

Emulation tool for credit card interface validation and authorization

HENRIK PIERROU

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LECS-2005-102

Emulation tool for credit card interface validation and authorization

H E N R I K P I E R R O U

Industry Advisor:
Nazir Goulamhousen, Amadeus SAS, Nice France

Thesis Examiner:
Prof. Vladamir Vlassov, Department of Microelectronics
and Information Technology, Royal Institute of
Technology

Master of Science Thesis
Stockholm, Sweden 2005
IMIT/LECS-2005-102

Abstract

Global Distribution Systems (GDS) provide users around the world access to traveling products such as tickets, hotel room reservations and car rental reservations. When an order is issued, these systems create records describing among other things the form of payment. Payment by credit card is usually an option and whenever this option is used, the GDS needs to send and receive one or more credit card messages to and from the appropriate credit card authorization institution. The responses inform of whether the payments are granted or not. If it is not granted, the reason for denial and/or a handling message is included.

When developing software used to send and receive such messages and when troubleshooting reported errors associated to them it is inevitable to send test credit card messages. Most credit card companies provide test links for these purposes but the inability to control what is received at the credit card company's end, how the message is handled and what is returned limits their usefulness. Also, the test links may sometimes be unavailable due to reasons outside the control of the GDS. A solution for emulating credit card interface validation and authorization, allowing control over the whole process from the sending of the request to the receiving of the response would help facilitate the support and the development in these situations.

This thesis addresses the issue of how such a system could be developed for a GDS company and describes the problems encountered and conclusions drawn from the work conducted in this area at GDS software company Amadeus SAS in Nice, France.

Sammanfattning

Globala distributionssystem (GDS) erbjuder användare i hela världen tillgång till reseprodukter såsom flygbiljetter, hotellrumsbokningar och bilhyrningreservationer. När en order placeras lagras dessa system information om ordern, bland annat vilket betalningssätt som användaren har valt. Betalning med kreditkort är vanligtvis ett alternativ och när detta är valt behöver GDS-systemet skicka och ta emot ett eller flera kreditkortsmeddelanden till och från det korrekta kreditkortsinstitutet. Svaren från kreditkortsinstitutionerna innehåller information om huruvida betalningarna är godkända eller inte. Om de inte är godkända inkluderas även information om varför den inte är godkänd.

Vid utveckling av mjukvara som skickar och tar emot sådana meddelanden och vid felsökning och support av dessa system är det oundvikligt att skicka testkreditkortsmeddelanden. De flesta kreditkortsföretag erbjuder testlänkar för detta ändamål men eftersom användaren av länkarna saknar kontroll över vad som tas emot på kreditkortsföretagets sida, hur meddelandet hanteras och vad som returneras, är dessas användbarhet begränsad. Dessutom är testlänkarna ibland otillgängliga av anledningar utanför GDS-systemens kontroll. En mjukvara som emulerar kreditkortsföretagets beteende och tillåter kontroll över hela processen från skickandet av det första meddelandet till mottagandet av svaret skulle underlätta utvecklings- och supportprocesserna i dessa situationer.

Den här uppsatsen behandlar frågan om hur ett sådant system skulle kunna utvecklas för ett GDS-företag och beskriver arbetet med att utveckla en prototyp på GDS-företaget Amadeus i Nice, Frankrike.

Acknowledgements

I would like to thank **Nazir Goulamhousen** and **Isabella Capella** at Amadeus for introducing me to the Amadeus world and for their constant willingness to answer my questions thoroughly.

I would also like to thank my KTH supervisor, **prof. Vladimir Vlassov** at the Department of Microelectronics and Information Technology, Royal Institute of Technology, Kista, Stockholm for his support and guidance.

Thank you!

Table of Contents

1	INTRODUCTION.....	1
1.1	PROBLEM AREA	1
1.2	COMPANY PRESENTATION	2
1.2.1	<i>GDS</i>	2
1.2.2	<i>E-travel</i>	2
1.2.3	<i>IT-Services</i>	3
1.3	PROBLEM DEFINITION.....	3
1.4	REPORT STRUCTURE.....	4
2	BACKGROUND.....	6
2.1	CREDIT CARD MESSAGES.....	6
2.1.1	<i>The Qantas IGW link – AS2805</i>	6
2.1.1.1	Header	7
2.1.1.2	Message Type ID	7
2.1.1.3	Bitmaps	8
2.1.1.4	Data Fields.....	8
2.1.1.5	Response codes.....	8
2.1.1.6	Message example	9
2.1.2	<i>American Express Link – ISO8583</i>	10
2.1.2.1	Message Type ID	10
2.1.2.2	Bitmaps	11
2.1.2.3	Data Fields.....	11
2.1.2.4	Response Codes.....	11
2.1.2.5	Message Example.....	12
2.1.3	<i>VISA Link – ISO8583</i>	12
2.1.3.1	Header	13
2.1.3.2	Message Type ID	13
2.1.3.3	Bitmaps	14
2.1.3.4	Data Fields.....	14
2.1.3.5	Response Codes.....	14
2.1.3.6	Message Example.....	15
2.1.4	<i>Other Links</i>	16
2.2	CREDIT CARD NUMBER VALIDATION	16
2.3	TTSERVER	17
2.3.1	<i>Receptor</i>	18
2.3.2	<i>Injector</i>	18
2.3.3	<i>Router</i>	18
2.3.4	<i>Dynamic responses</i>	19
2.4	EDIFACT.....	19
2.4.1	<i>Character set</i>	20
2.4.2	<i>Structure</i>	20
2.4.3	<i>HSFREQ/HSFRES</i>	24
2.5	PREVIOUS WORK	25
2.6	REQUIREMENTS.....	26
2.7	EXPECTATIONS.....	26
3	SOLUTION PROPOSAL ANALYSIS.....	27
3.1	GENERAL	27
3.2	IMPLEMENTATION OF EXISTING SOLUTIONS FOR TTSERVER	27
3.2.1	<i>Description</i>	27
3.2.2	<i>Benefits</i>	27
3.2.3	<i>Drawbacks</i>	27

3.3	CREATION OF NEW EDIFACT MESSAGE	28
3.3.1	<i>Description</i>	28
3.3.2	<i>Benefits</i>	28
3.3.3	<i>Drawbacks</i>	28
3.4	PYTHON SCRIPT BASED SOLUTION	29
3.4.1	<i>Description</i>	29
3.4.2	<i>Benefits</i>	29
3.4.3	<i>Drawbacks</i>	29
3.5	CONCLUSIONS	29
4	PROTOTYPE DESIGN	30
4.1	OVERVIEW	30
4.2	MESSAGE STANDARD DESCRIPTION FILES	31
4.2.1	<i>standard</i>	32
4.2.2	<i>field</i>	32
4.2.3	<i>size</i>	33
4.2.4	<i>compression</i>	33
4.2.5	<i>responseAction</i>	34
4.2.5.1	<i>action</i>	34
4.2.5.2	<i>value</i>	34
4.2.5.3	<i>respActionArgs</i>	35
4.2.6	<i>description</i>	35
4.2.7	<i>Example</i>	35
4.3	RESPONSE MESSAGE CREATION	36
4.3.1	<i>Response Action Functions</i>	36
4.3.2	<i>Credit Card - Response Mapping File</i>	37
4.3.3	<i>Credit Card Number Generator</i>	38
4.4	DESIGN PHASE TIME ALLOCATION	39
5	PROTOTYPE USAGE	41
5.1	INTRODUCTION	41
5.2	TYPICAL USAGE	41
5.3	DOWNLOADING	41
5.4	RUNNING	45
5.5	SENDING AND RECEIVING MESSAGES	49
5.6	SPECIFYING RESPONSES	52
5.6.1	<i>General</i>	52
5.6.2	<i>Using the Credit Card – Response Mapping File</i>	53
5.6.3	<i>Generating a credit card account number</i>	54
5.7	UPDATING/ADDING LINKS	54
5.8	MODIFYING RESPONSE FIELD CREATION	55
5.8.1	<i>General</i>	55
5.8.2	<i>Specific value or Echo</i>	55
5.8.3	<i>Response Action Functions</i>	56
6	ANALYSIS	61
6.1	MEASUREMENTS	61
6.2	RESULTS	61
6.2.1	<i>Performance</i>	61
6.2.2	<i>Dynamicity</i>	62
6.2.3	<i>Generality</i>	63
6.2.4	<i>Reliability</i>	64
7	CONCLUSIONS	66
8	FUTURE WORK	68

8.1	HANDLING OF TRUE BINARY DATA	68
8.2	TESTING AND MODIFICATION OF VISA LINK	68
9	REFERENCES	69
APPENDIX A - MESSAGE STANDARD SPECIFICATIONS.....		71
A.1	QANTAS AS2805.....	71
A.2	AMERICAN EXPRESS ISO8583	74
A.3	VISA ISO8583	77
APPENDIX B – ACRONYMS AND ABBREVIATIONS		81

1 Introduction

1.1 Problem Area

Software testing, the process of determining whether software behaves as specified or not, as well as detection and correction of reported bugs, are two important parts of most professional development processes. This requires the possibility to recreate the software's environment. Whittaker [19] states what might seem obvious; "to plan and execute tests, software testers must consider ... the environment in which the software will eventually operate", and continues to propose a four phase model for structuring the work of testing, in which modeling of the software's environment is the first phase. It is, in other words, desirable to have the software run in an environment that to as large a degree as possible corresponds to the one in which it is to be launched or, in the case of troubleshooting, the one in which the error was encountered. Therefore, to be able to correctly test software, all resources with which it may communicate need to be either in connection with the tested software or to be emulated.

Companies integrating credit card payment in their products regularly send credit card authorization request messages to the appropriate credit card issuers to determine whether the transaction associated with the request is granted by the issuer or not. This feature, like all other parts of the software, needs to be included in the testing routines and is by necessity also subject to troubleshooting.

To enable the credit card authorization communication needed in these test and troubleshooting scenarios, most credit card companies provide test links over which the tested application can send test credit card messages. The messages sent over the test links are treated the same way at the credit card company side as a real message would be and the response is created according to the information stored about the request credit card number and the logic built in to the receiving application. It differs only in that no real money transaction is being made.

Also included in the services provided by most credit card companies is the ability to predict to some extent what the responses created by the credit card company as a result of the request messages are going to be. This is made possible by a collection of un-issued credit card numbers which are mapped to certain responses in the test systems. By using one of these known credit card numbers the requesting side can expect to receive the error message or approval code that is mapped to the number.

The test links provided by the credit card companies are good ways for developers of applications integrating credit card payment to test the applications in a realistic way. They provide a means to simulate rather than emulate the behavior of the external resource which satisfies the objective of running the application in an environment as similar to the real one as possible.

They do however have drawbacks of which the main one is the lack of control over what is actually being done in the simulation. In order to be able to draw correct conclusions about the results of a test it is necessary to know why the result was generated. When the simulation is being made at the credit card company end the tester can not be sure that the results received from the simulation are the correct ones and hence can not

know whether the possible errors generated in the tested application are caused by an error in the application itself or by an error in the link or simulating application.

The advantages of the test links provided by the credit card companies are also limited by the number of un-issued test credit card numbers and corresponding response messages that are at the disposal of the developers. These do not include numbers for generation of all possible responses and can therefore not be used in every test case. This limitation becomes especially significant when trying to recreate a reported bug. For instance, imagine a credit card authorization message sending application, suspected to be erroneous due to the fact that it behaves unexpected upon receiving a specific error response from the credit card company. In this case, it would be of great help for the bug fixing developer to be able to recreate the exact communication with the credit card company, including the response message, as took place when the error was encountered. With no credit card number mapped to the specific error, this can not be done.

Being external resources, the simulating applications provided by the credit card companies also have a drawback when it comes to accessibility. Whenever the links for one reason or another are not available, the testing and troubleshooting processes concerning credit card message communication are stalled.

Because of the lack of control, the limited amount of response messages that can be simulated and the accessibility issues, the test links can be considered an insufficient tool for credit card authorization communication tests.

1.2 Company presentation

The project described in this thesis has been conducted at Amadeus SAS at their main development site in Nice, France. Amadeus is acting in three main markets; Global Distribution Systems (GDS), e-travel and IT-services (directed at airlines and other travel service providers) [17].

1.2.1 GDS

The original business idea and still remaining the core of the company is travel distribution. Global Distribution Systems are systems that allow users in disperse parts of the world to find travel information suited to their individual needs.

GDS's are used both by travel agencies to facilitate their service towards their customers as well as by web sites to automatically produce the travel information. They provide the computer network, the terminals, the software and the content that allows airlines, travel agents, hotel chains, car rental firms, ferry and cruise lines, train operators and insurance agents to distribute travel products all over the world.

Amadeus has an extensive international distribution network worldwide with more than 350,000 terminals to travel agencies and airline offices and holds, in strong competition with mainly American Airlines owned Sabre, the position as the number one player in the market.

1.2.2 E-travel

As a result of increasing competition in the GDS market from competitors and lighter low cost solutions, Amadeus is aiming to widen the business of the company. The company therefore today also offers a range of online travel solutions and web-booking tools that enable airlines, corporations, travel agencies and online travel portals to grow online business.

A main step towards the e-travel commitment of Amadeus was the acquisition of e-Travel, the leading US supplier of hosted corporate travel technology solutions in July 2001. Less than a year later, Amadeus launched e-Travel as a new e-commerce business unit that provides global online solutions for airlines, corporations, travel agencies and other travel partners.

Amadeus also has a wide range of joint-venture partners to gain positions in leading online travel sites around the world. Among these are sites like OneTravel.com, Opode.com and Scandinavian travellink.com.

1.2.3 IT-Services

Amadeus has expanded its System User Concept to what is called a Passenger Services Systems offer, targeting travel providers (including airline alliances) and adding Inventory, Yield Management and Departure Control to the distribution facilities offered to System Users. This offer has been packaged together under the Altéa brand, turning passengers into customers.

IT development centers have been established in the UK (London) and Australia (Sydney). These commercial developments and Amadeus' existing common platform for sales form the base of the company's new IT platform for airlines.

Amadeus Altéa integrates sales, inventory and departure control systems, leveraging a single source of data across all three environments. With the Altéa portfolio it is possible to extend the flow of information across the entire customer experience, making it a Customer Management Solution for airlines and airline alliances.

The Altéa suite comprises three solutions:

- Altéa Plan: inventory management system
- Altéa Sell: sales and reservation platform
- Altéa Fly: departure control system

1.3 Problem definition

Whenever an order is being made in the Amadeus GDS, a Passenger Name Record (PNR) is set up. The PNR consists of information fields describing the order. Examples of information stored in these fields are the purchasers name, the flight information, the price, the currency etc. The PNR also includes a field describing the form of payment that the user has chosen. One of the options is payment by credit card.

The work of creating and maintaining the parts of the Amadeus GDS system involved with the credit card communication requires regular sending and receiving of credit card messages. This can be done over the test links offered by the credit card companies but the dependence on these companies is unsatisfying because of the lack of control, the limited amount of response messages that can be simulated and the accessibility issues.

For these reasons the question has been raised of how an application emulating the behavior of the credit card companies can be built in such a way that all of these issues are solved.

An attempt to do this was made in 2002 by Ludovic Sonrel [1]¹. The proposed and implemented solutions described in that document were designed for the Qantas market and for use on Receptor, a simulation tool preceding the one used at Amadeus today, TTServer.

Sonrels solution was useful in that it provided a tool to which it was possible to send credit card request messages and receive response messages from. However, it lacked in dynamicity and ability to handle all credit card message fields² before timing out. It is therefore not a sufficient tool for solving the issues involved in credit card message communication emulation.

The project addressed in this thesis was aimed at resolving the following issues:

- **Lack of control over the simulation process** – In order to be able to draw correct conclusions about the results of a test it is necessary to know why the result was generated. The tester needs to be sure that the results received from the simulation are the correct ones and hence know whether the possible errors generated in the tested application are caused by an error in the application itself or by an error in the link or simulating application. Before the start of the project described in this thesis, the tester did not have this control.
- **Incomplete spectra of communication scenarios** – Prior to the start of this project, all communication scenarios were not possible to simulate. The capability to handle every possible input and response is missing as is the capability to easily conduct emulations of credit card links yet to be implemented.
- **Weak accessibility** – Simulations of credit card message communication could not always be conducted due to problems with transport links or other resources outside the control of the tester.
- **Incapability of handling all credit card message fields** – To fully be able to simulate a credit card message communication, it is necessary to be able to process the whole credit card messages before sending the response. This did not use to be possible.
- **Weak dynamicity** – No emulation tool used to be at hand which was able to consider the incoming fields, process the information according to user defined rules and return the response.

1.4 Report structure

In chapter 2, Background, any information which has been considered needed for the reader in order to be able to comprehend the rest of the report is presented. This chapter includes descriptions of the environment in which the Credit Card Interface Validation and Authorization tool is meant to be used (TTServer, EDIFACT, etc.), an overview of three credit card message standards and the credit card number validation principals. It also presents the previous work in this area and states the requirements and expectations set up before the project was launched.

¹ See 2.5 - Previous work

² See 2.1 - Credit Card Messages

Chapter 3, Solution Proposal Analysis, presents the three solution proposals that were discussed in the project. The pros and cons of the three are discussed and a conclusion of which was considered best is there to be read.

The following chapter, chapter 4, Prototype design, describes the design of the prototype. It includes detailed information on which the different parts of the system are and their roles in the therein. It also includes a table showing the time allocation of this phase.

Chapter 5, Prototype usage, is describing how the prototype is supposed to be used. It begins by presenting two types of users and gives examples of three specific cases in which the tool has been used. Thereafter follows a detailed description, in the form of a step-by-step user's guide, of how to make use of all the functionality of the tool.

The Analysis chapter (chapter 6) presents all the measurements that were made to insure that the goals of the project had been reached. It first describes how the measurements were conducted and then goes on to describe the results. The measurements and results are divided into four main areas: Performance, Dynamicity, Generality and Reliability.

Chapters 7 and 8 (Conclusions and Future work) presents the conclusions drawn from the work in this project and the work that is yet to be done in this area respectively.

2 Background

2.1 Credit Card Messages

The credit card messages that are sent to and received from the credit card companies are used to find out whether the credit card purchase is granted by the credit card company or the issuing bank. The exact flow of these messages varies between credit card companies but generally communication is started by an authorization or pre-authorization request sent by Amadeus. The credit card company then returns a response to the request. If no response has been received within a certain period of time Amadeus considers the request to be timed out and may send a reversal message which the credit card company is expected to respond to.

The structure of the credit card messages also varies depending on which company the credit card is tied to. All credit card message standards however have a basic structure in common. They all include a header field with communication data and a message body which contains the different data fields to be transmitted. Most standards also include one or two Bitmap fields between the header and the data fields which describe which of the possible data fields that are included in the message body. All fields are represented by a bit in the bitmap. If the bit is set to one the field is included and if it is set to zero it is not. There are usually 128 possible data fields so each of the two bitmaps has a size of 8 bytes.

Depending on which standard that is used, the fields of the credit card messages may contain character strings, numeric digits (packed so that one byte contains two symbols) or binary data.

The different credit card companies use different standards of formatting the data to be transmitted over their links. The three links that can be routed to TTServer for test purposes today are Qantas IGW, Visa/MasterCard and American Express but more links are likely to be added in the future.

2.1.1 The Qantas IGW link – AS2805

Qantas is a large airline company based in Australia. The communication initiated when a customer chooses to buy a service from Qantas by credit card via the Amadeus GDS is described in [2] and visualized in Figure 1. The credit card message is in such a case routed to the Qantas Interface Gateway (IGW). From there, Qantas routes the message further to Global Acquiring Bank (GAB) but the communication beyond the IGW, between the IGW and GAB, is transparent to the GDS and should not affect the way it is implemented. The procedure of sending the credit card message via the IGW is done regardless of what kind of credit card vendor is used.

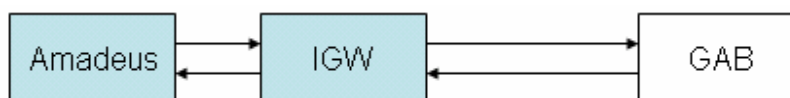


Figure 1 – Amadeus communication with the Qantas Interface Gateway

Messages sent to the Qantas IGW are formatted according to the AS2805 standard. Qantas' implementation of the standard consists of a message header, a message type ID, bitmaps and data fields as shown in Figure 2.



Figure 2 – The Qantas AS2805 implementation structure

2.1.1.1 Header

The first part of the Qantas implementation of the AS2805 standard is a header field containing communication data. The header used is of the P1024 type which consists of a 3 bytes field in front of the message. The header field is not included in credit card messages routed to TTServer³.

2.1.1.2 Message Type ID

The message type ID field is a two byte field identifying the type of message that is being sent. The credit card messages sent from Amadeus to the Qantas IGW can be of one of the following types:

Message 0100

Credit card pre-authorization. This is the type of messages that will be sent to the emulator tool.

A message 0100 sent a second time is considered a manual reversal message of the pre-authorization (the processing code field set to 20 differentiates it from the original 100 message⁴).

Message 0420

If no response message is received within a certain time frame a message 0420 reversal message is sent. This informs the IGW that the original request message should be discarded.

Message 0421

Similar to the 0420 reversal message but this message is used for reversals when a response was not received from the IGW following a 0420 reversal message.

The credit card messages received from the Qantas IGW can be of one of the following types:

Message 0110

Credit card pre-authorization response. Sent as responses to 0100 messages. This is the type of messages that will need to be generated in the emulator tool.

Message 0430

³ See 2.3 - TTServer

⁴ See 2.1.1.4 - Data Fields

Credit card reversal response. Sent as responses to 0420 and 0421 messages.

2.1.1.3 Bitmaps

The two bitmap fields are located between the header and the data fields and describe which of the possible data fields that are included in the message body. All fields are represented by a bit in the bitmap. If the bit is set to one, the field is included and if it is set to zero it is not. There are 128 possible data fields so each of the two bitmaps has a size of 8 bytes. The first bit in the first bitmap is used to determine whether the second bitmap is included or not. If none of the last 64 data fields are present the second bitmap is omitted and the first bit in the first bitmap is consequently set to zero.

2.1.1.4 Data Fields

The data fields contain the information to be sent to the credit card issuer. The fields are all defined by the five parameters Bit Nr, Field Name, Attribute, Size and Content. The Bit Nr is a number depicting the bit in the bitmap which describes whether or not the field is present. The Attribute parameter describes how the data is stored in the field, i.e. numeric, alpha-numeric or binary. The Size parameter states the size of the field or that it is variable and the Content and Field Name attributes describe the contents of the field and its name respectively.

The number of fields in a credit card message varies depending on what is to be communicated but some fields are always present. An example of such a field is the Primary Account Number. It has the Bit Nr 2, has numeric contents of maximum 19 digits. It holds the credit card number found in relief on the front of the card.

More info on the Qantas AS2805 credit card message fields can be found in Appendix A.

2.1.1.5 Response codes

The content of field 39 describes how the issuing bank responds to the request. In Qantas' implementation of the AS2805 standard, the codes described in Table A are available.

Table A – Response codes in the Qantas AS2805 standard

Response Code	Description
00	Approved
01	Refer to Card Issuer
02	Refer to card issuer's special conditions
04	Pick-up card
05	Do not honour
07	Pick-up card - special condition
08	Honour with ID
09	Request in progress
12	Message Format Problem - unable to process

13	Invalid amount
14	Invalid card number
20	Invalid response
21	No action taken
30	Format Error
31	Bank not supported by switch
33	Expired card - pick up
34	Suspect fraud - pick up
36	Restricted card - pick up
38	Allowable number of PIN tries exceeded - pick up card
39	No credit account
41	Lost card - pick up
42	No universal account
43	Stolen card - pick up
51	Not sufficient funds
52	No checking account
53	No saving account
54	Expired card
55	Incorrect PIN
56	No card record
57	Transaction not permitted to cardholder
59	Suspected fraud
61	Exceeds withdrawal amount limits
62	Restricted card
63	Security violation
65	Exceeds withdrawal frequency limits
75	Allowable number of PIN tries exceeded
91	Issuer or switch inoperative
94	Duplicate transmission
97	Reconciliation totals have been reset
98	MAC Error
99	Gateway host not available (Bank link to Gateway not up). Used if there is a timeout from the bank or bank link is not available

2.1.1.6 Message example

Qantas AS2805 Credit Card Authorization Response Message

Message ID:

X'0110'

Bitmaps:

Primary: x'722200012EC18001'

Secondary: N/A

Data Fields:

002 X'164532247214000966'
 003 X'003000'
 004 X'000000000100'
 007 X'0404180418'
 011 X'867716'
 015 X'0000'
 032 X'0856022004'
 035 X'00'
 037 X'F5F0F9F4F1F8F8F6F7F7F1F6'
 038 X'F0F0F0F1F5F8'
 039 X'F3F8'
 041 X'D8C1D5E3C1E24040'
 042 X'E5C940404040404040404040404040'
 048 X'F0F1F2F1404040404040404040E0'
 049 X'0036'
 064 X'0000000000000000'

2.1.2 American Express Link – ISO8583

The American Express link is used to send credit card messages whenever a customer buys a service from the Amadeus GDS with an American Express Card. The messages sent over the link are formatted according to the American Express implementation of the ISO8583 standard which is described in [3]. The standard includes three main blocks; the message type ID, the bitmaps and the data fields, as shown in Figure 3.

Message Type ID	Bitmap	Data Fields
-----------------	--------	-------------

Figure 3 – The American Express ISO8583 implementation structure

2.1.2.1 Message Type ID

The message type ID field is a four byte field identifying the type of message that is being sent. Each digit is represented by an EBCDIC character. The credit card messages sent from Amadeus over the American Express link can be of one of the following types:

Message 1100

Authorization Request. Used to request a transaction of money.

Message 1110

Authorization Response. Sent as a result of a 1100 message. Contains the response code telling the requesting application what the result of the request was.

Message 1804 and 1814

Network Management Request and Network Management Response. When the Amadeus GDS has not sent any messages over the American Express link during two minutes, American Express sends this message to check that the link is still working correctly. If no response is returned from the GDS new 1804 messages will be sent. If a certain number of consecutive tests fail, American Express will investigate the case further and the frequency of 1804 messages will increase until the GDS responds with a 1814 response message. The frequency of Echo Test requests then returns to normal.

2.1.2.2 Bitmaps

The two bitmap fields describe which of the possible data fields that are included in the message body. All fields are represented by a bit in the bitmap. If the bit is set to one the field is included and if it is set to zero it is not. There are 128 possible data fields so each of the two bitmaps has a size of 8 bytes. The first bit in the first bitmap is used to determine whether the second bitmap is included or not. If none of the last 64 data fields are present the second bitmap is omitted and the first bit in the first bitmap is consequently set to zero.

2.1.2.3 Data Fields

The data fields contain the information to be sent to the credit card issuer. The fields are defined by the five parameters Bit Nr, Field Name, Attribute, Size and Content. The Bit Nr is a number depicting the bit in the bitmap which describes whether or not the field is present. The Attribute parameter describes how the data is stored in the field, i.e. numeric, alpha-numeric or binary. The Size parameter states the size of the field or that it is variable and the Content and Field Name attributes describe the contents of the field and its name respectively.

The number of fields in a credit card message varies depending on what is to be communicated but some fields are always present. An example of such a field is the Primary Account Number. It has the Bit Nr 2, has numeric contents of maximum 17 digits. It holds the credit card number found in relief on the front of the card.

More info on the American Express ISO8583 credit card message fields can be found in Appendix A.

2.1.2.4 Response Codes

The content of field 39 describes how the issuing bank responds to the request. In American Express' implementation of the ISO2805 standard, the codes described in Table B are available.

Table B – Response codes in American Express ISO8583

Response Code	Description
000	Approved
001	Approve with ID
003	Approve VIP
092	Approved (Express Rewards Program)

100	Deny
101	Expired card
103	Deny — Invalid Manual Entry 4DBC
104	Deny — New card issued
105	Deny — Account Canceled
107	Refer to card issuer
109	Invalid merchant
110	Invalid amount
111	Invalid card number
115	Service not permitted
122	Invalid card (CID) security code
125	Invalid effective date
181	Format error
182	Please wait
183	Invalid currency code
189	Deny - Cancelled or Closed Merchant/SE
200	Deny - Pick up card
400	Reversal accepted

2.1.2.5 Message Example

American Express ISO8583 Credit Card Authorization Response Message	
Message ID: X'F1F1F1F0'	
Bitmaps: Primary: X'703000210EC08000' Secondary: N/A	
Data Fields:	
002	X'F1F5F3F7F8F2F9F1F0F2F7F8F5F1F0F0F4'
003	X'F0F0F4F0F0F0'
004	X'F0F0F0F0F0F0F0F0F1F8F6F0'
011	X'F8F3F4F7F2F6'
012	X'F0F5F0F4F0F4F1F3F0F5F2F4'
027	X'F2'
032	X'F0F6F3F7F0F1F5F0'
037	X'F5F0F9F4F4F7F1F2F4000000'
038	X'F0F540404040'
039	X'F0F0F1'
041	X'F0F9C2F5F2F7F6F3'
042	X'C1C640E3F0F0F0F0F0F0F0F0F0404040'
049	X'F8F4F0'

2.1.3 VISA Link – ISO8583

The VISA link is used to send credit card messages whenever a customer buys a service from the Amadeus GDS with a VISA credit card. The messages sent over the link are formatted according to the VISA implementation of the ISO8583 standard which is described in [4, 7, 8]. The standard includes four main blocks; the header, the message type ID, the bitmaps and the data fields, as shown in Figure 4.

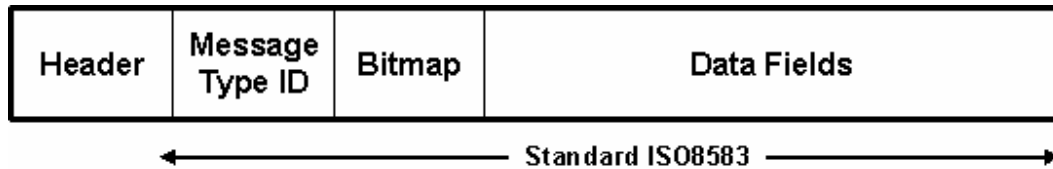


Figure 4 – The VISA ISO8583 implementation structure

2.1.3.1 Header

The header field in the VISA credit card message format consists of 12 (standard) or 14 (reject) fields which specify system ID, routing information, and message processing control codes and flags. The header is defined by Visa and is not part of the standard ISO 8583 message structure, nor is it included in credit card messages routed to TTServer⁵.

2.1.3.2 Message Type ID

The message type ID field is a four byte field identifying the type of message that is being sent. The credit card messages sent from Amadeus over the American Express link can be of one of the following types:

Message 1100

Authorization Request. Used to request a transaction of money.

Message 0101

Repeat Authorization Request.

Message 1110

Authorization Response. Sent as a result of a 1100 message. Contains the response code telling the requesting application what the result of the request was.

Message 0400

Reversal Request. Used to undo a previously sent request

Message 0401

Repeat Reversal Request.

Message 0410

Reversal Response.

Message 1800 and 1810

⁵ See 2.3 - TTServer

Network Management Request and Network Management Response. When the Amadeus GDS has not sent any messages over the American Express link during two minutes, American Express sends this message to check that the link is still working correctly. If no response is returned from the GDS new 1804 messages will be sent. If a certain number of consecutive tests fail, American Express will investigate the case further and the frequency of 1804 messages will increase until the GDS responds with a 1814 response message. The frequency of Echo Test requests then returns to normal.

2.1.3.3 Bitmaps

The VISA implementation of the ISO8583 message standard allows for three bitmap fields. These describe which of the possible data fields that are included in the message body. All fields are represented by a bit in the bitmap. If the bit is set to one the field is included and if it is set to zero it is not. There are 192 possible data fields so each of the bitmaps has a size of 8 bytes. The first bit in the first bitmap is used to determine whether the second bitmap is included or not. Similarly, the first bit in the second bitmap is used to determine whether the third bitmap is included or not. If none of the 64 data fields, which presence is determined in one of the two last bitmaps, are present, that bitmap is omitted and the first bit in the preceding bitmap is consequently set to zero. The third bitmap can only be included if the second one is.

2.1.3.4 Data Fields

The data fields contain the information to be sent to the credit card issuer. The fields are defined by the five parameters Bit Nr, Field Name, Attribute, Size and Content. The Bit Nr is a number depicting the bit in the bitmap which describes whether or not the field is present. The Attribute parameter describes how the data is stored in the field, i.e. numeric, alpha-numeric or binary. The Size parameter states the size of the field or that it is variable and the Content and Field Name attributes describe the contents of the field and its name respectively.

The number of fields in a credit card message varies depending on what is to be communicated but some fields are always present. An example of such a field is the Primary Account Number. It has the Bit Nr 2, has numeric contents of maximum 11 digits. It holds the credit card number found in relief on the front of the card.

More info on the VISA ISO8583 credit card message fields can be found in Appendix A.

2.1.3.5 Response Codes

The content of field 39 describes how the issuing bank responds to the request. In Visa's implementation of the ISO2805 standard, the codes described in Table C are available.

Table C – Response codes in VISA ISO8583

Response Code	Description
00	Approved

01	Refer to Card Issuer
02	Refer to card issuer, special conditions
03	Invalid merchant or service provider
04	Pickup card
05	Do not honour
06	Error
07	Pickup card , special condition (other than lost/stolen card)
12	Invalid transaction
13	Invalid amount
14	Invalid card number (no such number)
15	No such issuer
19	Re-enter transaction
41	Pick up card (lost card)
43	Pick up card (stolen card)
51	Insufficient funds
52	No checking account
53	No savings account
54	Expired card
55	Incorrect PIN
57	Transaction not permitted to cardholder
58	Transaction not allowed at terminal
61	Activity amount limit exceeded
62	Restricted card
77	Previous message located for a repeat or reversal, but repeat or reversal data are inconsistent with original message
80	Invalid date
81	PIN cryptographic error found
83	Unable to verify PIN
85	No reason to decline a request for account number verification or address verification
91	Issuer unavailable or switch inoperative
92	Destination can not be found for routing
93	Transaction can not be completed; violation of law
96	System malfunction or certain field error conditions

2.1.3.6 Message Example

Visa ISO8583 Credit Card Authorization Response Message

Message ID:

X'0110'

Bitmaps:

```

Primary: X'F22000810ED18014'
Second. Bit Map : X'0000000000000020'

Data Fields :
DF2 - X'104972053263029923'
DF3 - X'000000'
DF4 - X'000000220000'
DF7 - X'0131122059'
DF11 - X'010101'
DF25 - X'51'
DF32 - X'0B012345678901'
DF37 - X'F8F1F1F1F0F1F2F3F4F0F5F0'
DF38 - X'F1F2F3F4F5F6'
DF39 - X'F0F0'
DF41 - X'F0F9F0F2F2F7F1F9'
DF42 - X'F1F2F3F4F5F6F7C1C2C3C4C5C6F7C8'
DF44 - X'02F1C1'
DF48 - X'01C1'
DF49 - X'0840'
DF60 - X'0109'
DF62 - X'15E000000000000000D70123456789012345F1F2F3F4'
DF123 -
X'1DF1F2F3F4F540404040C1C2C3C4C5C6C7C8C9C0F1F2F3F4F5F6F7F8F9F0'

```

2.1.4 Other Links

Except for the Qantas, American Express and VISA links, there are others, which might impact the tool developed during the work described in this thesis. These use their own unique implementations of different credit card message standards but being outside the scope of the project, they will not be described in detail here. For the description of the work of making the tool generic it is however useful to know of them.

The links that are most likely to be emulated in the tool in the near future are Nedbank (used by airline company SAA) and RBoS (used by BMI). Nedbank and RBoS both use implementations of the ISO8583 standard. Other links, which may impact the tool, however do not use any of the previously described standards but such which for instance, do not include bitmaps or field numbers. The work described in this thesis has been done with this in mind.

2.2 Credit card number validation

Different credit card companies use different intervals of numbers for their credit card accounts. As shown in **Table D**, VISA card numbers always has the prefix 4 and American Express 34 or 37. Also visible from the table is the fact that the lengths of the numbers may vary.

Table D – Prefix and length of some credit card companies' account numbers.

Credit Card Company	Prefix	Length
---------------------	--------	--------

VISA	4	13, 16
American Express	34, 37	15
Mastercard	51-55	16

Apart from these properties, identifying the credit card company for a certain number, there is usually also a way to determine the validity of a credit card account number by a trailing check digit in the number.

A check digit is a number resulting from calculations made on the rest of the number in order to validate its authenticity. The calculation is made on an original number and the check digit is added to the end. By performing the same calculations on the number and checking whether it returns the expected result, the validity of the number can be determined.

Different algorithms for validating credit card account numbers exist but the one that is most commonly used is the LUHN formula, also known as the mod 10 algorithm [18]. This algorithm is used by all credit card companies mentioned in this report.

The LUHN formula consists of the following steps to check the validity of an account number:

1. Starting from the second digit from the right, double the value of every other digit.
2. If the resulting number from one of these operations is more than one digit long, add the values of the digits to form a one digit number (16 is for example transformed to 7 by adding one and six).
3. Add all digits in the account number, with every other digit transformed as described in step one and two. The resulting value should be a number ending with the digit zero, i.e. 30, 40, 50 etc. for the account number to be validated.

The LUHN formula is a good way for applications sending credit card authorization messages to the credit card companies to make sure that the right credit card number has been typed in by the user. It allows for a local check before sending the message. It does however not say anything more than this about the account. It offers no way to check whether the account number has been issued or not and the issue of whether the purchase can be granted or not can only be determined by the credit card company and the financial institution responsible for the account.

2.3 TTServer

Amadeus TTServer (Test Tool Server) is an internal test tool with the purpose of emulating any EDIFACT⁶ client and/or server. Messages are sent to and from TTServer scenarios in order to test the functionality of applications created at the company. TTServer is briefly described below and more thoroughly in [5].

The Test Tools is a set of three applications:

- The Injector is used to send messages and check responses.
- The Receptor is used to receive messages and send back the suitable reply.

⁶ See 2.4 - Edifact

- The Router is used to transfer messages from TPF (the Transaction Processing Facility conducting the sending of messages) to the suitable Receptor. The Test Tools Server includes an engine integrating the Injector, Receptor and Router components and a graphical console to manage the engine. These components are able to react to each other's events.

The application can manage different types of message formats. Primarily, it supports EDIFACT messages but it can be easily modified to support other formats, like XML.

2.3.1 Receptor

With receptor, TTServer emulates the behavior of a server. As shown in Figure 5 the application being tested sends request messages to TTServer. When receiving the request, TTServer looks for a match of it in the pre-written scenario files and returns the response associated to the request.

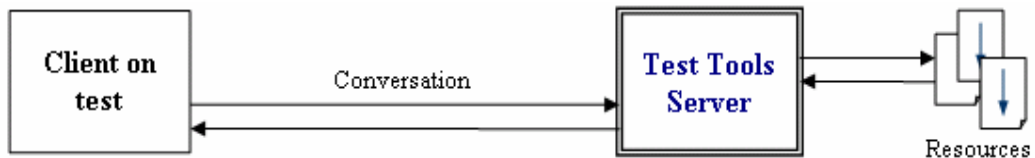


Figure 5 – Communication between tested application and TTServer used as receptor

2.3.2 Injector

With the Injector, TTServer emulates the behavior of a client. In this case, it sends messages to the tested application according to the scenario files. When the response is received TTServer compares it to the expected response described in the scenario files. The logs are stored in output files. The communication flow of the injector is shown in Figure 6.

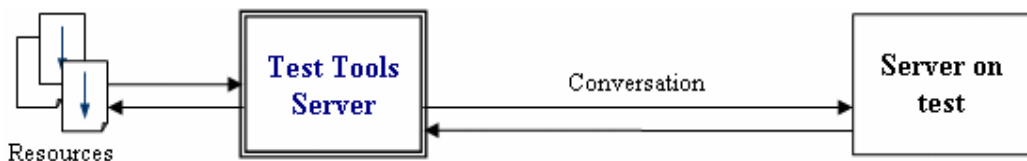


Figure 6 - Communication between TTServer used as injector and tested application

2.3.3 Router

The router component provides a mean of routing messages from TPF to receptor applications. It uses a mapping list of user IDs (Amadeus Terminal ID - ATIDs) and IP addresses/ports. The message flow through the router is visualized in Figure 7.



Figure 7 – Test Tool Router behavior

2.3.4 Dynamic responses

When using TTServer as a receptor the incoming requests are, as described above, matched against a set of predefined expected request messages and a corresponding prewritten response is sent. This way of returning responses statically is sometimes insufficient. Therefore, TTServer also features the ability to include python scripts in the processing of messages.

When a message is received at a TTServer scenario file supporting dynamic responses, the message can partially or in whole be passed on to python scripts inside the scenario file with the help of regular expressions and certain commands of TTServer. In the python scripts the data can be processed to dynamically create the appropriate response depending on the incoming message.

If the scripts used to create the dynamic responses are large, it is a good idea to split them up into Python modules and/or classes, preferably in separate files. This is possible from the scenario file Python script as it is from any other Python script. The object oriented features of Python [9, 10] allows for a wide variety of message processing possibilities.

2.4 Edifact

EDIFACT, also known as UN/EDIFACT, stands for Electronic Data Interchange for Administration, Commerce and Transport. It is an International EDI message standard introduced by the United Nations Economic Commission (UN/ECE) by combining UN/GTDI and ANSI X12 standards [11, 13] and is described in the ISO 9735 standard [12, 14]. It is today maintained by United Nations (UN) committee, UN/EDIFACT Working Group (EWG) [15].

The UN/ECE has prepared the Message Design Guidelines, included in the UN/ECE Trade Data Interchange Directory which was published in 1988 and amended with small changes in 1990 [16].

According to [16], the organization supporting the usage of the standard and maintaining the directory service in Europe is the Western European EDIFACT Board (WE/EB). This board is responsible for the coordination of the message design groups for the ten different application fields. The application fields are trade, transport, customs, finance, construction, statistics, insurance, tourism, health care and social administration.

The two main organizations dealing with Edifact are TT&L (Travel Tourism and Leisure – a United Nations working group) and IATA/ATA (International Air Transport Association/Air Transport Association America).

2.4.1 Character set

EDIFACT provides rules for the syntax of messages to be interchanged between two parties. Other standards like ISO/OSI service specifications and protocols are recommended to be followed for the communication of the messages. The standard is specified as levels which differ in the character sets used in them. Certain characters are reserved for use as terminator, separator and release character. The meaning and use of the different characters are shown in Table E.

Table E – Character meaning in EDIFACT

Description	Character(s)	Use
Letters, upper case	A to Z	Data
Numerals	0 to 9	Data
Space character		Data
Full Stop	.	Data
Comma	,	Data
Hyphen/minus sign	-	Data
Parentheses signs	()	Data
Oblique stoke	/	Data
Equal sign	=	Data
Exclamation mark*	!	Data
Quotation mark*	"	Data
Percentage sign*	%	Data
Ampersand *	&	Data
Semi-colon*	;	Data
Less-than sign*	<	Data
Greater-than sign*	>	Data
Apostrophe	'	Service Character: Segment terminator
Plus sign	+	Service Character: Segment tag and data element separator
Colon	:	Service Character: Component data element separator
Question mark	?	Service Character: Release character. Restores the normal meaning of characters ' + : ?'. if immediately following one of them. E.g. 10?+10=20 means 10+10=20. Question mark is represented by ??.
Asterisk	*	Service Character: Used to separate repeated occurrences of the same Data Element within a Data Segment.

* Can not be used internationally in telex transmissions

2.4.2 Structure

All EDIFACT Data consists of segments which are ended by the Service Character ' '. The main building blocks of the Edifact format are Interchanges, Messages, Segment Groups,

Data Segments and Data Elements. These are accompanied in the transmissions by the Service Characters. The structure of EDIFACT messages is shown in Figure 8.

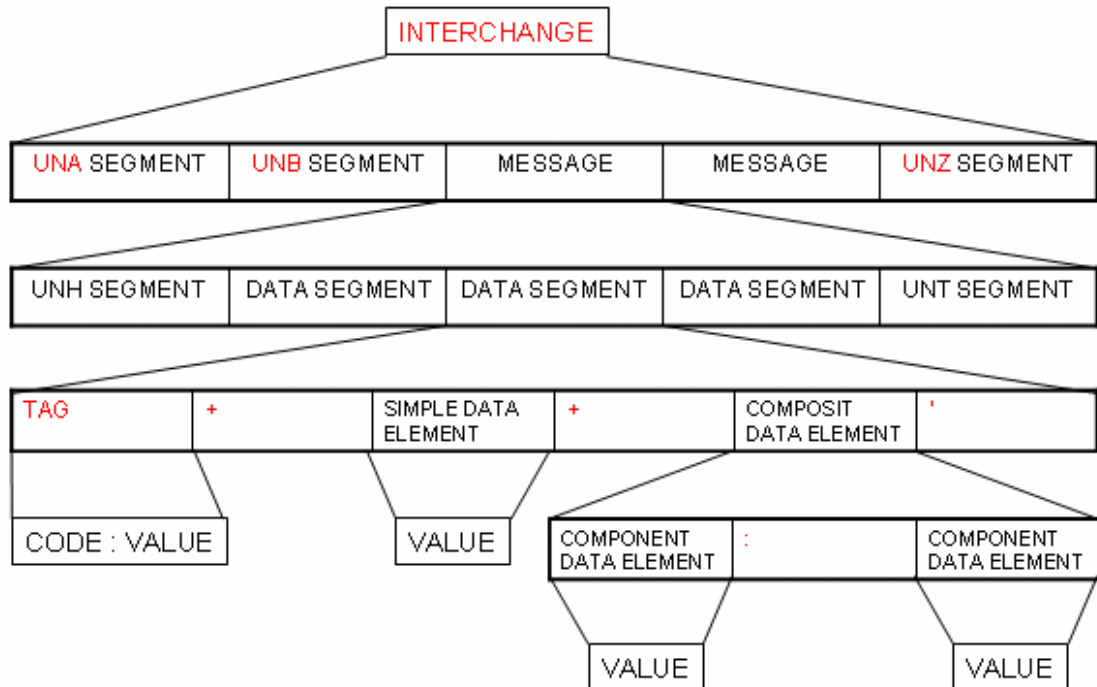


Figure 8 – The structure of an EDIFACT message [69].

An Interchange is a collection of Messages preceded by one or two Header Segments (UNA and/or UNB) and followed by a Trailer Segment (UNZ). Header and Trailer Segments in all levels of the Edifact hierarchy are examples of Service Segments. An Interchange level view of an EDIFACT Interchange is shown in Figure 9.

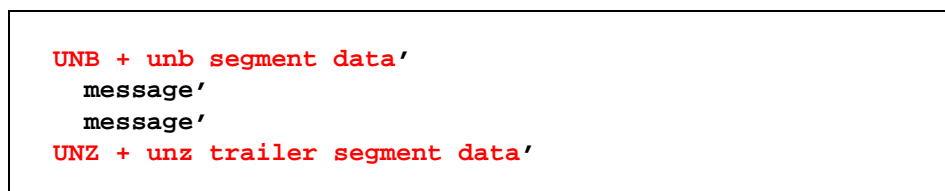
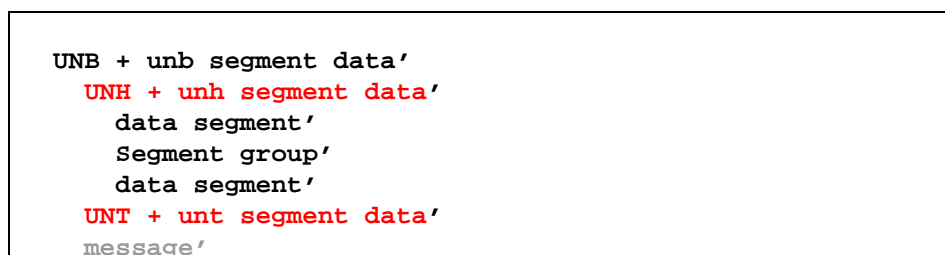


Figure 9 – Interchange level view of EDIFACT Interchange

Messages are built up by one or more Data Segments or Segment Groups preceded by a Header Segment (UNH) and followed by a Trailer Segment (UNT). A Data Segment level view of an EDIFACT Interchange is shown in Figure 10.



```
UNZ + unz trailer segment data'
```

Figure 10 – Data Segment level view of EDIFACT Interchange

Segment Groups consists of more than one Data Segment and are started by a Trigger Segment preceding Data Segments. A Segment Group level view of an EDIFACT Interchange is shown in Figure 11.

```
UNB + unb segment data'
  UNH + unh segment data'
    data segment'
      Trigger segment'
        data segment'
          data segment'
            data segment'
              UNT + unt segment data'
                message'
              UNZ + unz trailer segment data'
```

Figure 11 – Segment Group level view of EDIFACT Interchange

Data Segments consist of a Segment Tag and Data Elements separated by the Service Character +. If no value is present for a specified Data Element, the Service Characters surrounding it (+) must still be present. A Data Element level view of an EDIFACT Interchange is shown in Figure 12.

```
UNB + unb segment data'
  UNH + unh segment data'
    data segment'
      Trigger segment'
        TAG + data element + + data element'
          data segment'
            data segment'
              UNT + unt segment data'
                message'
              UNZ + unz trailer segment data'
```

Figure 12 – Data Element level view of EDIFACT Interchange

A Data Element can be a Simple Data Element (SDE), holding a value, or a Composite Data Element (CDE). Composite Data Elements are built up by Component (Simple) Data Elements holding values and separated by the Service Character :. If no value is present for a specified Data Element in a Composite Data Element, the Service Characters surrounding it (:) must still be present. A Composite Data Element level view of an EDIFACT Interchange is shown in Figure 13.

```
UNB + unb segment data'
  UNH + unh segment data'
```



```

data segment'
Trigger segment'
TAG + SDE + + SDE : : : SDE : : SDE'
data segment'
data segment'
UNT + unt segment data'
message'
UNZ + unz trailer segment data'

```

Figure 13 – Composite Data Element level view of EDIFACT Interchange

Note that the space character is a value of its own⁷ and should not be included anywhere in an Interchange if not part of a value to be transmitted. In Figure 9 through Figure 13, the space characters have been inserted for increased readability.

Figure 14 shows an example of a complete EDIFACT Interchange comprising one message with thirteen segments, header and trailer segments included.

UNB+IATB:1+1APPC+LHPPC+940101:0950+1'	Interch. Header
UNH+1+PAORES:93:1:IA'	Msg. Header
MSG+1:45'	Data Segment
IFT+3+?*AMADEUS AVAILABILITY?*'	Data Segment
ERC+A7V:1:AMD'	Data Segment
IFT+3+NO MORE FLIGHTS'	Data Segment
ODI'	Trig. Segment
TVL+240493:1000::1220+FRA+JFK+DL+400'	Data Segment
PDI++C:3+Y::4+F::1'	Data Segment
APD+74C:0:::6+++++1A'	Data Segment
TVL+240493:1740:::2030+JFK+MIA+DL+081'	Data Segment
PDI++C:4'	Data Segment
APD+EM2:0:1630:::6+++++DA'	Data Segment
UNT+13+1'	Msg. Trailer
UNZ+1+1'	Interch. Trailer

Figure 14 - Example of a complete Edifact Interchange

When using the Edifact format to transmit messages it is encouraged to use predefined messages to an as large extent as possible. Many messages are predefined by UN, IATA or ATA but when these messages are insufficient Amadeus specific ones have been

⁷ See 2.4.1 - Character set

developed internally at the company. New messages may be added to the list of such predefined messages if approved by the Edifact Board at Amadeus.

2.4.3 HSFREQ/HSFRES

The EDIFACT message used to send credit card messages to TTServer scripts is called HSFREQ and the one used to return the responses in is called HSFRES. Both of these messages consist of only one data segment containing the whole credit card message. The fields of the credit card message have thus not been separated into individual fields in the EDIFACT message. This is to make the emulated credit card messages as similar to the real ones as possible.

The credit card message in the data segment in the HSFREQ/HSFRES EDIFACT messages is not the exact same message as would have been sent over the real test links. This is because the real messages contain fields of binary format⁸ which EDIFACT is unable to handle. Before a HSFREQ message is created a translation of the credit card message is therefore necessary. An example is described in .

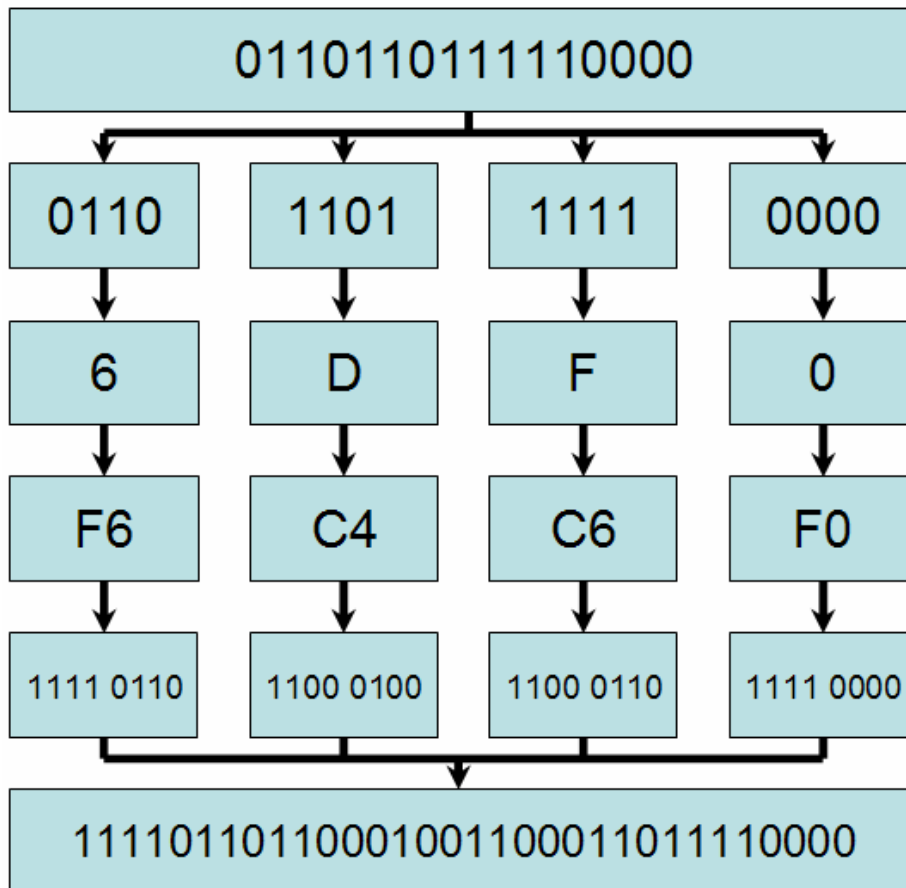


Figure 15 – Example illustration of the translation from original credit card message to HSFREQ/HSFRES compatible credit card message

⁸ See 2.1 - Credit Card Messages

For every four bits of the credit card message, the hexadecimal value is calculated. From each of these values the EBCDIC character value representing it is determined. The binary representation of these values is what is then inserted into the EDIFACT message. Because the values of EBCDIC characters have a size of one byte each, the HSFREQ/HSFRES credit card messages are always twice as big as the original ones.

The reversed process take place when the HSFRES response is received.

2.5 Previous work

Attempts to create an emulation tool for credit card interface validation were made in 2002 by Ludovic Sonrel. His work is described in [1]. The proposed and implemented solutions in that document were designed for the Qantas market⁹ and for use on Receptor, the simulation tool that preceded TTServer.

In the first solution created, the entire untransformed credit card message was sent to TTServer. However, TTServer's inability to extract binary data made it impossible for this solution to create a dynamic response to the received data. Instead the response was hard coded. The static nature of this solution made its usefulness very limited.

In the second solution, the credit card message data was transformed so that each set of four bits was translated into the EBCDIC representation of its value. Hence, a byte containing 00101100 was translated into the EBCDIC characters '2C'. In this way the information in the credit card message could be interpreted by TTServer and the response could be made dependant on it. However, the transformation from sets of four bits into bytes doubled the message size and therefore the processing time. This caused the sending application to regard the message processing as timed out after ten seconds (the maximum time out time in TTServer at the time) and therefore to send a reversal message. Because of this, the TTServer scripts were made to consider only a limited portion of the credit card authorization message's data fields. Among these fields was the System Trace Audit Number field¹⁰. This made it possible to respond uniquely to any credit card message.

The second solution is an improvement to the first one but has obvious flaws. The whole credit card message can not be considered and a lot of time is consumed in the processing. Furthermore, even though the second solution is dynamic in the sense that it considers the contents of the credit card message and makes the response dependent on it, it is not very useful since no real processing is being made. The solution simply reads the fields and copies them to the appropriate places in the response. Furthermore, the bitmap field is not regarded. The solution to be the result of this project should be able to consider the incoming fields, process the information and return a response dependent on the processed information.

A third alternative solution is proposed in Sonrel's document but has not been implemented. It suggests the creation of a new Edifact message constructed to include all fields from the credit card messages structured in to support the Receptor tool, preceding the TTServer.

⁹ The solution was made for reception of messages of the AS2805 standard, see 2.1.1 - The Qantas IGW link – AS2805

¹⁰ See 2.1.1.4 - Data Fields

2.6 Requirements

The project addressed in this thesis is required to result in a prototype solution for processing credit card messages and returning responses dynamically or a solution showing that this is not possible. Dynamicity in this case means ability to send different response codes depending on the input. The solution should be developed for the TTServer environment since this is the test environment used at Amadeus. Hence, all development requirements assigned by TTServer are inherited¹¹.

Evaluations of the performance of the new product should be conducted to determine whether the solution is an improvement of the existing solution or not.

If a faster solution is developed, it should be implemented for the Qantas, Visa/MasterCard, and American Express links. It should however be generic enough to be applicable to other credit card company links as well that might be added in the future.

Documentation on the use of the solution should be written, including information on how to implement the solution for new credit card company links.

If a faster solution could not be developed, documentation on why this was the case should be written.

2.7 Expectations

The prototype solution developed as part of the project is expected to be an improvement to the existing solutions. It will be considered an improvement if the time needed for processing the credit card messages and returning a response is reduced. Hence it is expected that the number of fields that can be read and processed before time out should increase. It is however not expected that the finished solution will be able to process all credit card message fields before time out.

¹¹ EDIFACT, Python etc. See 2.3 - TTServer

3 Solution Proposal Analysis

3.1 General

In the specification of the project, three solutions to the credit card interface validation and authorization problem were proposed. The two first of them were first proposed By Ludovic Sonrel in [1]. This chapter describes all three solution proposals and the conclusion regarding which one is the best suited to implement as the final solution.

3.2 Implementation of existing solutions for TTServer

3.2.1 Description

Ludovic Sonrel has created three scenarios for Receptor¹². The first scenario is a static solution that is capable of recognizing a credit card request but ignores the contents of it. A statically created (hard-coded) authorization response is generated. This scenario allows for testing of credit card message sending applications' ability to correctly send requests and receive responses. It does however not allow variable input and output. All changes must be made directly in the TTServer scenario code.

The second scenario is similar to the first one but instead of sending an authorization response an error response is sent (ALLOWABLE NUMBER OF PIN TRIES EXCEEDED - PICK UP CARD)¹³.

The third scenario is a more dynamic solution that captures some fields from the incoming message and echoes them into the response. This allows for sending authorization request messages that are to some extent variable. The message unique System Trace Audit Number field¹⁴ is for instance echoed which makes it possible to create correct replies to messages with any value in this field. All fields are however not captured and no processing of the fields is being done.

3.2.2 Benefits

- It is known to work. The scenarios created by Sonrel have all been successfully implemented for Receptor and should be possible to implement for TTServer.
- Allows input of partially variable authorization request messages.
- Allows testing credit card message sending applications' ability to correctly send requests and receive responses.

3.2.3 Drawbacks

¹² See 2.5 - Previous work

¹³ See 2.1.1.5 - Response codes

¹⁴ See 2.1.1.4 - Data Fields

- Does not allow completely variable input and output.
- Does not process incoming data to make response dependent on it.

3.3 Creation of new Edifact message

3.3.1 Description

TTSerVer can only handle messages of the Edifact format but credit card messages do not follow this standard. In the three scenarios from Sonrel's solution, the credit card messages are therefore converted into the character representation of the hexadecimal values of the messages binary code. The resulting character string is then placed as a field in an Edifact message called HSFREQ. In [1] the creation of a new Edifact message is proposed as a possible solution. By creating a new Edifact message every field in the credit card authorization message could be mapped to a corresponding field in the Edifact message. It is suggested by Sonrel that this might solve the problem imposed by Receptor of only being able to capture a portion of the fields before time out.

The creation of a new Edifact message creates a need for a converter that would turn the credit card message into the new Edifact message before sending it to TTSerVer.

To create a new Edifact message one needs to first model the message in a correct Edifact manner. Then the message is to be presented to the Edifact committee and in a later stage to the Edifact board, for approval.

3.3.2 Benefits

- The new Edifact message could easily be sent to TTSerVer and the individual fields could quickly be identified by the Edifact separators.

3.3.3 Drawbacks

- Relevance. In order to be able to correctly test credit card messages, it is necessary that the simulation tool behaves as much as possible as the real credit card company applications would do¹⁵. It is therefore desired to be able to send messages that are as similar as possible to the real credit card messages. Splitting the messages up and putting them in a new Edifact message would not be a step in that direction. It raises the question of what is really being tested, credit card messages or an Edifact message.
- Need for a converter. An external program/script would have to be created.
- Unnecessary. When Sonrel was proposing this solution, he was facing the time out problem of Receptor and the proposition was intended as a possible solution to this. It is likely that this problem does not exist in TTSerVer at all. Creating a new Edifact message would therefore only move the capturing of individual credit card message fields from TTSerVer to the converter, a process unlikely to speed up the over all process.
- Time consuming. The process of piloting a new Edifact message, having it approved by the Edifact committee and then the Edifact board before being able

¹⁵ See argumentation in 1.1 - Problem Area

to start creating scripts to handle the data in the credit card messages would take time. This is not a critical issue but should be considered in relation to other drawbacks.

3.4 Python script based solution

3.4.1 Description

The behavior of a TTServer simulation solution is determined by a set of scenario files, possibly including Python scripts and separate Python modules. When a message is received by the TTServer solution it can be made to extract data from the message, process it and send a response dependent on the processed data. This solution would, if successful, be an enhancement of the relatively static solutions created by Sonrel and converted for TTServer.

3.4.2 Benefits

- Fully dynamic. The solution would be able to emulate all relevant scenarios possible in real credit card message communication with the credit card companies.
- Realistic messages. The credit card messages sent to the solution would be changed as little as possible for applicability to TTServer. The only conversion would be from bits to their hexadecimal values expressed in characters.
- No new converter needed. All the processing of the credit card messages data fields would be conducted by scenario files and Python scripts in TTServer.

3.4.3 Drawbacks

No obvious drawbacks.

3.5 Conclusions

Considering the limited amount of data fields that are captured when simply implementing Sonrel's Receptor based scripts for TTServer and the static nature of this solution, it must be concluded that this solution needs to be improved in order to serve as an adequate tool for emulating credit card company applications.

It is also concluded that the lack of relevance and the fact that extra software would have to be created, excludes the solution of creating a new Edifact message as a possible option.

The conclusion is instead that the best suited solution for the problem is the third alternative, improvement of Sonrel's scripts to apply dynamic capturing and processing of the data and make the response dependent on it.

4 Prototype design

4.1 Overview

The prototype consists of three main parts; the emulation tool, The Message Standard Description (MSD) files¹⁶ and the Credit Card – Response Mapping File. The application utilizing the prototype sends a credit card message to the emulation tool. The tool then opens one of the MSD files in the predefined MSD directory to find out how to read the message and compose the response. If the MSD file matches the incoming message the response is composed according to the contents of the MSD and sent back to the user application. If the MSD file does not match the incoming message, another MSD file is opened. This continues until a match has been found or all MSD files have been tested without finding a match. A UML view of the relationship between the user application, the emulation tool, the MSD files and the Credit Card – Response Mapping File is shown in Figure 16.

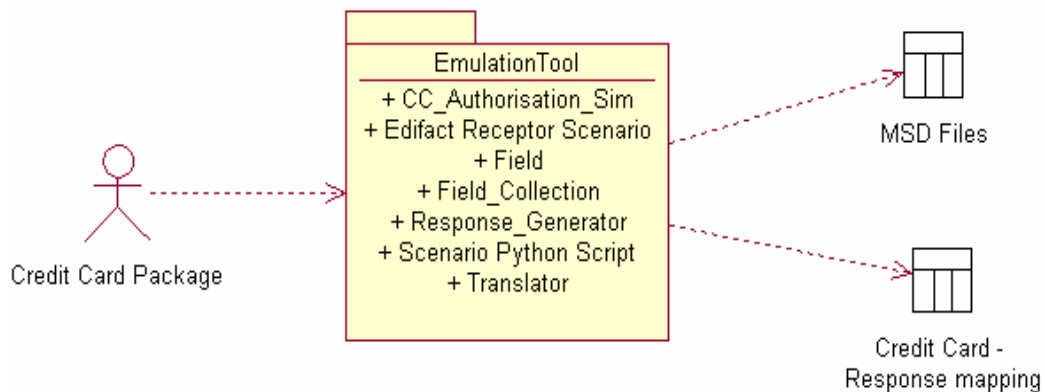


Figure 16 – UML view of the relationship between user application, emulation tool, MSD files and Credit Card – Response Mapping File

The emulation tool in the prototype consists of a receptor file in TTServer and five Python classes in external modules. After starting the TTServer Receptor file, the tool is ready to receive EDIFACT messages. Upon receiving the message, the Receptor file passes the credit card message, which is embedded in the EDIFACT message, to the python script in the file. From here, the emulating class (CC_Autorisation_Sim), contained in an external module, is instantiated. This class supervises the emulation and delegates the tasks to the other classes which it instantiates. Figure 17 shows how the classes are connected to each other.

¹⁶ See 4.2 - Message Standard Description files

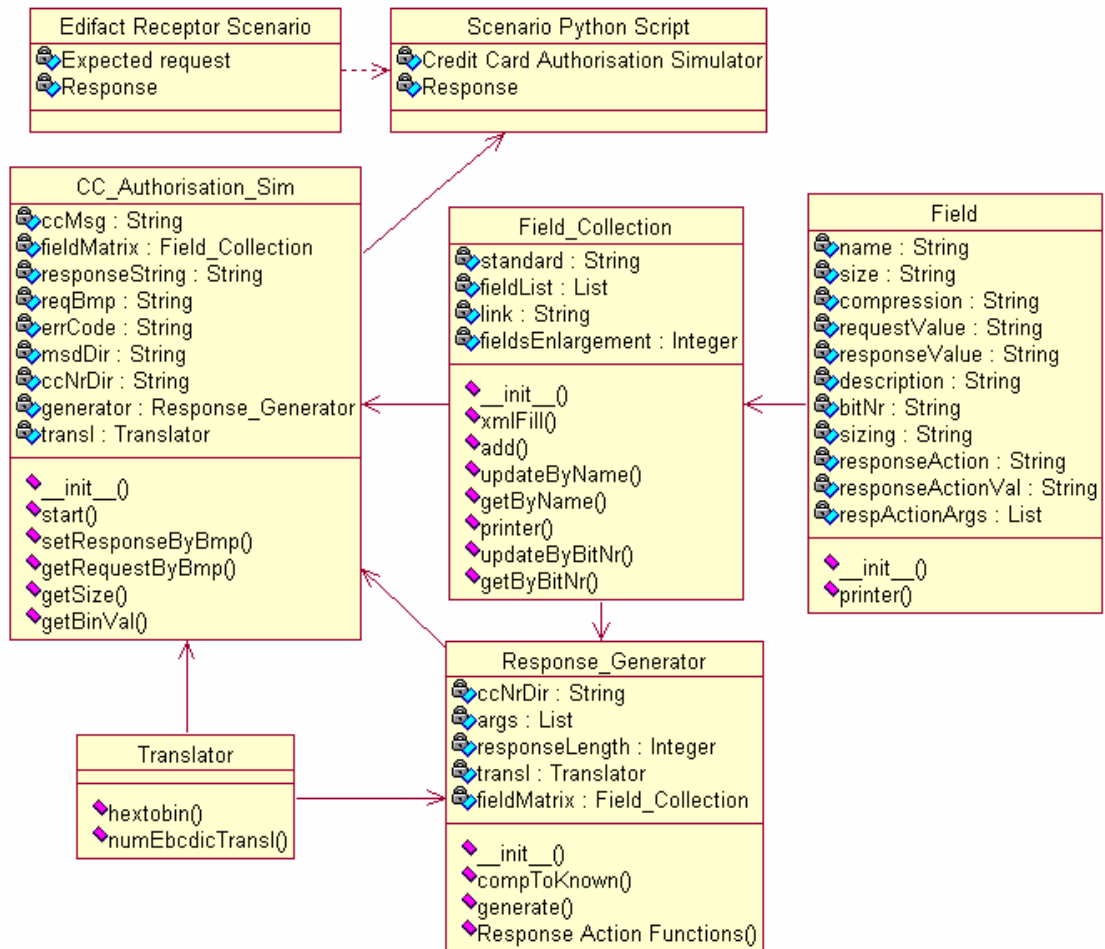


Figure 17 – Class Diagram of the Emulation tool

The emulator starts by looping through all MSD files in the predefined MSD directory. When a matching MSD file is found the `CC_Authorisation_Sim` instance of the `Field_Collection` class is filled with the contents of the MSD file. This “virtual” MSD file is then updated with the parsed request message fields and later the generated response fields.

4.2 Message Standard Description files

The Message Standard Description Files (MSD files) are XML files informing the emulation tool of how an incoming credit card request message is to be read and how the response should be created. They are of particular importance to the Credit Card Interface Validation and Authorization tool since they are the only places, together with the Response Action Functions¹⁷ and the Credit Card - Response Mapping File¹⁸, where users should update the tool in order to adapt it to their needs. The structure of the MSD files will therefore be described in some detail below.

¹⁷ See 4.3.1 - Response Action Functions

¹⁸ See 4.3.2 - Credit Card - Response Mapping File

The MSD files can be considered Amadeus adapted versions of the standard descriptions provided by the credit card companies and describe the fields to be used in the standard. The basic structure of an MSD file comprises six tags as shown in Figure 18. The **standard** tag is the container of the entire standard description and the **field** tag contains the description of specific fields. The **size**, **compression**, **responseAction** and **description** tags are attribute tags inside the **field** tag.

```

<standard>
  <field>
    <size></size>
    <compression></compression>
    <responseAction></responseAction>
    <description></description>
  </field>
  ...
</standard>

```

Figure 18 – Basic structure of an MSD file

4.2.1 standard

The **standard** tag comprises the entire standard description and contains attributes as shown in Table F:

Table F – Attributes in the standard tag

Attribute	Description
name	Name of the standard. Does not have to be an official name but should be one that uniquely identifies the described standard.
link	Name of the link to be emulated with the described standard.
fieldsEnlargement	The enlargement of field sizes imposed by the transformation of the credit card message into characters.

Example:

```
<standard name="ISO8583_VISA" link="VISA" fieldsEnlargement="2">
```

4.2.2 field

The **field** tag within the **standard** tag holds information on the different fields in the standard. Table G shows the attributes of the tag.

Table G – Attributes in the field tag

Attribute	Description
name	Name of the field. Generally a number.
bmpNr	The number of the bit in the bitmap, from left to right, that represents the presence or absence of the field.

Example:

```
<field name="3" bmpNr="2">
```

4.2.3 size

The **size** tag within the **field** tag holds information on the size of the field. The tag has one attribute called **sizing**. This attribute describes the way the size is stated in the tag. Table H shows the possible values of the attribute.

Table H – Values of the sizing attribute in the size tag

Attribute value	Description
Fixed	The size of the field is fixed. The value stated in the tag is the size of the field.
variable_packed	The size of the field is variable. The value stated in the tag is the length of the initial part of the field which holds the length of the field (the initial part itself excluded). The length in this part is stated in packed format. A field with this sizing and the value 2, in an incoming message where the field starts with "10345" is 10 digits long plus the two length digits.
variable_binary	The size of the field is variable. The value stated in the tag is the length of the initial part of the field which holds the length of the field (the initial part itself excluded). This part states the length as binary code. A field with this sizing and the value 2, in an incoming message where the field starts with "10345" is 16 digits long plus the two length digits (10 = 0001 0000 = 16).

Example:

```
<size sizing="fixed">8</size>
```

4.2.4 compression

The **compression** tag within the **field** tag holds information on how the incoming data in the field is stored. The value of the tag should be one of the alternatives displayed in Table I.

Table I – The values of the compression tag

Tag value	Description
Byte	The value of the field is stored so that one byte contains one value (eight bits per value)
Packed	The value of the field is stored so that one byte contains two values (four bits per value)
Binary	The value of the field is stored so that one byte contains eight values (one bit per value)

The `compression` tag has no attributes.

Example:

```
<compression>Binary</compression>
```

4.2.5 responseAction

The `responseAction` tag within the `field` tag holds information on how the present field's response value should be generated. This tag has no attributes but three sub tags. These are the `action` tag, the `value` tag and the `respActionArgs` tag.

4.2.5.1 action

The `action` tag within the `responseAction` tag holds information on how to generate the response value. The value of the tag should be one of the alternatives displayed in Table J.

Table J – The values of the action tag.

Tag value	Description
Echo	The response value should be echoed (copied) from the request value.
Value	The response value is stated in the MSD file, in the <code>value</code> tag ¹⁹ . This is the default alternative and assumed if the tag is left empty.
Generate	The response value should be generated by a user defined Response Action Function in the <code>Response_Generator</code> class ²⁰ .

The `action` tag has no attributes.

Example:

```
<action>Echo</action>
```

4.2.5.2 value

¹⁹ See 4.2.5.2 - value

²⁰ See 4.3.1 - Response Action Functions

The **value** tag within the **responseAction** tag holds data representing different things depending on the value of the **action** tag²¹. The meanings of this tag are displayed in Table K.

Table K – The meanings of the value tag

action tag value	value tag meaning
Echo	Not used
Value	The value to be put in the response for this field
Generate	A key word that is mapped to a Response Action Function in the Response_Generator class ²² .

The **value** tag has no attributes.

Example:

```
<value>F0F0F0F1F5F8</value>
```

4.2.5.3 respActionArgs

The **respActionArgs** tag within the **responseAction** tag holds the arguments to be passed to the Response Action Function in the **Response_Generator** class and is therefore only used if the **action** tag value is **Generate**. This tag has no attributes but one sub tag that can be repeated for as many arguments as needed. The sub tag is called the **arg** tag.

Example:

```
<respActionArgs>
  <arg>002</arg>
  <arg>example value</arg>
</respActionArgs>
```

4.2.6 description

The **description** tag within the **field** tag holds a description of the field.

Example:

```
<description>BITMAP Secondary</description>
```

4.2.7 Example

Figure 19 shows an example of a short MSD file describing one field.

```
<standard name="AS2805" link="Qantas" fieldsEnlargement="2" >
```

²¹ See 4.2.5.1 - action

²² See 4.3.1 - Response Action Functions

```
<field name="038" bmpNr="37">
  <size sizing="fixed">6</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Value</action>
    <value>F0F0F0F1F5F8</value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>Authorisation ID Response</description>
</field>
</standard>
```

Figure 19 – Example of MSD file XML code

4.3 Response message creation

The response message is created with regard to the response bitmap stated in the MSD file and the information, also found in the MSD file, about the fields to be included.

If the field is set in the `responseAction action` tag to `Echo`, the corresponding incoming field value is re-used as the response value. If it is set to `Value` (or nothing, or anything else) the value from the MSD file is used as response value. If it is set to `Generate` a Response Action Function is called.

4.3.1 Response Action Functions

All Response Action functions are located in the `Response_Generator` class. This class also contains a `generate()` function. Whenever a field's `responseAction action` tag is set to `Generate`, this function is called with among other things the key word from the `value` tag and the arguments from the `respActionArg` tag as arguments.

In the `generate()` function, the key word is mapped to the Response Action Function of choice. If a function exists that performs the desired tasks in order to generate the response, the key word should be mapped to this function. If it does not, the user of the system can add whichever functionality that is found necessary, by defining a new Response Action Function.

In the Response Action Function, the user has access to the Credit Card – Response Mapping File, the translation functions of the `Translator` class²³ and all information about every MSD file defined field in the standard, including the request values. In this way, the response can be generated depending on this data.

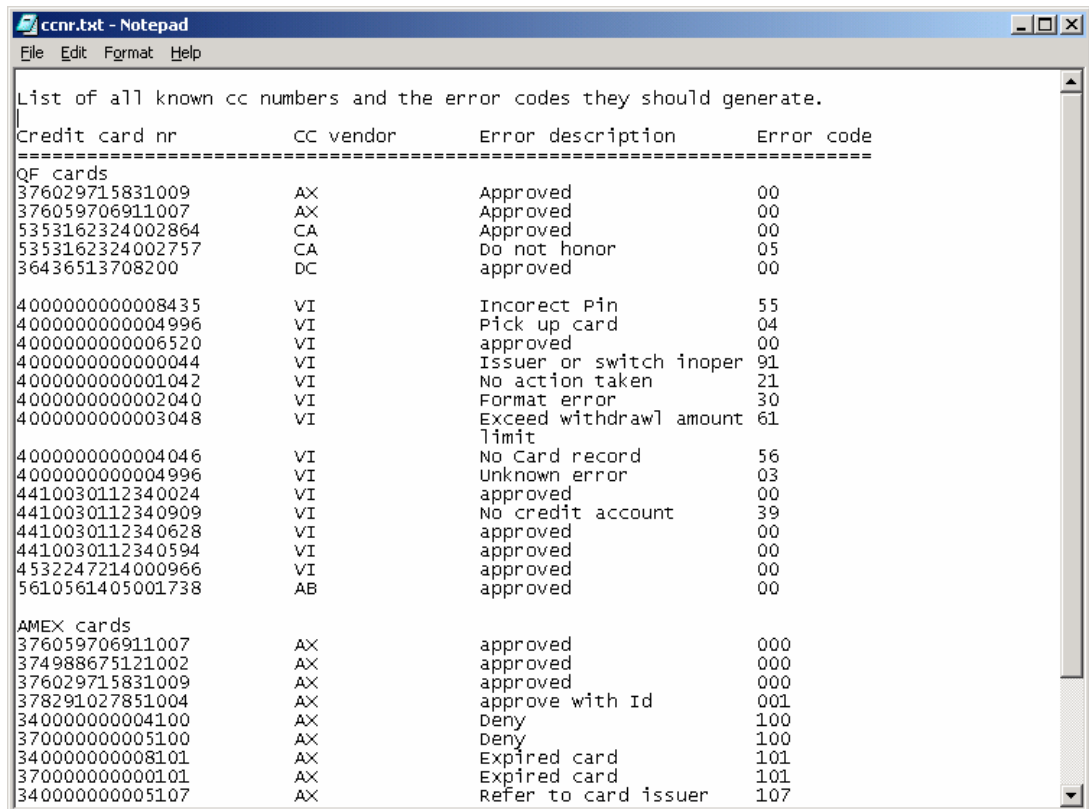
This feature has been included in the system in order to make it as generic as possible and allow for any field in any standard to be generated in exactly the way that the user wants it to be.

²³ See 4.1 - Overview

4.3.2 Credit Card - Response Mapping File

The real test links, provided by the credit card companies, generates certain response codes when some predefined credit card numbers are used. It is desired to emulate this in the tool. The Credit Card – Response Mapping File adds this feature.

This file contains credit card numbers and the response values that they are supposed to generate. It can be used to make the tool generate a specific response code for a certain credit card number. This is done from a Response Action Function. Figure 20 shows an example of a Credit Card – Response Mapping File.



```

List of all known cc numbers and the error codes they should generate.
Credit card nr      CC vendor      Error description      Error code
-----
QF cards
376029715831009    AX             Approved               00
376059706911007    AX             Approved               00
5353162324002864    CA             Approved               00
5353162324002757    CA             Do not honor           05
36436513708200     DC             approved               00

4000000000008435    VI             Incorect Pin           55
4000000000004996    VI             Pick up card           04
4000000000006520    VI             approved               00
4000000000000044    VI             Issuer or switch inoper 91
4000000000001042    VI             No action taken        21
4000000000002040    VI             Format error            30
4000000000003048    VI             Exceed withdrawl amount 61
limit

4000000000004046    VI             No Card record         56
4000000000004996    VI             Unknown error          03
4410030112340024    VI             approved               00
4410030112340909    VI             No credit account      39
4410030112340628    VI             approved               00
4410030112340594    VI             approved               00
4532247214000966    VI             approved               00
5610561405001738    AB             approved               00

AMEX cards
376059706911007    AX             approved               000
374988675121002    AX             approved               000
376029715831009    AX             approved               000
378291027851004    AX             approve with Id        001
3400000000004100    AX             Deny                   100
3700000000005100    AX             Deny                   100
3400000000008101    AX             Expired card           101
3700000000000101    AX             Expired card           101
3400000000005107    AX             Refer to card issuer   107

```

Figure 20 – Credit Card – Response Mapping File example

The file is text based in order to make it as understandable as possible and easy to update for the users.

To add a new credit card – response mapping, the user must write the credit card number, insert one or more tabulators and write the response code, all in the same line. In between the numbers, the user is free to add any information, as long as the credit card number is the first data in the line, the response number is the last and the data is separated by tabulators.

The Qantas AS2805 and the VISA ISO8583 standards should have response codes with two digits and the American Express ISO8583 standard should have three. If a number is too long (according to the response code size depicted in the MSD file), only the rightmost digits are used. If it is too short it is padded to the left with zeros.

If the credit card number, which is used in the transmission to the tool, is not found in the first tabulator separated segment of a line, the Response Action Function should be written so that the line is skipped and the next one investigated. In this way, the user is free to add whatever is found necessary in between the credit card – response code mapping lines. In the example in Figure 20, descriptive titles have been inserted which might be useful for the human reader of the file but will be ignored by the tool.

4.3.3 Credit Card Number Generator

When generating the response code, the Credit Card – Response Mapping File is a convenient tool for stating the response code to be generated. However, when a credit card number is received that is not represented in the file, some other way of determining the response code is necessary (given that the MSD file says it is to be generated). Since it is desirable to have the user remain in control of the outcome of the transmission, data transmitted by him/her needs to be used in the response creation. That is because this allows the user to emulate a response code without accessing the Credit Card – Response Mapping File.

The amount would be a possible alternative for a field used to determine the response code since it is always sent in a credit card request message. However, this field is not always under the user's control. If the call is made from a PNR²⁴, the prize of the purchase is automatically collected from the databases containing the information about the specific product requested.

All except one field sent in the request message similarly fails to satisfy the requirements for a response code determining field. The only data transmitted by the user that can be controlled in every case where the tool might be used is the account number.

Though the functionality of the response generating Response Action Functions is decided by the user, it is recommended for uniformity (and implemented in the prototype for all links) that the last digits of the credit card number is used as the response code. An American Express card with the account number 378291027851004 should therefore generate the response code (referred to as the Action code in the American Express terminology) 004 and a VISA card with the account number 4532247214000966 should generate the response code 66²⁵.

To be able to use the credit card number field as the field determining the response code in this manner, it is necessary to know which credit card numbers end with the desired response code and at the same time is a valid credit card number according to the LUHN formula²⁶.

To facilitate this, a credit card number generator was developed. The main parameters of this tool are the credit card type, the link on which it is to be tested and the response code to be generated (the end of the credit card number). From this information, using the LUHN formula, it generates one or more credit card numbers which can be used in

²⁴ See 1.3 - Problem definition

²⁵ The lengths of the response codes are determined by the MSD file and should correspond to the lengths stated in the credit card message specifications (see 2.1 - Credit Card Messages).

²⁶ See 2.2 - Credit card number validation

the credit card message communication with the Credit Card Validation and Authorization tool to generate the desired response message.

4.4 Design phase time allocation

The Credit Card Interface Validation and Authorization Emulation prototype, described in this chapter, was developed at Amadeus SAS during spring 2005. The development was divided in two segments. As the first part of the work, the solutions created by Sonrel for Receptor was implemented for TTServer. The scripts were adapted to fit into the newer tool and performance tests were made. The second part involved creating python scripts for dynamic handling of the incoming fields. The entire credit card message was cut out of the incoming EDIFACT message, sent to the scripts and parsed. Table L describes the main events of this work.

Table L – Main events of the development of the Credit Card Interface Validation and Authorization Emulation prototype

Date	Event
1 April	Created a version for VISA cards on the Qantas link that created the response based on the purchase amount.
4 April	Based the response on the credit card number with the help of the LUHN formula (since the amount was not a good enough value to set the response value with, I needed a way to create valid credit card numbers).
5 April	Created Credit Card – Response mapping file.
6 April	Finished version for the Qantas link with VISA cards based on hard coded message standard information.
7 April	Created the first, simple and text based, MSD file. Only included the name of the field, the size and the description.
11 April	Finished version for the Qantas link including both VISA and American Express cards.
15 April	Updated the format of the MSD file to include the response values (including the response bitmap) and data about whether the data is packed or in full bytes. Also included information on how each individual field is compressed (bin, packed or byte) and the type of values to be sent in the response (specified value, echo or generate).
18 April	Finished a version of the solution working for American Express and VISA cards on the Qantas link and American Express cards on the American Express link. This version was able to do everything stated in the project specification but was hard to manage.
21 April	Up until this date, the solution had been a simple function driven one. Restructured it into an object oriented version.
3 May	Finished an XML based version of the MSD files.
9 May	The tool was used for the first time. A developer needing to simulate credit card message responses tried it.
11 May	Implemented automatic message standard recognition. Up until this date,

	users had to update a path pointing to the MSD file describing the credit card message standard to be used. After this update, the tool itself could recognize which standard to use making it easier to use.
16 May	The implementation of the VISA link introduced a new way of describing variable field lengths ²⁷ . To handle this, the MSD file was updated with the key word "sizing".
27 May	Before this date, the PYTHONPATH environment variable had to be set for each user to point to the directory where the python scripts were located. This update made this unnecessary. The pointer to the directory was instead inserted in the code.
30 May	Updated the MSD files to describe the creation of response fields in a more generic way. Introduced the tag 'responseAction' containing the tags 'action', 'value' and 'respActionArgs'. In the code, a separate module containing the responseGenerator class was created, containing the Response Action Functions for generating response field values. After this update, any field in any credit card message standard can be generated dynamically in any way the user wants.

²⁷ See 2.1.3.4 - Data Fields

5 Prototype usage

5.1 Introduction

This chapter covers the usage of the emulator prototype created in the Credit Card Interface Validation and Authorization Project in the form of a user's guide. It describes how to set up the tool and send correct messages to it. It also describes how to add new credit card links to it or modify existing ones as well as how to make the tool generate response values for fields in different ways.

The emulator is in this chapter referred to as the Credit Card Interface Validation and Authorization (CCIVA) Emulator since the scope of the project was to implement it for credit card authorization messages. It should however be working just as well for any other kind of credit card messages as long as the settings are correctly made. This has however not been tested.

5.2 Typical usage

The CCIVA prototype is a TTServer tool that can be used for emulating credit card message communication from any implemented credit card company. It receives credit card authorization requests in the form of an EDIFACT HSFREQ message, extracts the credit card message inside it, reads it and creates the response according to the request message and the settings made in the tool.

There are two types of users of the tool. The common user will be using the tool for emulating responses and will be interested in making it generate certain types of responses. These users will find what they need in the chapters 5.3 - 5.6. The advanced user will be more of a maintainer of the tool. These users will be adding new links to it and will be able to generate the fields of the response message in any desired way. The chapters 5.7 and 5.8 describe how to do this.

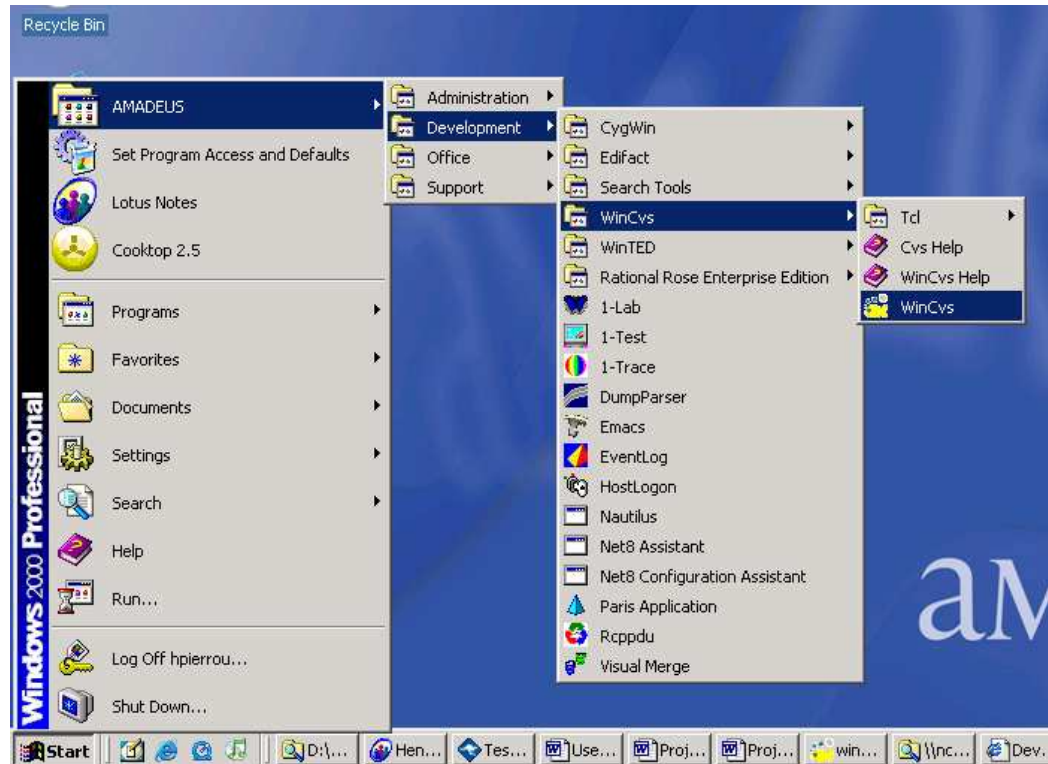
At Amadeus, during the work of developing of the prototype, the first users of the system were a developer in a Ticketing team, two members of a Product Definition team and a development coordinator.

In the first case, the tool was used during the development to constantly have correct and controllable emulated credit card companies to interact with in order to be able to create solutions for every specified scenario. In the second case the prototype was used to verify that newly written code behaved as specified in the communication with the credit card companies. The development coordinator acted as a maintainer of the tool and implemented a new credit card link by adding a new MSD file to the system which described the message standard used in the new link.

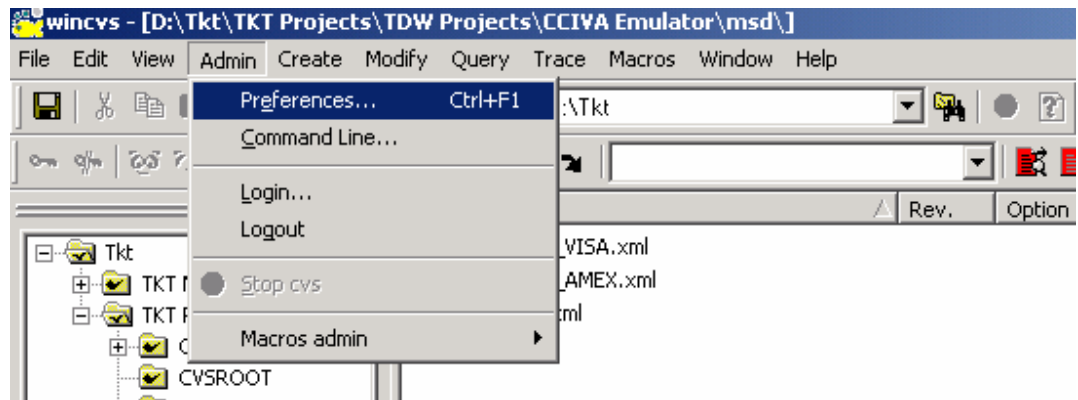
5.3 Downloading

To use the CCIVA Emulator, download the tool using WinCvs.

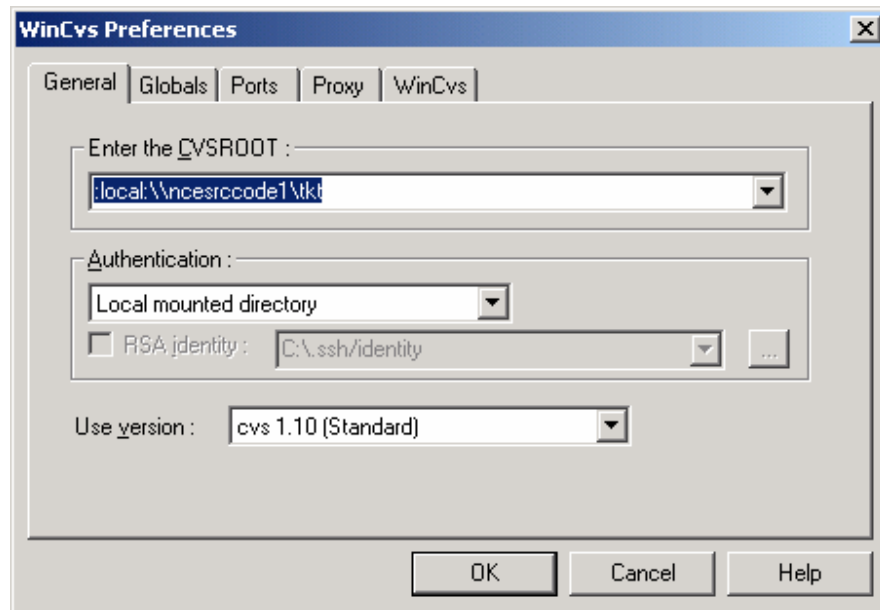
1. Start WinCvs by opening the start menu and clicking on AMADEUS – Development – WinCvs – WinCvs.



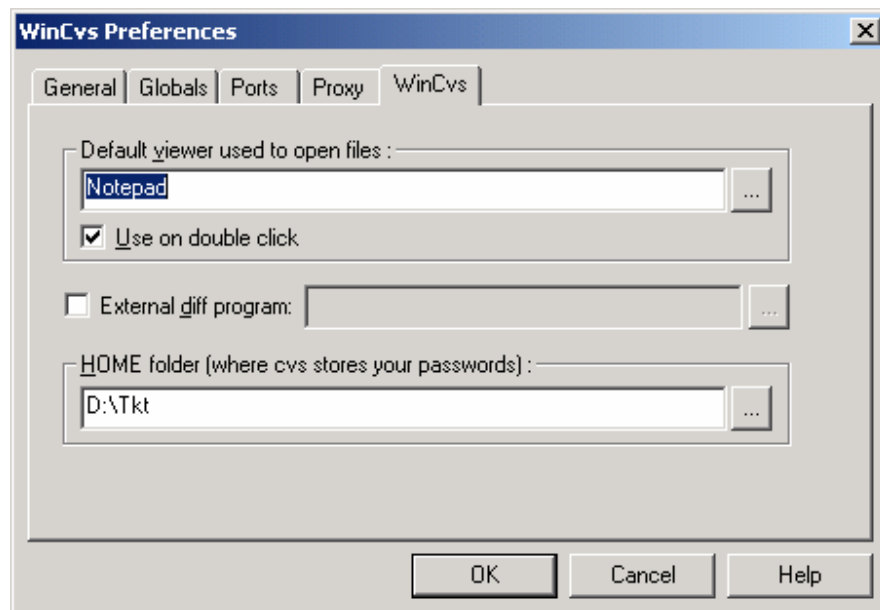
2. Open Admin – Preferences



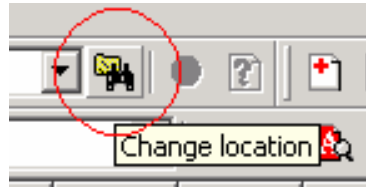
3. In the General Tab, add the CVS root “:local:\\ncesrcode1\tkt”.



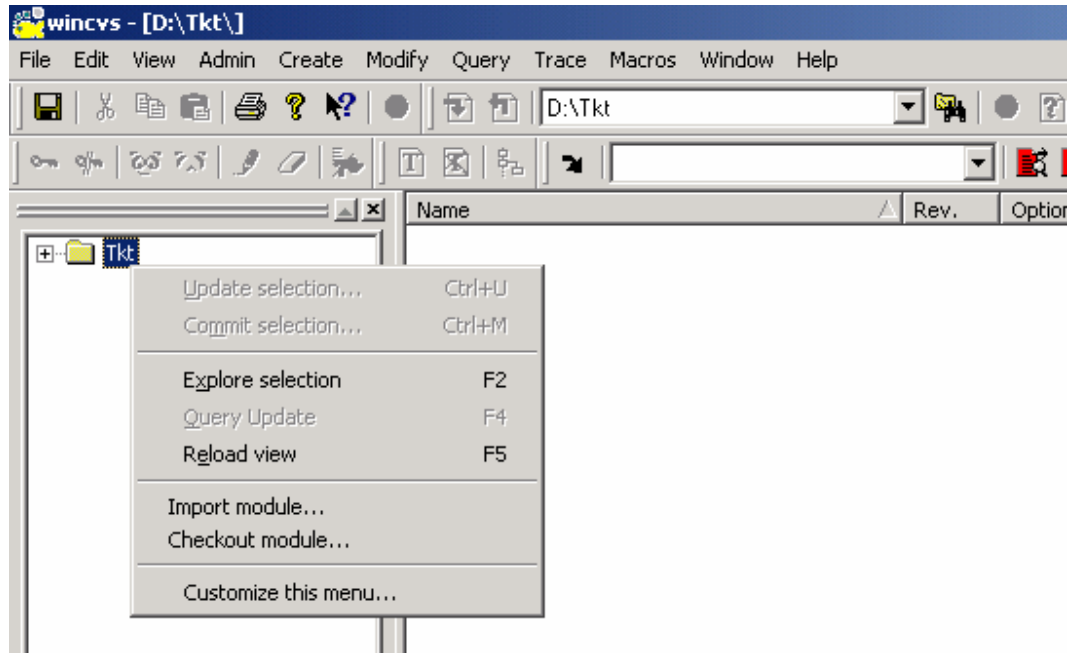
4. In the WinCvs tab, add a home folder of choice where the tool will be stored.



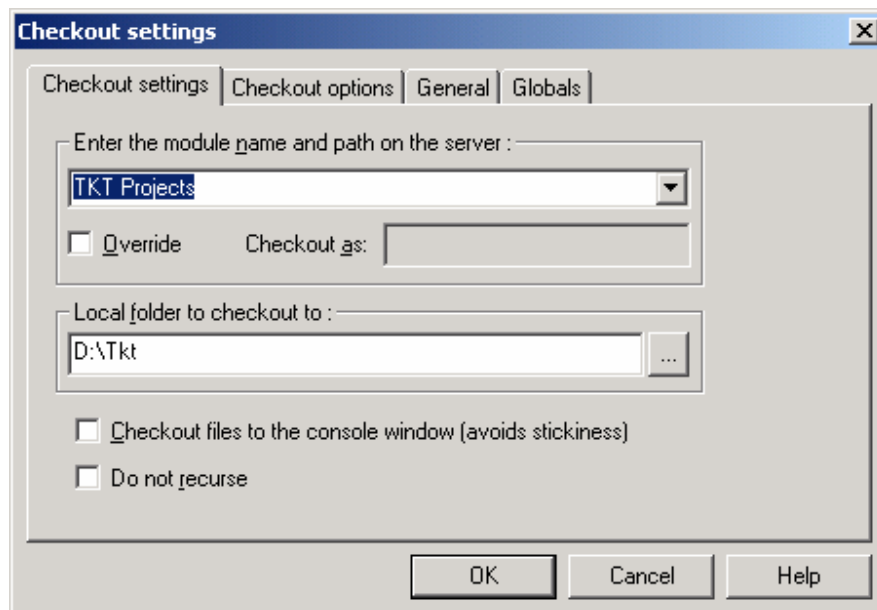
5. Click OK.
6. Click on the change location button. From the directory menu that appears, mark the directory you chose as your home folder in step 4 to make it appear on the left side of the screen.



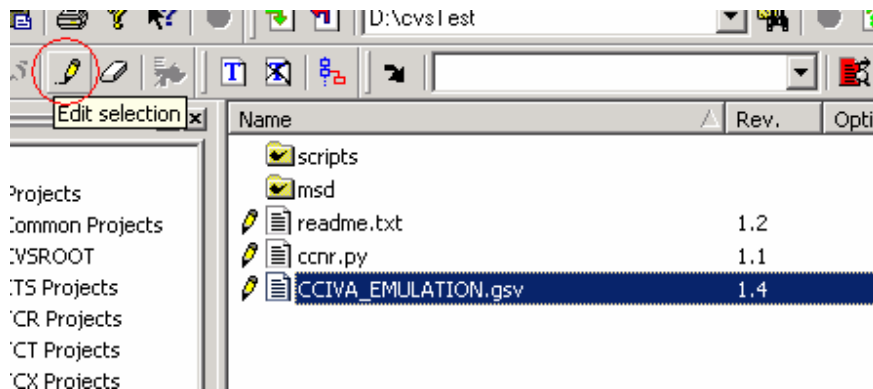
7. Right click the home folder and click Checkout module...



8. Enter the module name TKT Projects and let the home folder be the local folder to checkout to. Click OK.



9. The contents of the CVS repository under TKT Projects are downloaded to the local computer. This may take a while.
10. When the download is finished, reload the view to see the subdirectories. In the directories that appear, click on TKT Projects - TDW Projects - CCIVA Emulator. This is the root folder of the CCIVA Emulator. It should also be found in explorer by browsing the home directory from step 4 and clicking TKT Projects - TDW Projects - CCIVA Emulator.
11. Whenever a file needs to be updated (like the Credit Card – Response Mapping File²⁸) the file needs to be made writable. This is done by marking the file and clicking on the Edit Selection button.

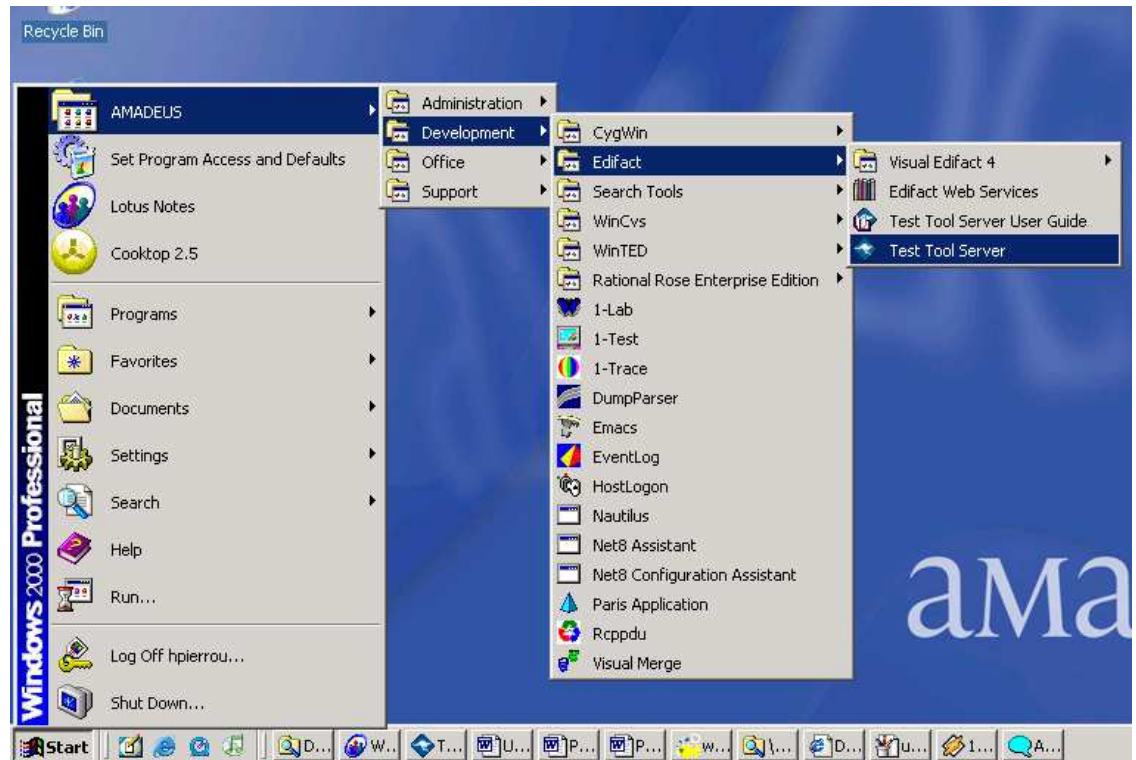


5.4 Running

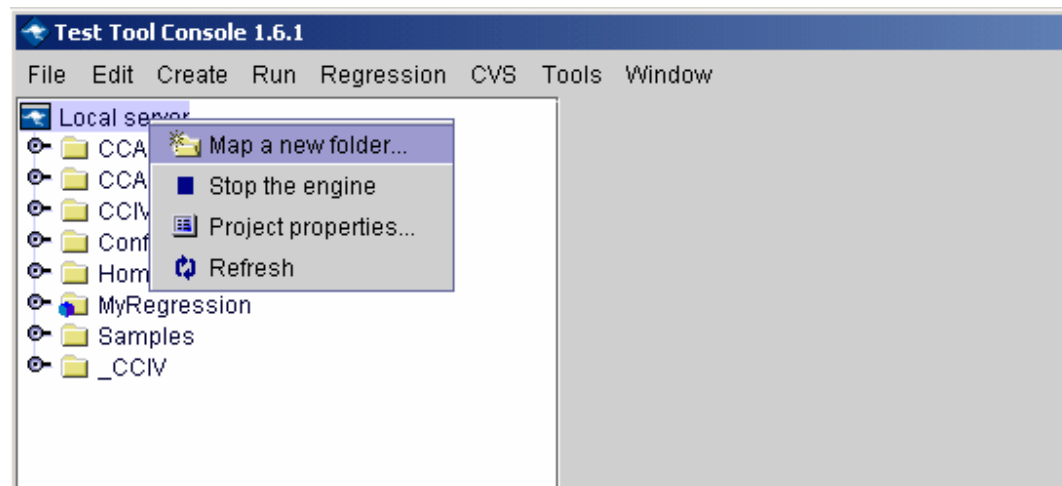
To run the tool, follow these steps:

1. The CCIVA Emulator is a TTSERVER application, so first open TTServer.

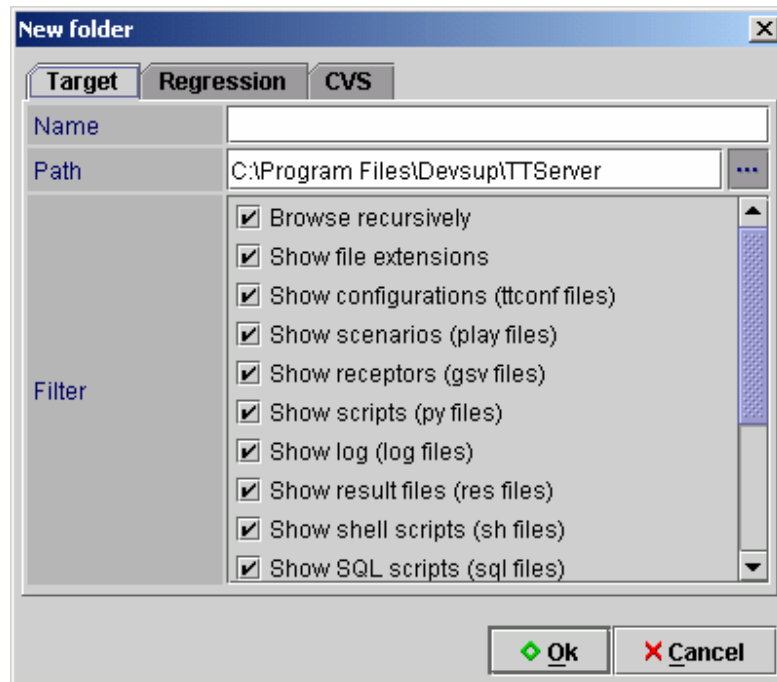
²⁸ See 5.6.2 - Using the Credit Card – Response Mapping File



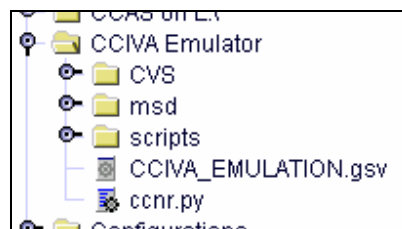
- From the directory view to the left, right click the Local server icon and click on Map a new folder.



- In the New folder window, click on the Path button (the button with three dots on it).

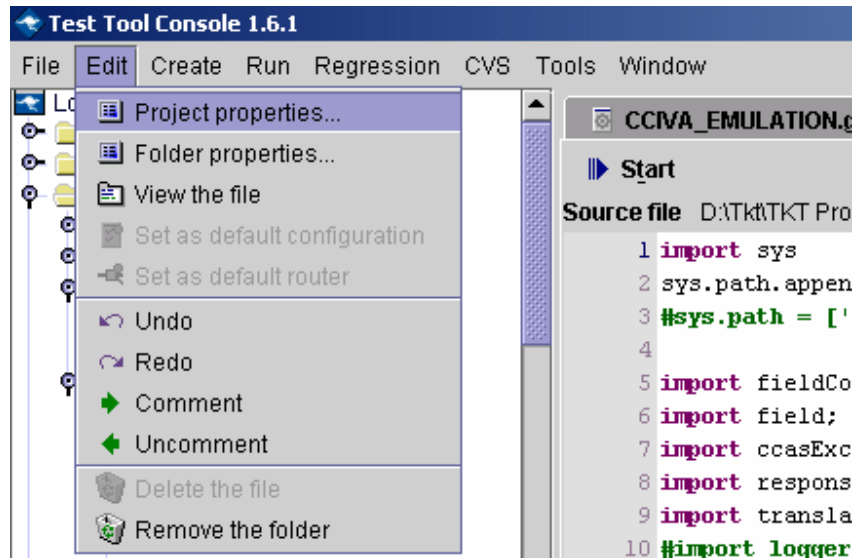


4. Locate the root folder of the CCIVA Emulator²⁹. Click Open.
5. In the New Folder window, enter a suiting name and click OK.
6. The folder should now be available in the directory view to the left. By clicking on the icon to the left of the folder, the sub folders get visible. The msd and scripts folders are required. The msd folder contains the Message Standard Description files describing the links that can be emulated and the scripts folder contains the scripts performing the emulations. The receptor scenario to run and to send the EDIFACT HSFREQ messages to is called CCIVA_EMULATION.gsv and is located in the root folder of the CCIVA Emulator.

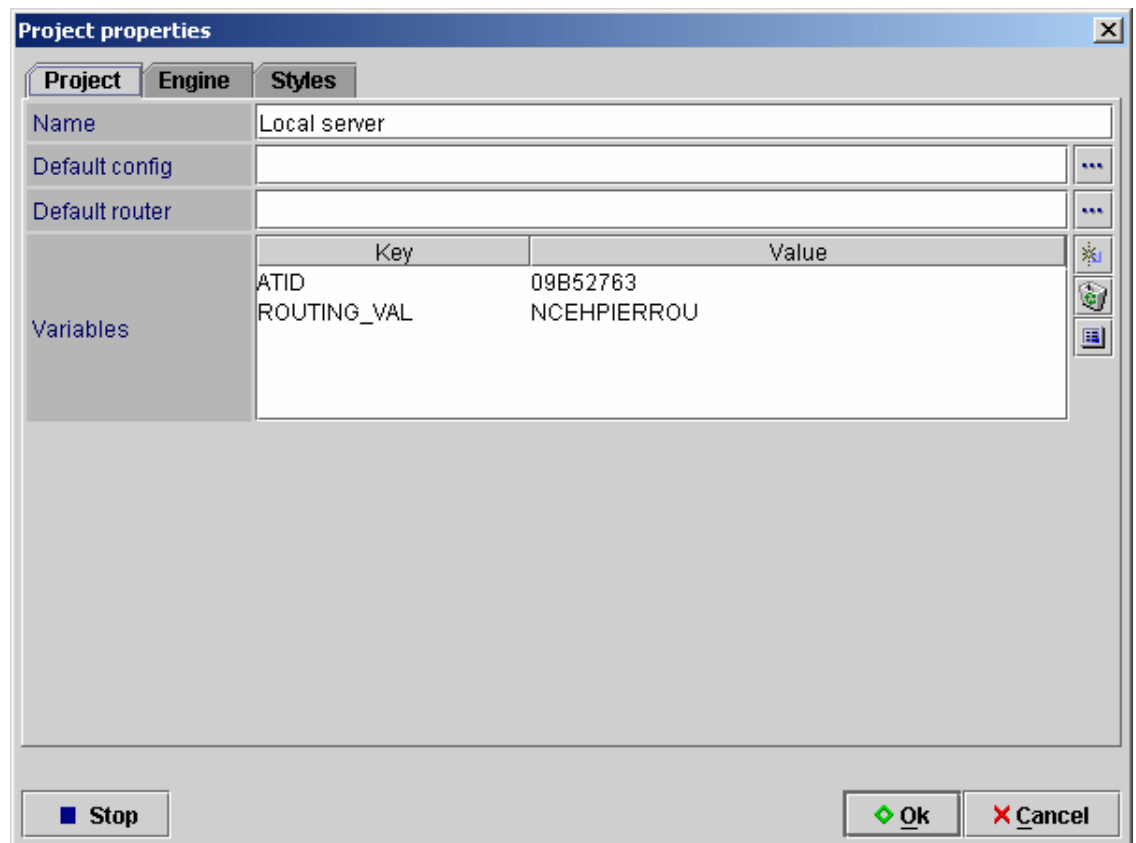


7. Click on Edit – Project properties...

²⁹ See chapter 5.3, step 10.

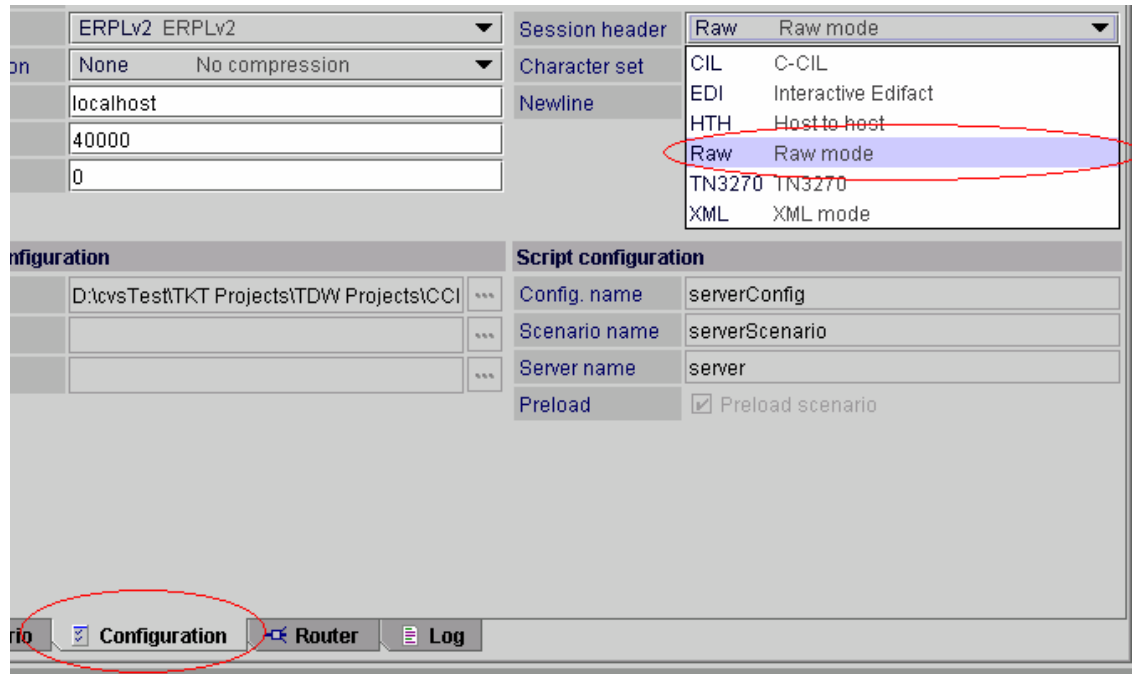


8. In the Project Properties window, add the variable ROUTING_VAL and the correct value. Also add the ATID variable if not already present.

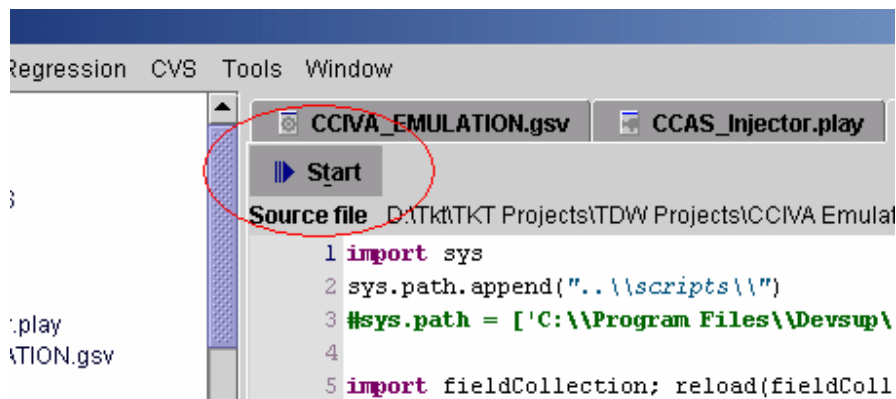


9. Click OK.
10. Open the scenarios folder.
11. Double click the CCIVA_EMULATION.gsv file to open it.

12. Make this file writeable in WinCvs³⁰.
13. Click on the Configuration tab beneath the code window and change the Session header drop down menu (not the Character set) to Raw mode. Save.



14. Click on the start button. This starts the emulator and makes it ready to receive HSFREQ credit card messages.

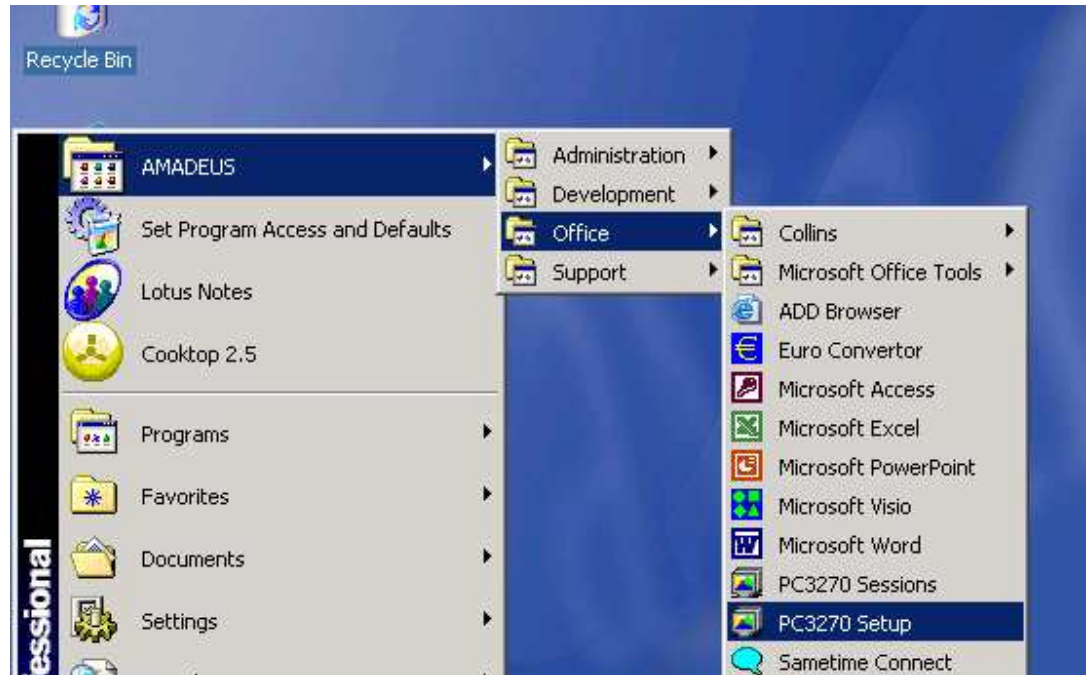


5.5 Sending and receiving messages

Messages are sent to the CCIVA emulator either from a PC3270 session or from a .play file in TTServer. The .play files are simply collections of the commands that are used in the PC3270 sessions so the following steps are generally applicable to both cases.

³⁰ See chapter 5.3, step 11

1. Open a PC3270 session (not applicable when sending messages from .play files).

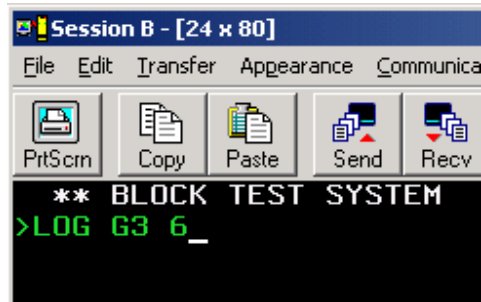


2. In session B, print codb and press the right ctrl button on the key board (not applicable when sending messages from .play files).

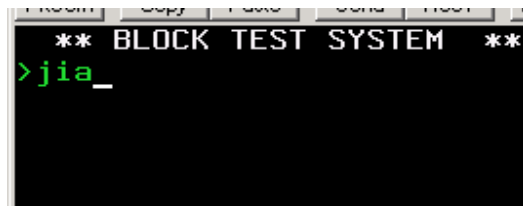
```
aaaaa MM MM aaaaa DDDDD
      Global Distribu

Enter Application Name (ex.: MUCTSO, CICS
=====> codb_
```

3. When LOGI COMPLETED is written on the screen, press the Pause button on the key board (not applicable when sending messages from .play files).
4. Choose back end by entering LOG G3 1 or LOG G3 6.



5. Log in using the JIA command.



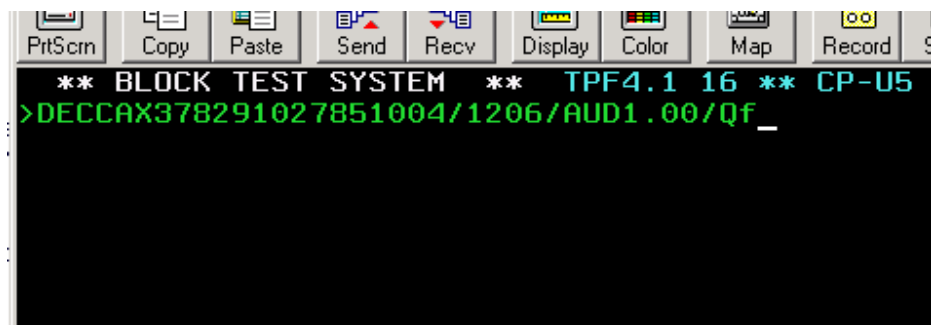
6. In order to route the messages to the TTServer session on the local computer and not to the real test links, print one or both of the following commands:

```
O*TTUG/ADD/EDTOOL/KEY1-%ATID%/KEY2-%UNTO%
O*AMCZ-EDTOOL-ON-AX-%UNTO%-%ATID%
```

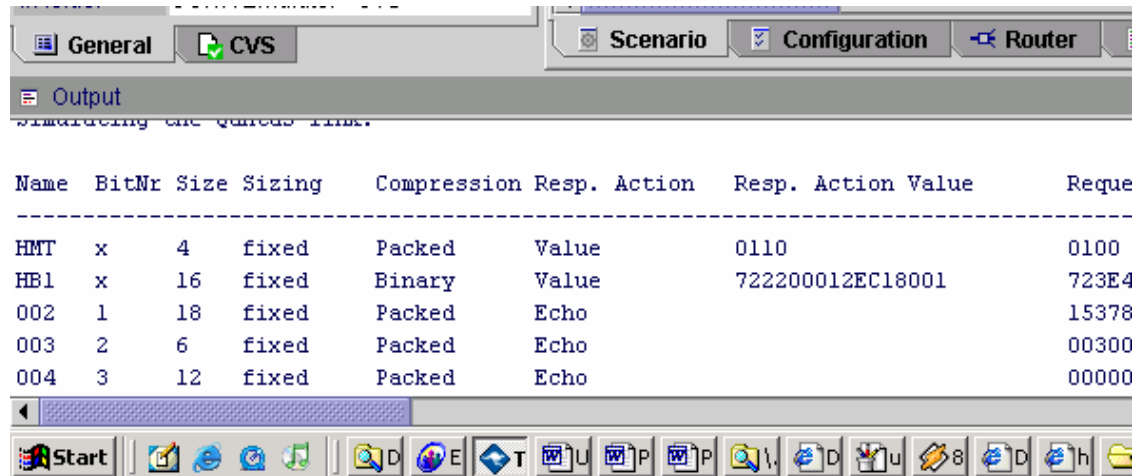
%ATID% should be the ATID of the computer using the tool and %UNTO% should be the UNTO corresponding to the link to be emulated.

Link	UNTO
American Express	AMEXDA
QANTAS	QF0CCP
VISA	VISADA

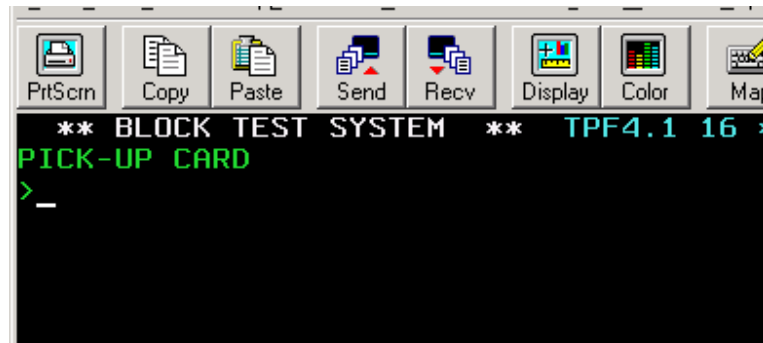
7. Make sure that the CCIVA_EMULATION.gsv file is running in TTServer (step 13 in chapter 5.4) and send the credit card request message.



8. In TTServer, the message is received and parsed and the response is created and sent back. The results of the message processing are displayed in the output window.



9. In the PC3270 Session, the response message corresponding to the response created by the CCIVA Emulator is printed.



5.6 Specifying responses

5.6.1 General

The response value of a field in a credit card message is specified in the MSD file. For the links implemented so far, the response code is set in the MSD files to be generated in Response Action Functions³¹. These use two ways to specify the response to be sent back from the CCIVA Emulator. The primary one is to map the credit card account number to the response code in the Credit Card – Response Mapping File³². The secondary is to use the last digits of the account number as the response code. In this case, the user can generate a credit card account number corresponding to the response code to be generated. Both cases are described below in chapters 5.6.2 and 5.6.3.

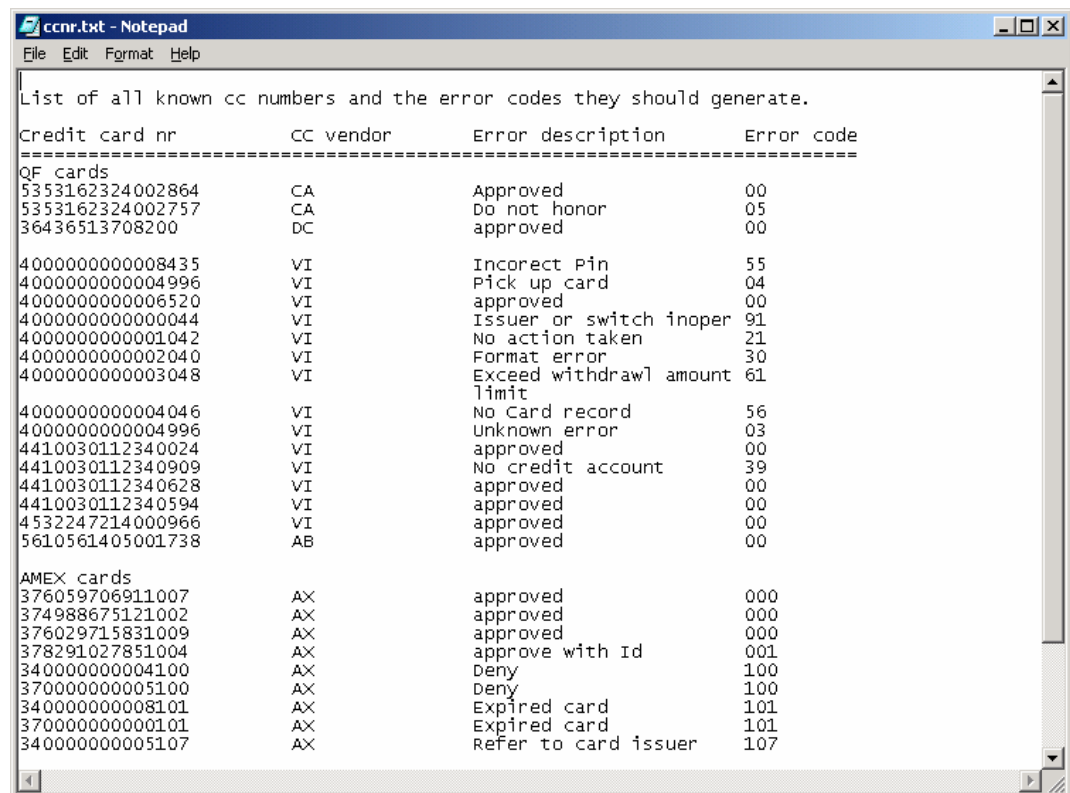
³¹ See 5.8 - Modifying response field creation, or the Project Report – Emulation Tool for Credit Card Interface Validation and Authorization.

³² For information on the Credit Card – Response Mapping File, see Project Report – Emulation Tool for Credit Card Interface Validation and Authorization.

5.6.2 Using the Credit Card – Response Mapping File

To use the Credit Card - Response Mapping File for specifying the response, follow these steps:

1. Open the root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3. The file is called ccnr.py. The .py extension is a result of TTServer's inability to perform commit, update and checkout on text files.
2. Remember to make the file writeable from WinCvs (as described in chapter 5.3, step 11) before updating it.



```

ccnr.txt - Notepad
File Edit Format Help
List of all known cc numbers and the error codes they should generate.
Credit card nr      CC vendor      Error description      Error code
-----
QF cards
5353162324002864    CA             Approved               00
5353162324002757    CA             Do not honor           05
36436513708200     DC             approved               00
4000000000008435    VI             Incorect Pin           55
4000000000004996    VI             Pick up card           04
4000000000006520    VI             approved               00
4000000000000044    VI             Issuer or switch inoper 91
4000000000001042    VI             No action taken        21
4000000000002040    VI             Format error            30
4000000000003048    VI             Exceed withdrawl amount 61
limit
4000000000004046    VI             No Card record         56
4000000000004996    VI             Unknown error          03
4410030112340024    VI             approved               00
4410030112340909    VI             No credit account      39
4410030112340628    VI             approved               00
4410030112340594    VI             approved               00
4532247214000966    VI             approved               00
5610561405001738    AB             approved               00
AMEX cards
376059706911007     AX             approved               000
374988675121002     AX             approved               000
376029715831009     AX             approved               000
378291027851004     AX             approve with Id        001
340000000004100     AX             Deny                   100
3700000000005100     AX             Deny                   100
3400000000008101     AX             Expired card           101
3700000000000101     AX             Expired card           101
3400000000005107     AX             Refer to card issuer   107

```

3. In the file, add the account number and the corresponding response code below the line with the = characters. The available response codes for a certain message standard can be found in the ICD's or specifications for that standard. The valid credit card numbers for a certain vendor (with the right bin number etc.) can be found in the credit card tables.

A few rules need to be followed:

- Each new credit card account number mapped to a response code should be written on a separate line.
- The credit card account number and the response code should be separated by tabulators.
- The account number should be the first data in the line and the response code the last.
- In between the account number and the response code as well as between lines, any descriptive data can be inserted.
- If a response code is longer than it is supposed to be for a certain link, the

rightmost digits are used.

- If a response code is shorter than it is supposed to be for a certain link, the response code is padded to the left with zeros.

4. Save the file and send a credit card message with the credit card account number just added to the file.

5.6.3 Generating a credit card account number

1. In TTServer, open the CCNrGen file from the scripts folder found in the root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3.
2. At the bottom of the file, locate the call of the function CCNrGen:

```
56 #  
57 CCNrGen("AX", "111", 1)  
58 #({VI/AX, Error code, Amount})  
59 #  
60
```

3. Update the arguments to the function CCNrGen to make it generate the desired result.
 - The first argument states which kind of credit card number to generate. American Express and VISA are supported. For American Express, use AX, for VISA use VI.
 - The second argument is the response code to generate, i.e. the last digits of the credit card number to generate.
 - The fourth argument is the number of credit card numbers to generate.
4. Use the credit card number generated in a message sent to the CCIVA Emulator. The response code will be equal to the last digits of the credit card account number. Remember however, that the Credit Card – Response Mapping File is the primary way of determining the response code. Therefore, if the credit card number is present in the file, the response code mapped to it will be used and not the last digits of the account number.

5.7 Updating/Adding links

The MSD file is an XML file describing all fields in a given message standard that are used by Amadeus (request messages as well as response messages etc.) To create a new link, a new Message Standard Description (MSD) file needs to be created. To update the definition of a link, the corresponding MSD file should be changed. The rules for doing this can be read in the Project Report – Emulation Tool for Credit Card Interface Validation and Authorization.

To create an MSD file follow these steps:

1. Create a new file in the msd folder found in root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3.

2. Give the file a name corresponding to the standard it is to describe.
3. Open the file and add the base tag <standard> and end it with </standard>.
4. For every field to be described, add a field description set containing the tags <field>, <size>, <compression>, <responseAction>, <action>, <value>, <respActionArgs> and <description>.

```
<field name="004" bmpNr="3">
  <size sizing="fixed">12</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Echo</action>
    <value></value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>Transaction Amount</description>
</field>
```

5. All fields in a message standard do not need to be described in the MSD file, only the ones to be used in the credit card message transmissions to and from the CCIVA Emulator tool.
6. The bitmaps should be named HB1 (primary) and HB2 (secondary).
7. The fields can be added in any order with two important exceptions. The primary bitmap must come before the secondary bitmap and the fields not described in the bitmap must come in the order in which they appear in the message. It is recommended however to add all fields in the order in which they appear in the credit card message.
8. The tags in the responseAction tag need only be filled with values if the field is part of the response.
9. If a new link is created, don't forget to add the UNTO in the table of UNTOs as specified in step 6 of chapter 5.5.

5.8 Modifying response field creation

5.8.1 General

There are three ways to specify what the response value for a certain field in a credit card message should be. One is to have a specific value inserted in the response, another is to have the field's value echoed from the request message into the response. The third is to have the value generated in certain user created functions called Response Action Functions.

5.8.2 Specific value or Echo

To make the response value for a certain field a specific value or to have the value echoed from the request message follow these steps:

1. Open the corresponding MSD file in the msd folder found in the root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3.
2. Locate the field to update or insert it if it does not exist.

3. To insert a specific value in the response field, insert **Value** in the **action** tag inside the **responseAction** tag. In the **value** tag, insert the value to be used in the response field. The **respActionArgs** tag should be left empty.

```
<field name="38" bmpNr="37">
  <size sizing="fixed">6</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Value</action>
    <value>F0F0F0F0F0F5</value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>System Trace Audit Number</description>
</field>
```

4. To echo the value from the request message, insert **Echo** in the **action** tag inside the **responseAction** tag. The **value** and **respActionArgs** tags should be left empty. Trying to make a field echo its value from the request message though the field is not part of the request message will raise an error.

```
<field name="38" bmpNr="37">
  <size sizing="fixed">6</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Echo</action>
    <value></value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>System Trace Audit Number</description>
</field>
```

5.8.3 Response Action Functions

If a response value needs to be generated dynamically, the CCIVA Emulator can be made to run a user defined Response Action Function to generate the value in the desired way. To make the tool do this, follow these steps:

1. Open the corresponding MSD file in the msd folder found in the root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3.
2. Locate the field to update or insert it if it does not exist.
3. Insert **Generate** in the **action** tag inside the **responseAction** tag.

```
<field name="39" bmpNr="38">
  <size sizing="fixed">3</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Generate</action>
    <value></value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>Action Code</description>
</field>
```

4. In the **value** tag, insert a key word of choice to be used later to map this field to a Response Action Function. The key word should be as descriptive as possible.

```
<field name="39" bmpNr="38">
  <size sizing="fixed">3</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Generate</action>
    <value>ISO8583_Amex_ActionCode</value>
    <respActionArgs></respActionArgs>
  </responseAction>
  <description>Action Code</description>
</field>
```

5. In the **respActionArgs** tag, insert any number of arg tags. The contents of these tags are accessible from the Response Action Function.

```
<field name="39" bmpNr="38">
  <size sizing="fixed">3</size>
  <compression>Byte</compression>
  <responseAction>
    <action>Generate</action>
    <value>ISO8583_Amex_ActionCode</value>
    <respActionArgs>
      <arg>002</arg>
      <arg>Another argument</arg>
    </respActionArgs>
  </responseAction>
  <description>Action Code</description>
</field>
```

6. From TTServer, open the responseGenerator.py file, located in the scripts folder found in the root folder of the CCIVA Emulator, specified in step 4 of chapter 5.3.
7. This file contains the **Response_Generator** class. The first function in this class is the **generate** function. This is where the key word from the **value** tag in the MSD file is mapped to a certain Response Action Function.

```
5 class Response_Generator(object):
6
7 #=====
8 # ADD NEW MAPPINGS BETWEEN MSD RESPONSE ACTIONS AND FUNCTIONS HERE #
9 #=====
10
11 def generate(self, action, args, fieldMatrix, responseLength):
12
13     self.args = args
14     self.fieldMatrix = fieldMatrix
15     self.responseLength = responseLength
16
17     if action == "ISO8583_Amex_ActionCode":
18         return self.iso8583_Amex_ActionCode()
19     if action == "AS2805_Qantas_RespCode":
20         return self.as2805_Qantas_RespCode()
21     if action == "ISO8583_Visa_RespCode":
22         return self.iso8583_Visa_RespCode()
23
```

8. If there is not already a mapping for the key word stated for the field in the MSD file, add one at the bottom of the function. This is done by inserting the following code with the KEY_WORD expression replaced with the key word from the MSD file and the RESPONSE_ACTION_FUNCTION expression replaced with a valid Response Action Function (created from step 10 and forward):

```
If action == KEY_WORD:
    Return self.RESPONSE_ACTION_FUNCTION()
```

9. Scroll down the file to the place where the Response Action Functions are located.

```

24 #=====
25 # ADD NEW RESPONSE ACTION FUNCTIONS HERE #
26 #=====
27
28 def iso8583_Amex_ActionCode(self):
29
30     returnVal = ""
31
32     # Getting credit card number
33     ccNumber = self.transl.numEbcDicTransl( self.fieldMatrix.getByNome
34
35     errNr = self.compToKnown(ccNumber[2:], self.responseLength)
36     if(errNr != ""):
37         for x in range(len(errNr)):
38             returnVal = returnVal + "F" + errNr[x]
39
40     else:
41         for x in range(-self.responseLength, 0):
42             returnVal = returnVal + "F" + ccNumber[x]
43
44     return returnVal

```

10. Insert a new Response Action Function with the name that was called in the mapping in the **generate** function. This is done by adding the following code (with **RESPONSE_ACTION_FUNCTION** replaced with the name of the function):

```
def RESPONSE_ACTION_FUNCTION(self):
```

11. Below this code, any code can be inserted to generate the desired response value for the field. The following variables are accessible to assist in the processing.

Variable	Type	Content
self.args	List	List containing the arguments passed from the MSD file's respActionArgs tag.
self.fieldMatrix	fieldCollection	Collection of all the fields and their attributes collected from the MSD file, plus their values in the request message.
self.responseLength	Integer	The length of the value to be returned.

12. In order to access the values of the fieldMatrix variable, use the **getByName** call.

```
self.fieldMatrix.getByNome(name, attribute)
```

13. The **getByName** call uses the following two arguments.

Argument	Info
----------	------

name	String. The name of the field to be investigated.
attribute	String. The attribute in the field to be returned. Can be one of the following: bitNr – The bitNr attribute value in the field tag. size – The size tag value sizing – The sizing attribute value in the size tag. compression – The compression tag value. responseAction – The action tag value. responseActionVal – the value tag value. requestValue – The value of this field in the request message. description – The description tag value.

14. In order to find the response code (if any) mapped to the used credit card number in the Credit Card – Response Mapping File, use the **compToKnown** call.

```
self.compToKnown(ccNumber, responseLength)
```

15. The **compToKnown** call uses the following two arguments.

Argument	Info
ccNumber	String. The credit card number to be looked for in the Credit Card – Response Mapping File.
responseLength	Integer. The length of the response code to be returned. Must be equal to the field length stated in the MSD file.

6 Analysis

6.1 Measurements

In order to verify that the requirements described in 2.6 - Requirements were fulfilled, various types of measurements and tests were made. These have been divided into the four areas performance, dynamicity, generality and reliability.

The performance measurements were aimed at proving that the tool is faster than the one created by Ludovic Sonrel, which was one of the requirements. The requirement of having the prototype be compatible with the Qantas, Visa/MasterCard, and American Express links and still generic enough to be applicable to other credit card company links, possibly added in the future, was targeted by the generality segment. The dynamicity part describes measures taken to make sure that the tool is dynamic, i.e. able to process credit card messages and return responses depending on the input data whereas the reliability measurements shows that the tool is more reliable than the test credit card links used prior to the emulation prototype.

6.2 Results

6.2.1 Performance

To verify that the prototype is faster than the tool created by Ludovic Sonrel in 2002, the transaction time was measured. Fifty credit card messages per implemented message standard (Qantas, American Express and VISA) were sent to the emulator and the corresponding responses were returned. The average required time for sending the messages, processing them and returning the responses are shown in Table M. Also depicted in this table and in the pie chart in Figure 21 is how the processing time is divided between the different parts of the credit card interface validation and authorization emulation process.

Table M – Measurements of the time required to send and process credit card messages as well as return the corresponding response with the emulator.

Link	Request Transmission	MSD file Detection	Request parsing	Response creation	Other (printouts etc)	Response Transmission	Total
Qantas	0.5853	0.1295	0.0027	0.0153	0.0516	0.6483	1.4327
AMEX	0.4893	0.0703	0.0027	0.0172	0.0523	0.5127	1.1445
VISA	0.5219	0.1291	0.0027	0.0158	0.0529	0.6157	1.3381
Average	0.5322	0.1096	0.0027	0.0161	0.0523	0.1807	1.3051

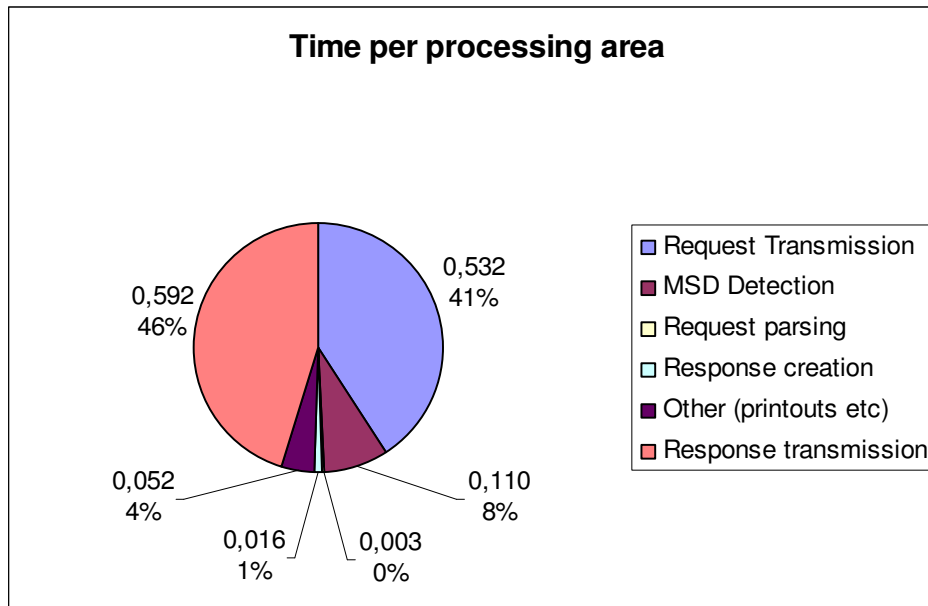


Figure 21 – Time per processing area

Evident from the table and figure above is that the bulk of the processing time is spent on transactions to or from the emulator. This part was left outside the scope of the project for reasons related to the inevitably complex structures created in systems of the Amadeus GDS systems' size and the time constraints of the project. Attempting to improve the EDIFACT message assembling, transaction protocols and mediums would require involving many more development teams and rethink standardized ways of working which are built in to practically every part of the company. If any improvements of the message transactions at Amadeus would need to be done, it is therefore a task to be dealt with by others, however interesting and challenging the task might be.

Despite this, the performance measurements must be considered satisfactory considering that the credit card interface validation emulator manages to handle the entire processing, including transaction times, in less than one and a half second. This is to be compared to the earlier tool from 2002³³ which could not process an entire credit card message before timing out after ten seconds.

6.2.2 Dynamicity

To achieve dynamicity, the emulator was made to parse the request messages into its fields and create the response depending on the contents of these and the information stored in the MSD files. By modifying the MSD file, the response can be altered and made dependent on the request message.

To verify that the dynamic features functioned as expected as well as to make sure that the responses were correct in case of other response creation methods (such as fixed value or echo³⁴), tests were made in which several different credit card messages from the standards used in the implemented links were sent to the prototype. The settings in

³³ See 2.5 - Previous work

³⁴ See 4.2.5 - responseAction

the MSD files were altered to make the emulator create the responses in different ways. As can be seen in Table N the tests were satisfactory. In all cases, the prototype returned the expected response message.

Table N – Tests of the prototype's ability to return correct responses in various cases

Link	Message nr ³⁵	Tested response action	Result
Qantas	1	Fixed value	Correct
Qantas	1	Echoed values	Correct
Qantas	1	Generated values	Correct
Qantas	2	Fixed value	Correct
Qantas	2	Echoed values	Correct
Qantas	2	Generated values	Correct
American Express	3	Fixed value	Correct
American Express	3	Echoed values	Correct
American Express	3	Generated values	Correct
American Express	4	Fixed value	Correct
American Express	4	Echoed values	Correct
American Express	4	Generated values	Correct
VISA	5	Fixed value	Correct
VISA	5	Echoed values	Correct
VISA	5	Generated values	Correct
VISA	6	Fixed value	Correct
VISA	6	Echoed values	Correct
VISA	6	Generated values	Correct

6.2.3 Generality

It was stated in the project specification that the tool had to be implemented for the Qantas, American Express and VISA links and made generic enough to be applicable to additional credit card company links that might be added in the future.

The tool was successfully implemented and tested for American Express, Qantas and VISA. The VISA link implementation however had to be based on a sample message from [4] and not messages traced from real transactions with the VISA test link. This is because the VISA link, planned to be implemented in the framework used in the development of the prototype in May 2005, had not yet been implemented at the end of the project period. As a result of this, no comparisons could be made between the performance of the real VISA test link and the emulated one. It also makes the VISA link

³⁵ The Message Nr only illustrates how different messages were used in the tests.

implementation in this project's emulation prototype dependent on the correctness of the used sample message and therefore possibly less reliable.

To satisfy the requirement of a solution generic enough to be applicable to new links, a number of measures were taken during development. For simplicity, all these measures were included in the MSD files which therefore are the most important components in this aspect. By creating a new such file, or modifying an existing one, the prototype can be informed of how to interpret the incoming message and create the response. As shown in previous chapters, this can be done statically by inserting fixed values into the response, or dynamically by echoing the request values into the response or creating them in Response Action Functions.

6.2.4 Reliability

The regression framework scripts, sending several credit card messages of different types to the prototype and the real test links, were run twice a day for two weeks. One message per possible response code and link were sent. The tests showed how reliable the links were in terms of accessibility. In Table O, the success rate of communicating the different credit card messages are shown.

Table O – Success rate of communicating the different credit card messages in the regression framework tests.

	Prototype		Real test links	
	AMEX	Qantas	AMEX	Qantas
2005-06-16, Run 1	Yes	Yes	No	Yes
2005-06-16, Run 2	Yes	Yes	No	Yes
2005-06-17, Run 1	Yes	Yes	No	No
2005-06-17, Run 2	Yes	Yes	No	No
2005-06-20, Run 1	Yes	Yes	Yes	No
2005-06-20, Run 2	Yes	Yes	Yes	No
2005-06-21, Run 1	No (coding error)	No (coding error)	Yes	Yes
2005-06-21, Run 2	Yes	Yes	Yes	Yes
2005-06-22, Run 1	Yes	Yes	Yes	Yes
2005-06-22, Run 2	Yes	Yes	Yes	Yes
2005-06-23, Run 1	Yes	Yes	No	No
2005-06-23, Run 2	Yes	Yes	No	No
2005-06-24, Run 1	Yes	Yes	Yes	Yes
2005-06-24, Run 2	Yes	Yes	Yes	Yes

Run 2				
2005-06-27, Run 1	Yes	Yes	No	Yes
2005-06-27, Run 2	Yes	Yes	No	Yes
2005-06-28, Run 1	Yes	Yes	No	Yes
2005-06-28, Run 2	Yes	Yes	Yes	Yes
2005-06-29, Run 1	Yes	Yes	Yes	No
2005-06-29, Run 2	Yes	Yes	Yes	No
2005-06-30, Run 1	Yes	Yes	No	No
2005-06-30, Run 2	Yes	Yes	No	No

7 Conclusions

The project covered in this report was aimed at resolving the issues, involved in credit card communication over test links, presented in chapter 1.3. It was a prerequisite to develop a prototype to answer the question of how these issues would best be solved. It was to be made generic enough to handle any credit card message format.

The idea was to create an emulator that could be used to send the credit card messages to, instead of the real test links. The emulator would receive the messages and process them in a way similar to that of the credit card company applications and return correct responses to the sending side. Since the processing of the request message would then be conducted locally within the company, the control over what is being done would be in the hands of the sender of the message. By making the emulator flexible, the response returned from the application could be created in exactly the way that the sender would want.

The prototype developed in the Credit Card Interface Validation and Authorization project was built in such a way and successfully put the control over the processing and response creation in the hands of the credit card message sender. This was achieved by combining the receptor script's functionality, to receive credit card messages translated into EDIFACT, with an application built in Python, to handle the processing of the message, and other, application-external resources developed in the project such as the MSD files.

By moving the processing from the credit card companies to a local location, the issue of accessibility was resolved. Instead of relying on external links for the communication, the only real link that needs to be up is the network connection between the sending application and the emulator which is internal and controllable.

The processing of the messages has also come under the control of the sender. The emulator's functionality can be easily modified by making changes to the MSD files and/or the Response Action Functions. If more detailed changes need to be done than can be achieved by updating these, the application, being located locally, can still be updated by modifying the Python source code³⁶.

With complete control over the processing, the issue of limited variation possible in the response messages is also resolved. By modifying the MSD files, the user can make the emulator return any value in any field of the response messages. This can be done statically by stating in the MSD file the exact value to be returned, or it can be done dynamically by either echoing the value from the request message or generating the value in a Response Action Function.

The work with the Credit Card Interface Validation and Authorization project at Amadeus has been interesting in many levels. The free hands that I was given gave me useful experiences in managing a project from start to finish and to present and defend my ideas to my industrial supervisor and other Amadeus employees with various degrees of experience in this field.

³⁶ This is not the recommended way to go but illustrates how the sender using the prototype holds the complete control over the processing.

The project also gave important insights when it comes to how the architecture of systems like the CCIVA system ought to be planned. My lack of experience as an architect sometimes became evident when parts of the code had to be rewritten in order to become more efficient or to facilitate some communication process between two objects. These mistakes are not likely to be remade since the delayed they caused was quite distressing.

8 Future work

8.1 Handling of true binary data

As described in chapter 2.4.3 - HSFREQ/HSFRES, the original credit card message is translated before it is used in the HSFREQ EDIFACT message in order to make it suited for the TTServer environment. This translation however means moving away from the ideal emulation, resembling the real message passing to as large a degree as possible. If TTServer in the future is updated to be able to handle true binary data it would therefore be desired to make the emulator able to do so as well.

8.2 Testing and modification of VISA link

As mentioned, the VISA link was implemented based on a sample message from [4] and not messages traced from real transactions with the VISA test link because the VISA link, planned to be implemented in May 2005 in the framework used in the development of the prototype, had not yet been implemented at the end of the project period. When this has been done, the VISA link implementation would need to be tested properly and possible errors be corrected.

9 References

Internal Amadeus documents³⁷

1. L. Sonrel. 2002. *Receptor tool for credit card authorization message*. Sophia-Antipolis: Amadeus SAS.
 2. L. Sonrel. 2002. *Qantas Gateway Link Interface Control Document*. Sophia-Antipolis: Amadeus SAS.
 3. R. Humphries. 2000. *Amex Direct Link Interface Control Document*. Sophia-Antipolis: Amadeus SAS.
 4. R. Humphries. 2000. *Visa Direct Link Interface Control Document*. Sophia-Antipolis: Amadeus SAS.
 5. E. Peran. 2005. *Test Tool Server User Guide*. Sophia-Antipolis: Amadeus SAS.
 6. E. Peran. 2003. *Edifact Training*. Sophia-Antipolis: Amadeus SAS
-

Books

7. 2002. *BASE I Technical Specification Volume 1 VISA V.I.P. System*, VisaNet Technical Documentation
 8. 2002. *BASE I Technical Specification Volume 2 VISA V.I.P. System*, VisaNet Technical Documentation
 9. M. Lutz. 1996. *Programming Python*. ISBN X. X:O'Reilly
-

Web pages

10. *Python 2.3.5 Documentation*. < <http://www.python.org/doc/2.3.5/> >. Released February 8th, 2005.
 11. *VIRADIX Terminology* < <http://www.viradix.com/terminology.html#e> >. Accessed May, 2005.
 12. *XML Glossary* < <http://www.softwareag.com/xml/about/glossary.htm> >. Accessed May, 2005.
 13. *GIRO Bankcard Ltd. – Glossary of the bankcard industry* - < <http://www.gbc.hu/english/bszotare2.htm> >. Accessed May, 2005.
-

³⁷ All internal Amadeus documents are referred to with the approval of Mr. Nazir Goulamhousen, Unit Manager of the Ticketing team, Global Core, Amadeus.

-
14. *ISO9735*. < <http://www.edifactory.de/ISO9735.TXT> > Accessed May, 2005
-
15. *UN/EDIFACT Working Group (EWG)* < http://www.unece.org/trade/untdid/sessdocs/ewg_0598.htm >. Accessed May, 2005
-
16. *ISO 9735 EDIFACT*. < <http://www.nls.fi/ptk/standardisation/2.html> >. Accessed May, 2005
-
17. *Opera, the Amadeus Intranet*. < <https://opera.amadeus.com/intranet/html/index.jsp> >. Accessed May, 2005
-
18. H. Stiles. *Credit Card Check Digit Validation* < <http://www.beachnet.com/~hstiles/cardtype.html> >. Accessed June, 2005.
-

Articles

19. J. A. Whittaker. 2000. What Is Software Testing? And Why Is It So Hard?. *IEEE Software*: 70-79
-

Appendix A - Message standard specifications

A.1 Qantas AS2805

The field with bit number 1 (the first bit, in other contexts usually referred to as bit 0) is the secondary bitmap. Other fields omitted in Table A1, describing the fields of the Qantas AS2805 standard, are fields that are not used by Amadeus.

Table A1 – Qantas AS2805 credit card message fields used by Amadeus

Bit Nr	Field Name	Attribute	Size	Content
2	Primary Account Number	Numeric Maximum 19 digits	Variable	Credit card number information The first byte holds the number length, as two numeric digits. The rest is the card number. If of odd length it is padded to the right with four bits set to one. Example: x'164000111100001111' – 16 digits long number, value: 4000111100001111
3	Processing Code	Numeric 6 digits	3 Bytes	Information on what kind of request is being made The first two digits: 00 for purchase 20 for refund Digits three and four: 30 for credit account The Last two digits Always 00 Example: x'003000' – Purchase x'203000' – Refund
4	Transaction Amount	Numeric 12 digits	6 Bytes	The requested amount of money Leading zeros The decimal point position is implied by the currency code (field 49) Example: x'000000050000' - AUD\$500.00
7	Transmission Date & Time	Numeric 10 digits	5 Bytes	The date of the transmission Syd/Mel system time Stated in the form MMDDhhmmss Example:

				x'0124130404' - 13:04:04, January 24 th
11	System Trace Audit Number	Numeric 6 digits	3 Bytes	Unique system generated number to identify each transaction
12	Local Transaction Time	Numeric 6 digits	3 Bytes	The local time of the transaction Stated in the form Hhmmss GMT time Example: x'130404' – 13:04:04
13	Local Transaction Date	Numeric 4 digits	2 Bytes	The local date of the transaction Stated in the form MMDD GMT date Example: x'0124' – January 24 th
14	Expiry Date	Numeric 4 digits	2 Bytes	The expiry date of the credit card Stated in the form YYMM Example: x'0205' - May 02
15	Settlement Date	Numeric 4 digits	2 Bytes	Date when funds will be credited to Qantas by the acquirer Stated in the form MMDD Default is present date Example: x'0124' – January 24 th
18	Merchant's type	Numeric 4 digits	2 Bytes	Code describing the merchant's type of business product or service.
22	POS Entry Mode	Numeric 3 digits	2 Bytes	Identifies the method used to enter the credit card number (POS – Point Of Sale) First two digits: 01 manual 02 magnetic strip 05 integrated circuit card n b90 full and unaltered magnetic stripe read and transmitted Third digit: 2 No PIN entry capability
25	POS Condition	Numeric	1 Byte	Information on the device used at POS

	Code	2 digits		00 normal (customer present) 04 electronic cash register 07 telephone device 08 mail/telephone order 16 administration terminal 44 travel ticket vending machine 48 Electronic Commerce (Internet)
32	Acquiring Institution Identification Code	Numeric Maximum 11 digits	Variable	The financial institution for Qantas to debit Always x'0856022004'
35	Track 2 Data	Packed Track data Maximum 37 values	Variable	Swipe card data The first byte holds the remaining field length.
37	Retrieval Reference Number	Alpha Numeric ECBDIC 12 char.	12 Bytes	Message identifier used to match a request to a response Byte 1-4 - Julian date (YDDD) Byte 5-6 - Transaction hour, from Field 7 (hh) Byte 7-12 - Trace number, from Field 11 Example: c'102413002345'
38	Authorisation Identification Response	Alpha Numeric ECBDIC 6 char.	6 Bytes	Authorization Response identifier Any value
39	Response Code	Alpha Numeric ECBDIC 2 char.	2 Bytes	The response value. Authorization or denial and reason ³⁸ .
41	Card Acceptor Terminal ID	Alpha Numeric ECBDIC 8 char.	8 Bytes	Identifies the acceptor terminal (the receiver and acceptor of the credit card requests) Always x'D8C1D5E3C1E24040' (c'QANTAS ') with trailing blanks to 8 bytes
42	Card Acceptor Identification Code	Alpha Numeric ECBDIC 15 char.	15 Bytes	The credit card vendor code
43	Card Acceptor Name Location	Alpha Numeric ECBDIC	40 Bytes	Name of the credit card acceptor First characters: c'QANTAS' + space + agent's city code + spaces

³⁸ See 2.1.1.5 - Response codes

		40 char.		Last two digits: country code Example: c'QANTAS QSC AU'
48	Additional Data - Private	Alpha Numeric ECBDIC 999 char.	Variable	Additional data
49	Transaction Currency Code	Numeric 3 digits	2 Bytes	The transaction currency Leading 0
64	Message Authentication Code	Binary 64 bits	8 Bytes	Not used, filled with zeros
90	Original Data Elements	Numeric 42 digits	21 Bytes	Position 1- 4 = original message type Position 5-10 = original STAN (bit 11) position 11-20 = original transaction date and time (bit 7) Position 21-31 = original acquiring institution ID (bit 19) x'00056022004' Position 32-42 = original forwarding institution ID (bit 33) x'00000000000'
128	Message Authentication Code	Binary 64 digits	8 Bytes	Not used, filled with zeros

Numeric fields are stored as unsigned packed, i.e. one byte contains two values. A byte containing x'16 (bin 0001 0110) holds the numbers 1 and 6.

A.2 American Express ISO8583

The data fields contain the information to be sent to the credit card issuer. The field with bit number 1 (the first bit, in other contexts usually referred to as bit 0) is the secondary bitmap. Other fields omitted in Table A2, describing the fields of the American Express ISO8583 standard, are fields that are not used by Amadeus.

Table A2 – American Express ISO8583 credit card message fields, used by Amadeus

Bit Nr	Field Name	Attribute	Size	Content
2	Primary Account Number	EBCDIC (Numeric) Maximum 17 digits	Variable	Credit card number information The first two bytes holds the number length, as EBCDIC characters. The rest is the card number. Example: x' F1F5F3F7F8F2F9F1F0F2F7F8F5F1F0F0F4'

				– 15 digits long number, value: 378291027851004
3	Processing Code	EBCDIC (Numeric) 6 digits	6 Bytes	Information on what kind of request is being made 000000 – authorization only 004000 – authorization and address verification
4	Transaction Amount	EBCDIC (Numeric) 12 digits	12 Bytes	The requested amount of money Leading zeros
11	System Trace Audit Number	EBCDIC (Numeric) 6 digits	6 Bytes	Unique system generated number to identify each transaction
12	Date and Time, Local Transaction	EBCDIC (Numeric) 12 digits	12 Bytes	The local time of the transaction Stated in the form YYMMDDhhmmss
14	Expiration Date	EBCDIC (Numeric) 4 digits	4 Bytes	The expiry date of the credit card Stated in the form YYYY
22	POS Data Code	EBCDIC (Alpha-numeric) 12 digits	12 Bytes	Identifies the method used to enter the credit card number, e.g. manual, magnetic stripe, card-holder present (POS – Point Of Sale).
24	Function Code	EBCDIC (Numeric) 3 digits	3 Bytes	Message purpose. Used in network management messages (1804/1814 – echo test) ³⁹ .
26	Card Acceptor Business Code	EBCDIC (Numeric) 4 digits	4 Bytes	A code that identifies the type of business conducted by the service establishment (Amadeus).
27	Approval Code Length	EBCDIC (Numeric) 1 digits	1 Byte	The maximum length of the approval code that can be displayed or printed.
32	Acquiring Institution ID	EBCDIC (Numeric) Maximum 13 digits	Variable	Identifies the acquirer (Amadeus). The first two bytes specifies the number of digits, as EBCDIC characters. The rest is ID code.
37	Retrieval Reference	EBCDIC (Alpha-	12 Bytes	Message identifier used to match a request to a response

³⁹ See 2.1.2.1 - Message Type ID

	Number	numeric) 12 digits		
38	Approval Code	EBCDIC (Alpha-numeric) 6 digits	6 Bytes	The authorization code for the credit approval. Spaces indicates that approval is not given.
39	Action Code	EBCDIC (Numeric) 3 digits	3 Bytes	Response to the authorization. Holds the error code or action to be taken.
41	Card Acceptor ID	EBCDIC 15 digits	8 Bytes	The identification of the terminal at the card acceptor location
42	Card Acceptor ID Code	EBCDIC (Alpha-Numeric) 15 digits	15 Bytes	code that identifies the card acceptor (sales establishment).
44	Additional Response Data	EBCDIC Maximum 25 digits	Variable	Miscellaneous data needed in a response, such as Address Verification Result Code and Telecode Verification Result Code. The first two bytes specifies the number of digits, as EBCDIC characters. The rest is the additional response data.
49	Currency Code	EBCDIC (Alpha-Numeric) 3 digits	3 Bytes	The transaction currency
63	Private Use Data	EBCDIC Maximum 103 digits	Variable	Contains address details when address verification is requested. The first three bytes specifies the number of digits, as EBCDIC characters. The rest is the private use data.
93	Transaction Destination Institution ID	EBCDIC Maximum 11 digits	Variable	Identifies the destination institution (system). The first two bytes specifies the number of digits, as EBCDIC characters. The rest is the ID code.
94	Transaction Originator Institution ID	EBCDIC Maximum 11 digits	Variable	Identifies the originating institution (system). The first two bytes specifies the number of digits, as EBCDIC characters. The rest is the ID code.

--	--	--	--	--

A.3 VISA ISO8583

The data fields contain the information to be sent to the credit card issuer. The field with bit number 1 (the first bit, in other contexts usually referred to as bit 0) is the secondary bitmap. Other fields omitted in Table A3 are fields that are not used by Amadeus.

Table A3 – VISA ISO8583 credit card message fields used by Amadeus

Bit Nr	Field Name	Attribute	Size	Content
2	Primary Account Number	Numeric Maximum 11 digits	Variable	Credit card number information The first byte holds the number length, as binary value. The rest is the card number. If the account number has an odd number of digits, a leading zero is used to pad the first half-byte. This is not included in the count of digits Example: x' 104972053263029923 ' – 16 digits long number, value: 4972053263029923
3	Processing Code	Numeric 6 digits	3 Bytes	Coding which identifies the customer and account types.
4	Transaction Amount	Numeric 12 digits	6 Bytes	The requested amount of money Leading zeros
6	Cardholder Billing Amount	Numeric 12 digits	6 Bytes	Multicurrency field. Transaction amount ⁴⁰ converted to the currency used to bill the cardholder's account.
7	Transmission Date and Time	Numeric 10 digits	5 Bytes	The date and time in Greenwich mean time when the request or advice entered VisaNet.
10	Conversation Rate, Cardholder Billing	Numeric 8 digits	4 Bytes	Conversion factor used by Visa to calculate the billing amount ⁴¹ from the transaction amount ⁴² .
11	System Trace Audit Number	Numeric 6 digits	3 Bytes	Unique system generated number to identify each transaction
14	Expiration Date	Numeric	2 Bytes	The expiry date of the credit card

⁴⁰ See field 4 – Transaction Amount

⁴¹ See field 6 – Card Holder Billing Amount

⁴² See field 4 – Transaction Amount

		4 digits		Stated in the form YYMM
18	Merchant Type	Numeric 4 digits	2 Bytes	Code describing the merchant's type of business product or service.
19	Acquiring Inst. Country Code	Numeric 3 digits	2 Bytes	Identifies the country of the acquiring institution for the merchant or ATM. Leading zero
22	POS Entry Mode Code	Numeric 4 digits	2 Bytes	Identifies the method used to enter the credit card number, e.g. manual, magnetic stripe, card-holder present.
25	POS Condition Code	Numeric 2 digits	1 Bytes	A code that identifies the condition under which the transaction takes place at the point of service e.g. customer not present, mail/telephone order etc.
32	Acquiring Institution ID	Numeric Maximum 11 digits	Variable	Identifies the acquirer (Amadeus). The first byte holds the field length, as a binary value. The rest is the ID number. If the ID has an odd number of digits, a leading zero is used to pad the first half-byte. This is not included in the count of digits.
37	Retrieval Reference Number	EBCDIC (Alpha-numeric) 12 digits	12 Bytes	Message identifier used to match a request to a response
38	Authorization ID response	EBCDIC (Alpha-numeric) 6 digits	6 Bytes	The authorization code for the credit approval.
39	Response Code	EBCDIC (Alpha-numeric) 2 digits	2 Bytes	A code that defines the response to a request e.g. successful, do not honor, pick up card, invalid amount etc ⁴³ .
41	Card Acceptor Terminal ID	EBCDIC (Alpha-numeric) 8 digits	8 Bytes	The identification of the terminal at the card acceptor location
42	Card Acceptor ID Code	EBCDIC (Alpha-Numeric) 15 digits	15 Bytes	Code identifying the card acceptor (sales establishment).

⁴³ See 2.1.3.5 - Response Codes

43	Card Acceptor Name/Location	EBCDIC (Alpha-Numeric) 40 digits	40 Bytes	The name and location of the card acceptor.
44	Additional Response Data	EBCDIC (Alpha-Numeric) Maximum 25 digits	Variable	Miscellaneous data needed in a response, such as Address Verification Result Code and Telecode Verification Result Code. The first byte holds the field length, as a binary value. The rest is the data.
48	Additional Private Data	EBCDIC (Alpha-Numeric) Maximum 256 digits	Variable	Miscellaneous information. 16 different usages. The first byte holds the field length, as a binary value. The rest is the data.
49	Currency Code	Numeric 3 digits	2 Bytes	The transaction currency
51	Currency Code, Cardholder Billing	Numeric 3 digits	2 Bytes	3 digit numeric code identifying the billing currency (multicurrency).
59	National POS Geographic Data	EBCDIC (Alpha-Numeric) Maximum 14 digits (+ size value)	Variable	Geographical location data e.g. state, country. The first byte holds the field length, as a binary value. The rest is the data.
60	Additional POS Info	Numeric Maximum 10 digits (+ size value)	Variable	VISA private use. The first byte holds the field length, as a binary value. The rest is the data.
62	CPS Fields	Numeric Bit-mapped	Variable	VISA private use. The first byte holds the field length, as a binary value. The rest is the data.
70	Network Management Information Code	Numeric 3 digits	2 Bytes	Defines the type of Network Management needed.
90	Original Data Elements	Numeric 42 digits	21 Bytes	Contains selected data from the original message (identifies the message being reversed in reversal messages).

95	Replacement Amounts	EBCDIC (Alpha-Numeric) 42 digits	42 Bytes	The corrected amount of a transaction in a partial reversal.
123	Address Verification Data	EBCDIC (Alpha-Numeric) Maximum 256 digits	Variable	VISA private use, address information. The first byte holds the field length, as a binary value. The rest is the data.

Appendix B – Acronyms and Abbreviations

ATID	Amadeus Terminal ID
BMI	British Midland
CPS	Custom Payment Service
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport
EWG	UN/EDIFACT Working Group
GAB	Global acquiring Bank
GDS	Global Distribution System
IATA/ATA	International Air Transport Association/Air Transport Association America
IGW	Interface Gateway
MSD	Message Standard Description
PNR	Passenger Name Record
POS	Point Of Service
RBoS	Royal Bank of Scotland
SAA	South African Airlines
TPF	Transaction Processing Facility
TT&L	Travel Tourism and Leisure
TTServer	Test Tool Server
UN	United Nations
UN/ECE	United Nations Economic Commission
WE/EB	Western European EDIFACT Board
XML	Extensive Markup Language