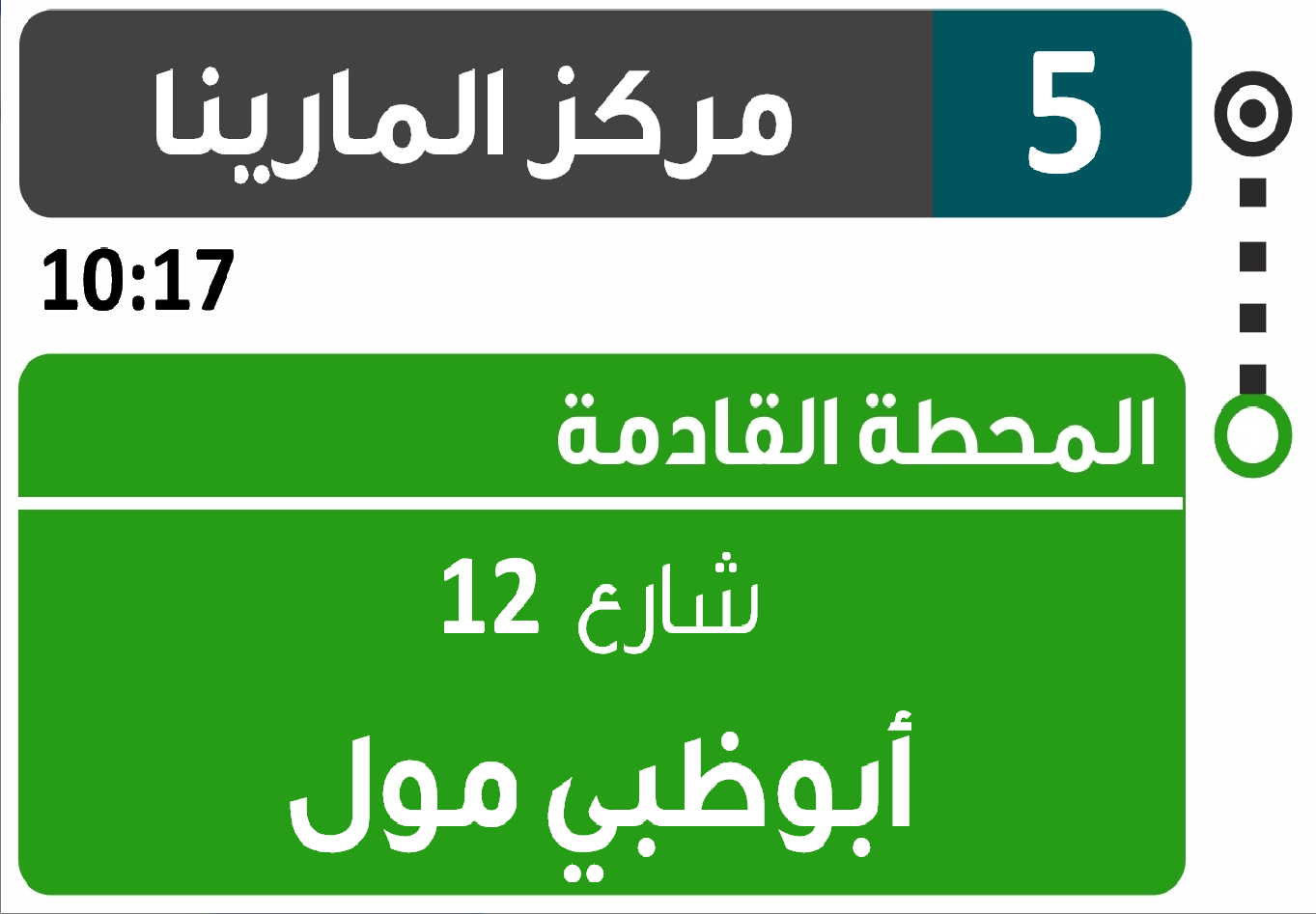


Figure ‎5‑3 the typical Stop approaching cycle

The forth cycle, “Stop requested AND approaching” is as shown below:



Figure ‎5‑4 the typical Stop requested and approaching cycle

To switch from a cycle to a different one is enough to send some specific XIMPLE cells to InfoMedia.exe (see Dictionary.xml and InfoMedia.xml). The object tasked to understand which cell has to be sent, depending on the information received from the source is

Gorba.Motion.Protran.AbuDhabi.StateMachineCycles.CycleManager

Depending on which is the current selected source, the CycleManager receives different information and therefore are also different the triggers to switch from a state to the new one. The state machines implemented for the two sources, IBIS and ISI, will be explained in the next paragraphs.

## ISI State Machine

The state machine implemented for the ISI protocol is shown on the figure below:



Figure ‎5‑5 State Machine for ISI Protocol

As you can see, are present the four states mentioned before and the triggers that switch from a state to another. When the system is in the “Main cycle” it can go to the “Stop requested” cycle whenever Protran detects a level “1” to the CTS line of the serial port COM2 and can go to the “Stop approaching” cycle whenever the ISI remote computer sends to Protran the ISI item **IsVehicle100mBeforeStopOrAtStop** with value 1. For all the other events, the system stays on the “Main cycle”.

Once the system is in the “Stop requested” cycle, it can go to the “Stop requested AND approaching” state whenever the ISI remote computer sends to Protran the ISI item **IsVehicle100mBeforeStopOrAtStop** with value 1, and can go back to the “Main cycle” whenever Protran detects a level “0” to the CTS line of the serial port COM2.

If the system is in the “Stop approaching” cycle, it can go to the “Stop requested AND approaching” cycle whenever Protran detect a level “1” to the CTS line of the serial port COM2 and can go to the “Main cycle” if the ISI remote computer sends to Protran the ISI item **IsVehicle100mBeforeStopOrAtStop** with value 0 or the ISI remote computer updates the stops list.

If the system is in the “Stop requested AND approaching” state it can go to the “Stop requested” state if the ISI remote computer sends to Protran the ISI item **IsVehicle100mBeforeStopOrAtStop** with value 0 or can go to the “Main cycle” if Protran detects a level “0” to the CTS line of the serial port COM2.

## IBIS State Machine

The state machine implemented for the IBIS protocol is shown on the figure below:



Figure ‎5‑6 State Machine for IBIS Protocol

The way in which Protran performs the switches between the states is exactly the same as the one explained for the ISI state machine, but are different only the triggers.

Basically, the **IsVehicle100mBeforeStopOrAtStop**’s equivalent for the IBIS protocol is the telegram „xE“: this telegram contains the value “0” or “1” that informs Protran that the bus is approaching to the next stop or if the stop is over. The update of the stops list is determined by the telegram “xI”: this telegram contains the index of the next stop and once Protran receives it, it has to update the “Perlschnur” on the screen.

Keeping in mind these equivalences, the two state machines behave completely in the same way.

In the next paragraph it will be given an overview of the state machine with UML diagrams.

## State Machine Architecture

The state machines (and their users) are implemented by the set of classes shown in the figure below:

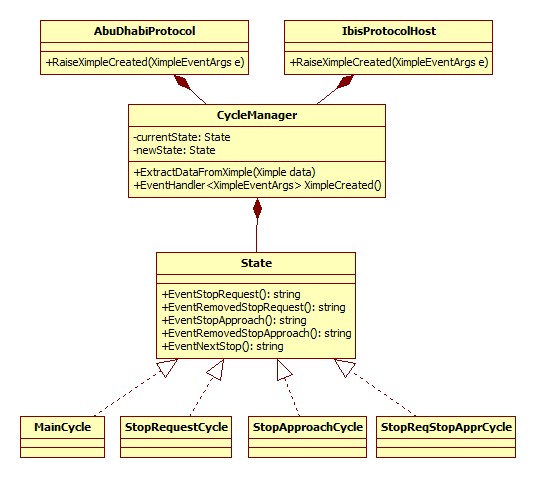


Figure ‎5‑7 State Machine Architecture

Basically, the objects Gorba.Motion.Protran.AbuDhabi.AbuDhabiProtocol and Gorba.Motion.Protran.AbuDhabi.Ibis.IbisProtocolHost pass all their XIMPLEs to the proper instance of Gorba.Motion.Protran.AbuDhabi.StateMachineCycles.CycleManager and obtain new XIMPLEs in their previously registered function

private void RaiseXimpleCreated(XimpleEventArgs e)  
to the event

public event EventHandler<XimpleEventArgs> XimpleCreated;  
A **trigger** “activates” the state machine passing XIMPLEs to it using the function

public void ExtractDatafromXimple(Ximple data)  
and the outcomes “exit” from the state machine with the event XimpleCreated.

The “calculation” of the next state is completely demanded to the specific state implementation and they follow the rules explained in the previous paragraphs.

# Update

For the AbuDhabi project is essential the **update** feature. Two are the devices to be updated in the system:

* The TopBox
* The CU5

For the first device, it is possible to update all its software and also all its data files (for example the various XML files used for configurations, images, videos, etc…), and the same also for the second.

Due to the completely different targets, the update processes will be detailed in two different paragraphs.

## TopBox Update

For this kind of update process, the entities involved are:

* The ISI board computer with its applications COPILOT Simulator **and** MiniFTP
* Protran

With the cooperation of that set of three applications, it’s possible to update the TopBox and to provide to Protran the files to update the CU5.

Basically, Protran checks for the existence of a specific FTP server installed in the INIT board computer, and once it detects it, checks for the existence of specific files (only files with specific extensions will be downloaded). After the download, the connection is closed and will be re-opened later in polling in order to check always if there’s something new to download. It’s not enough the only existence of the FTP server in the INIT board computer because the update process is “synchronized” using the ISI item **IsiClientRunsFtpTransfers**: whenever Protran starts the FTP download, it sends to COPILOT Simulator the **IsiClientRunsFtpTransfers** with the value 1, and once the download is finished it sends **IsiClientRunsFtpTransfers** with the value 0 (you can assume that the combination of FTP and ISI makes the **ISM** protocol).

The update process is explained with the following sequence diagrams:

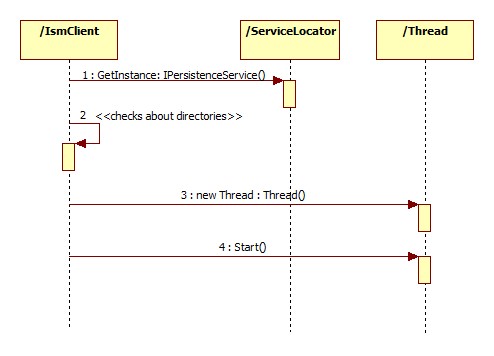


Figure ‎6‑1 Update TopBox Start Procedure

The Gorba.Motion.Protran.AbuDhabi.IsmClient firstly, asks for an *IPersistenceService* in order to have a service that “remembers” what Protran downloads (this avoids the eventual unuseful downloads of files already downloaded), secondly does some checks about the existence of the directories in which will be stored the new files and finally starts a separate thread giving to it the task to connect to the FTP server. This last task (the FTP connection) is explained with the following sequence diagram:

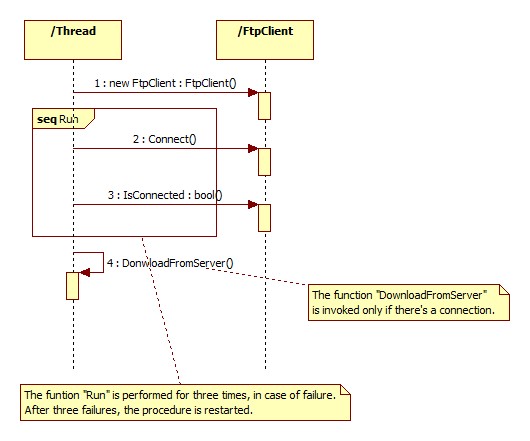


Figure ‎6‑2 Connection Procedure

Firstly, the thread tasked to download instantiate an Gorba.Common.Ftp.FtpClient and then tries for three times to establish a connection with the FTP server. In case of three failures, the procedure is restarted (with a new FtpClient instance, and again other three attempts). In case of connection succeeded, starts the real download (summarized in the following scheme):

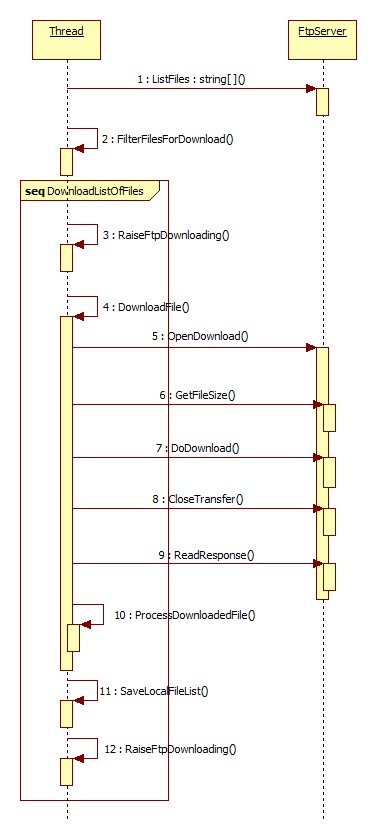


Figure ‎6‑3 Dialogs with the FTP server

Firstly, the thread asks to the FTP server the list of the files ready to be downloaded; after that it filters them to keep only the ones with the valid extensions (configurables) and then, one by one, is performed the real file transfer process. This process consists of notifying the ISI remote computer about the FTP activity, than with the instantiation of a channel with the FTP server, with the retrieval of the file’s size, the file’s download, the save of the file just downloaded in the correct destination directory and finally notifying the ISI remote computer that the download is finished.

It’s important to mention that the files downloaded are **ZIP archives**: Protran has also to extract them before really storing them in the destination directories and this is done within the function “ProcessDownloadedFile”.

## CU5 Update

For this kind of update process, the entities involved are:

* The TopBox with Protran and FTP File Zilla Server
* The CU5

With the cooperation of that set of three applications, it’s possible to update the CU5.

Between Protran and the CU5 there’s no connection, but rather only an exchange of UDP datagrams. These UDP datagram are based on the CTU protocol, and the information contained on theme are basically:

* The “Start to download”, ordered by Protran to the CU5
* The “Abort download”, ordered by Protran to the CU5
* A status request, ordered by Protran to the CU5 about the currently active download process

You can assume that Protran is the master of the whole update process because it decides when to start, when to stop and when to ask for status information.

The description of the whole update process is completely detailed in the document that specifies the CTU protocol. You can find it at this path:

[http://tfsgorba/sites/teamsoftware/Shared Documents/Products/Motion/CUx/Documentation/TD\_CTU\_Protocol\_Specification.docx](http://tfsgorba/sites/teamsoftware/Shared%20Documents/Products/Motion/CUx/Documentation/TD_CTU_Protocol_Specification.docx)

Here below is described in a sequence diagram only how to start the update:

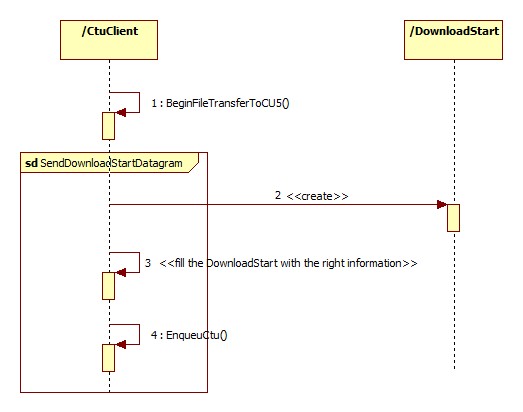


Figure ‎6‑4 Start Update CU5

1. “Cycle”, for historical reasons [↑](#footnote-ref-1)