



beyond
payment

TML Tutorial



V3.0

Service Business Unit

Tuesday, 4 August 2009

Contents

<i>Chapter 1: About this document</i>	5
Reading prerequisites	5
Related documents	5
Conventions.....	5
Document structure.....	6
<i>Chapter 2: Application selection menu</i>	7
Purpose, constituents and their locations.....	7
Application selection menu access mechanism.....	8
Implementation for Ingenico 5100 (the file <i>5100\index.tml</i>)	9
<i>Chapter 3: Four simple coding exercises</i>	11
Overview of the exercises.....	11
Prerequisites	11
Commands you are going to use	12
If you quit the exercises before completing them.....	12
A note for HTML developers	13
Exercise 0: Designing a screen and planning the code	13
Exercise 1: Creating a table.....	15
Exercise 2: Applying a style defined in external style sheet	17
Exercise 3: Inserting images	19
Exercise 4: Final adjustments of the screen appearance	21
<i>Chapter 4: Payment application examples: overview and prerequisites</i>	24
Overview	24
Magnetic stripe card authorisation example (MAGCARD)	24
Smart card authorisation example (ICCEMV)	24
Basic TML Payment Application example (BTMLPA)	24
Installation package and source files	25
Compilation of source files.....	26
Installing Apache Ant	26
Prerequisites for running application examples	27
<i>Chapter 5: MAGCARD application example</i>	28
Application header and variable declarations.....	28
Application logic.....	29
Initiate Transaction block	30
<i>init_prompt</i> screen	30
Transaction Amount block	31
<i>read_amount</i> screen.....	31
Risk Management block.....	32
<i>mag_rm</i> screen	32
Submit Data block	33
<i>mag_submit</i> screen	34
Connection Error block.....	34
<i>check_err</i> screen.....	34
<i>pool_full</i> screen.....	35
Host Authorisation block	35
Dynamic Page block.....	35
<i>auth_ok.tml</i> page	36
<i>amnt_big.tml</i> page.....	36
<i>failed.tml</i> page	36
Print Receipt block.....	37
<i>receipt</i> screen	38

<i>receipt_int</i> screen.....	38
<i>receipt_ext</i> screen	39
<i>rec_ext_er</i> screen	40
<i>rec_ext_ok</i> screen.....	40
Transaction Completed block	40
<i>compl_trans</i> screen.....	40
Transaction Rejected block.....	41
<i>reject_trans</i> screen.....	41
<i>void_trans</i> screen.....	41
Chapter 6: ICCEMV application example	42
Application header and variable declarations.....	42
Application logic.....	43
Initiate Transaction block	44
<i>init_prompt</i> screen	44
<i>final_init</i> screen.....	45
Transaction Amount block.....	45
<i>read_amount</i> screen.....	45
ICC Verification block.....	46
<i>get_cvm</i> screen	47
<i>enter_pin</i> screen	48
<i>verify_pin</i> screen	49
<i>bypass_cvm</i> screen.....	49
<i>cv_no_cvm</i> screen.....	50
<i>cvr_no_pin</i> screen.....	50
<i>assert</i> screen.....	50
Risk Management block.....	50
<i>risk_mgmt</i> screen	51
Submit Data block	51
<i>submit</i> screen.....	52
Connection Error block.....	53
<i>icc_conn_fld</i> screen.....	53
Host Authorisation block	54
Dynamic Page block.....	54
<i>auth_ok.tml</i> page	55
<i>failed.tml</i> page	55
<i>amnt_big.tml</i> page.....	55
ICC Authorisation block.....	56
Authorisation Results block	56
<i>subm_tc_aac</i> screen.....	57
<i>end_trans</i> screen.....	57
Transaction Rejected block.....	57
<i>reject_trans</i> screen.....	58
<i>void_trans</i> screen.....	58
Print Receipt block.....	58
<i>receipt</i> screen	58
Transaction Completed block	59
<i>compl_trans</i> screen.....	59
Remove Card block	59
<i>remove_card</i> screen.....	60
Chapter 7: BTMLPA architecture	61
BTMLPA terminal-side module	61
BTMLPA host-side module.....	62
BTMLPA Java servlets.....	62
Dynamic TML pages	63
BTMLPA database.....	63
BTMLPA web interface (GUI).....	63

<i>Chapter 8: BTMLPA details</i>	64
BTMLPA preliminaries	64
Starting a BTMLPA transaction	64
General transaction flow	64
Supported transaction types	65
Transaction Preparation block	66
REVERSAL transaction and Duplicate receipt	67
Transaction Reversal block.....	67
Receipt Duplicate block	68
SALE, SALE WITH CASHBACK or REFUND transaction	69
Initiate Transaction block	71
Manual Entry block	72
Transaction Amount block	72
Select ICC Application	73
ICC Verification	74
Risk Management block.....	76
Submit Data block.....	77
Connection Error block	79
Host Authorisation block.....	80
Dynamic Page block	82
ICC Authorisation block	84
Authorisation Results block	84
Remove Card block	84
Print Receipt block.....	85
Signature Check block.....	85
Transaction Completed block	86
Transaction Rejected block	86
assert screen.....	86

About this document



This tutorial is designed to introduce the basics of card payment application development with Incendo Online and Terminal Markup Language (TML).

The tutorial is a key to TML payment examples provided with Incendo Online distribution which you can use as a basis for development of your own TML applications.

Reading prerequisites

A reader of the document is assumed to be familiar with the main principles of web application development.

Experience of working with

- Apache Tomcat or other similar web server
- Ingenico terminals
- card payment applications

is a plus but not a prerequisite.

Related documents

The document should be studied in parallel with *TML Application Development Guidelines* which provide a general approach to TML application development and explain TML elements and attributes in detail.

For information on how to install application examples as well as other Incendo Online components, refer to *Incendo Online Desktop Installation Guidelines*.

Basics of Working with Incendo Online-enabled Terminals may be of help for dealing with Ingenico terminals or Incendo Online Terminal Simulator.

For more information about Incendo Online documentation set, refer to *Incendo Online Documentation Roadmap*.

Conventions

We use the notation `%[application name]_HOME%` to refer to the path of an application (installation) directory on your computer.

For example, `%OE_HOME%` and `%CATALINA_HOME%` denote the paths to Incendo Online host software and Apache Tomcat installation directories respectively.

If you, say, installed Apache Tomcat in the directory

`C:\Program Files\Apache Software Foundation\Tomcat 5.5`, then `%CATALINA_HOME%` would mean the same thing as

`C:\Program Files\Apache Software Foundation\Tomcat 5.5`.

Document structure

Chapter 2, [“Application selection menu” on page 7](#), discusses the TML implementation of the menu for selection of payment application examples.

Chapter 3, [“Four simple coding exercises” on page 11](#), offers a number of exercises showing how you can transform a very basic-looking screen into the one that is pleasant to look at – without changing its functionality.

Chapter 4, [“Payment application examples: overview and prerequisites” on page 24](#), provides a brief overview of TML payment application examples as well as the instructions for compiling the application source files. It also lists the prerequisites for running the application examples.

Chapters from 5 to 8 discuss the TML implementation of the terminal and the host parts of the application examples (MAGCARD, ICCEMV, and BTMLPA).

Application selection menu

2

This chapter discusses the implementation – by means of TML – of the menu for selection of TML payment application examples, or application selection menu for short.

Purpose, constituents and their locations

The purpose of the application selection menu, obviously, is to provide access to TML payment application examples (BTMLPA, MAGCARD, and ICCEMV, see [“Payment application examples: overview and prerequisites” on page 24](#)) included in the Incendo Online distribution.

[Figure 2 - 1 below](#) shows how a user sees this menu depending on the model of the terminal being used:

Figure 2 - 1: Ingenico 8550 and Ingenico 5100 application selection screens



The menu implementation includes:

- a TML page implementing the menu for *Ingenico 8550* terminals
- a TML page implementing the menu for *Ingenico 5100* terminals
- a TML page for automatic selection of the menu version appropriate for the terminal model being used
- auxiliary resources (a style sheet and images) for *Ingenico 8550*

[Table 2 - 1 below](#) gives a list of files that the menu implementation includes; for each of the files its location, name and brief description are presented. All locations in this table are relative to the base %CATALINA_HOME\webapps\ROOT. Thus, for example, \index.tml should be read as %CATALINA_HOME\webapps\ROOT\index.tml.

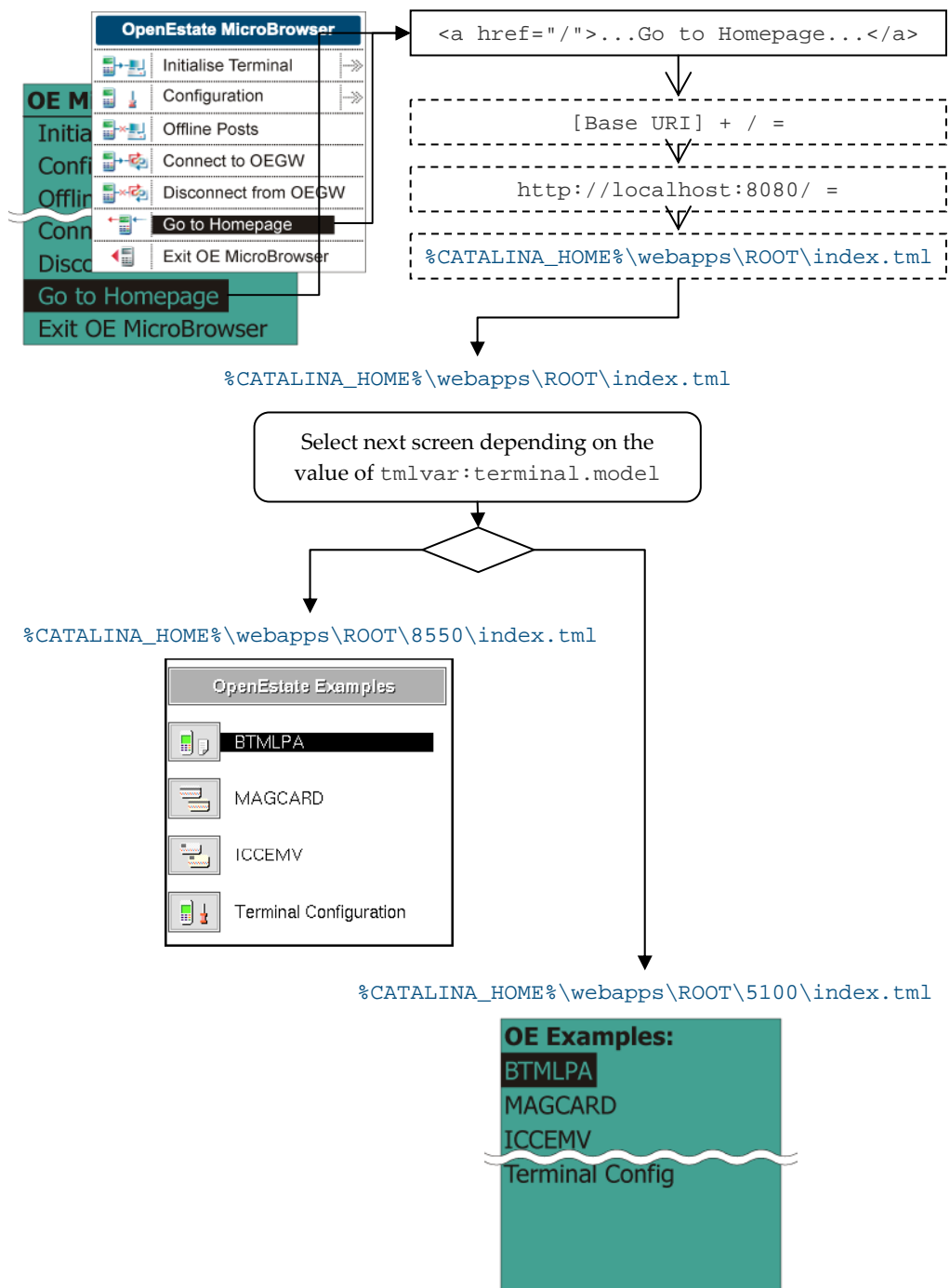
Table 2 - 1: relative location and description of the menu implementation files

File location and name	Description
\index.tml	a single screen TML page whose only function is to define the page Incendo Online Terminal MicroBrowser should switch to depending on the model of the terminal (5100 or 8550) being used
\5100\index.tml	a single screen TML page implementing a menu for selection of TML payment application examples for <i>Ingenico 5100</i> terminals
\8550\index.tml	a single screen TML page implementing a menu for selection of TML payment application examples for <i>Ingenico 8550</i> terminals
\8550\style.css	a style sheet file containing definitions of styles used to control the appearance of the screens for <i>Ingenico 8550</i> terminals
\8550\images*.gif	files containing the images used on the application selection screen for <i>Ingenico 8550</i> terminals; these files are referenced from the file \8550\index.tml

Application selection menu access mechanism

The application selection menu is displayed when a user selects the **Go to Homepage** function in the **Incendo Online MicroBrowser** start-up screen (Figure 2 - 2 below).

Figure 2 - 2: processing the Go to Homepage link



This function is implemented as a hyperlink which within the corresponding TML screen is defined as `...Go to Homepage...`.

When this hyperlink is selected, the terminal issues and sends to Incendo Online Gateway a request for a TML page with the URL `/`.

Incendo Online Gateway prefixes the URL received from the terminal with the value of the base URL parameter^a whose default value is `http://localhost:8080`. So the URL of the page requested by Incendo Online Gateway on the terminal's behalf is `http://localhost:8080/`.

Since, `localhost` means "this computer," that is, the one on which Incendo Online Gateway runs and 8080 is the port of the Apache Tomcat web server, the URL `http://localhost:8080/` for desktop Incendo Online installation identifies the default page deployed on Apache Tomcat, that is, the page `index.html` located in the directory `%CATALINA_HOME\webapps\ROOT`^b.

The code for this page looks this way:

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#dispatcher" cancel="#dispatcher"/>
  </head>
  <screen id="dispatcher" cancel="#dispatcher">
    <next uri="5100/index.html">
      <variant lo="tmlvar:terminal.model" op="equal" ro="8550"
uri="8550/index.html"/>
    </next>
  </screen>
</tml>
```

As you can see, the page contains one screen called `dispatcher`, and there is nothing on this screen that the terminal could display. The only screen's purpose is to specify what page the terminal should go to next. The page to switch to is defined by the `next` element (in combination with its child `variant` element).

Incendo Online MicroBrowser first processes the `variant` element. If the value of the variable `terminal.model` is equal to 8550, the terminal switches to the screen (or page) whose URL is defined by the value of the `uri` attribute of the `variant` element (`8550/index.html`). Note that the URL is specified relative to the "current location", and thus `uri="8550/index.html"` identifies the file `%CATALINA_HOME\webapps\ROOT\8550\index.html`.

If the condition defined in the `variant` element is false, the screen (or page) to switch to is defined by the value of the `uri` attribute of the `next` element.

The `uri="5100/index.html"` in this case is an equivalent of `%CATALINA_HOME\webapps\ROOT\5100\index.html`.

Implementation for Ingenico 5100 (the file `5100\index.html`)

The code for the page `%CATALINA_HOME\webapps\ROOT\5100\index.html` is shown in [Figure 2 - 3 below](#).

Figure 2 - 3: TML code of the `5100\index.html` file

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmplpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccmv/iccmv.tml" rev="text/tml"/>
  </head>
  <screen id="initialscr">
    <next uri="5100/index.html">
      <variant lo="tmlvar:terminal.model" op="equal" ro="8550"
uri="8550/index.html"/>
    </next>
  </screen>
</tml>
```

^a This parameter can be set individually for each terminal via Incendo Online Gateway Web GUI. For more details, refer to *Incendo Online Gateway User Guide*.

^b Since the directory name is not specified, Apache Tomcat looks for the page in the directory `%CATALINA_HOME\webapps\ROOT`. As the file name is also not specified the page with the name `index` is assumed by default.

```
</head>
<screen id="initialscr" cancel="#initialscr">
  <display>
    <h1>OE Examples:</h1>
    <a href="/btmlpa/tmlapp.tml">BTMLPA</a>
    <br/>
    <a href="/magcard/magcard.tml">MAGCARD</a>
    <br/>
    <a href="/iccemv/iccemv.tml">ICCEMV</a>
    <br/>
    <a href="emb://embedded.tml">Terminal Config</a>
  </display>
</screen>
</tml>
```

The `link` elements (within the `head` element) define the linked TML pages – the ones that should be downloaded from the server and stored in the terminal cache right away. When a user (at a later time) selects a hyperlink pointing to such page, the page will be retrieved from the terminal cache rather than requested from the server.

Note that the `href` attributes in this case (for example, `href="/btmlpa/tmlapp.tml"`) define *absolute* locations of the resources (compare with the `uri="8550/index.tml"` and `uri="5100/index.tml"` in the code for `%CATALINA_HOME\webapps\ROOT\index.tml` in [Figure 2 - 3 on page 9](#)).

As in the case with other URLs, Incendo Online Gateway, prior to requesting the resources from the server, adds the prefix `http://localhost:8080` thus transforming `/btmlpa/tmlapp.tml`, `/magcard/magcard.tml`, and `/iccemv/iccemv.tml` into `http://localhost:8080/btmlpa/tmlapp.tml`, `http://localhost:8080/magcard/magcard.tml`, and `http://localhost:8080/iccemv/iccemv.tml` respectively.

The only screen within the `5100\index.tml` page called `initialscr` contains four hyperlinks defined by the `a` elements. The targets (specified by the `href` attributes) of the first three hyperlinks identify TML implementations of BTMLPA, MAGCARD, and ICCEMV deployed on Apache Tomcat web server.

As for the fourth one (`Terminal Config`), it points to the embedded TML page permanently residing in the terminal memory which, among other screens, implements the **Incendo Online MicroBrowser** start-up screen.

The TML implementation of application selection menu for *Ingenico 8550* (the file `8550\index.tml`) is discussed in detail in [“Four simple coding exercises” on page 11](#). The code for the page `8550\index.tml` is shown on [Figure 3 - 13 on page 22](#).

Four simple coding exercises

3

The exercises described in this chapter are aimed at giving you practical experience of creating visually appealing screens for Incendo Online-enabled Ingenico payment devices. After completing these very simple exercises you will be able to:

- design the appearance of your screens and map it onto the code elements thus creating “code blueprints”
- use tables – an effective instrument for accurate positioning of screen elements
- use the styles defined in external style sheets enabling you to create visually consistent user interfaces and considerably speeding up TML code development
- use graphical images as screen elements
- use very simple means for making fine adjustments in the appearance of your screens

Overview of the exercises

The main purpose of the coding exercises described in the following sections is to show you how to transform a very basic-looking screen into the one that is pleasant to look at – without changing its functionality.

In the course of the exercises, you will work with the file `index.tml` that implements a menu for selection of TML payment application examples (BTMLPA, MAGCARD, and ICCEMV).

You will be making changes to the version of the file deployed on Apache Tomcat web server, that is, with the one located in the directory
`%CATALINA_HOME%\webapps\ROOT\8550`.

This will allow you to see the immediate result of your manipulations on the terminal display or by means of the Terminal Simulator – as soon as your terminal (or the Terminal Simulator) gets the updated version of the page from the server.

You will start working with the version of `index.tml` developed for terminal models with limited graphical capabilities (such, for example, as *Ingenico 5100*). This file should have been deployed by now in `%CATALINA_HOME%\webapps\ROOT\5100`. To be able to view this file with a real *Ingenico 8550* terminal or the Terminal Simulator (which is assumed and should be now working as the emulation of *Ingenico 8550*^a), this file should be copied to `%CATALINA_HOME%\webapps\ROOT\8550` (see [“Application selection menu access mechanism” on page 8](#)).

To edit `index.tml`, you can use any text editor available on your computer.

Prerequisites

Before proceeding to the exercises make sure that:

- Incendo Online host software as well as third-party components required for its functioning are installed on your computer as described in *Incendo Online Desktop Installation Guidelines*.
- MySQL Server, Incendo Online Gateway, and Apache Tomcat are running.
- An *Ingenico 8550* terminal with Incendo Online MicroBrowser installed and running is connected to your computer

^a This is the default mode of operation for Incendo Online Terminal Simulator.

or

Incendo Online Terminal Simulator is installed on your computer as described in *Incendo Online Desktop Installation Guidelines* and is running.

- The terminal you are using (or Incendo Online Terminal Simulator) has already been initialised (for instructions on how to initialise a terminal, refer to *Incendo Online Desktop Installation Guidelines* or *Configuration and Initialisation of Incendo Online-enabled Terminals*).
- The file `index.tml` originally deployed in the directory `%CATALINA_HOME%\webapps\ROOT\8550` has been overwritten with the one from the directory `%CATALINA_HOME%\webapps\ROOT\5100`.
- A text editor is available on your computer.

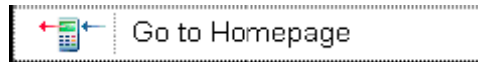
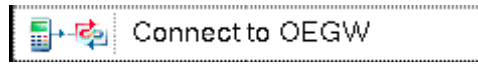
Commands you are going to use

Each time after you made changes to `index.tml` (and saved a new version of the file under the same name), you should make your terminal (or the Terminal Simulator) request and receive the updated version of the page from the server – so that you could see the result.

In addition to that, during the exercises you are likely to very frequently switch between the **Incendo Online MicroBrowser** (start-up) and the **Incendo Online Examples** screens. (The latter is the application examples selection screen corresponding to the file `index.tml`.)

[Table 3 - 1 below](#) gives the summary of how to perform these actions.

Table 3 - 1: frequently performed actions

To	Do this
Switch from the Incendo Online MicroBrowser (start-up) screen to the Incendo Online Examples screen (the application selection screen corresponding to the version of the file <code>index.tml</code> currently stored in the cache of the terminal (or the Terminal Simulator))	Select the Go to Homepage command in the Incendo Online MicroBrowser screen: 
Switch from the Incendo Online Examples screen back to the Incendo Online MicroBrowser (start-up screen)	Select the Terminal Configuration (or the Terminal Config) in the Incendo Online Examples screen.
Make the terminal (or the Terminal Simulator) connect to the server, download the updated versions of all resources (including <code>index.tml</code>), and store them in cache	Select the Connect to OEGW command in the Incendo Online MicroBrowser (startup) screen: 

If you quit the exercises before completing them

If for some reason you decided to quit the exercises before completing them, restore the initial state of the file `%CATALINA_HOME%\webapps\ROOT\8550\index.tml` – the one this file had before the exercises.

To do that, overwrite this file with its “original” version, that is, with `%OE_EXAMPLES\index\8550\index.tml`.

However, if you carry out the exercises in full, you do not have to worry about that as the exercises are designed so that in the end this file will come back to its initial state and thus the “status quo” of the original deployment of web components will be restored.

A note for HTML developers

Since the changes made to the page `index.html` in the course of the exercises do not go beyond the scope of pure HTML, you can make all the modifications yourself – in one go – right after looking through “[Exercise 0: Designing a screen and planning the code](#)” below. Having done that, compare your final code with that presented in “[Exercise 4: Final adjustments of the screen appearance](#)” on page 21.

Tip. To position (align) the screen elements within the cells of a table, one of the styles defined in `%CATALINA_HOME%\webapps\ROOT\8550\style.css` is used. As a self-test you can try and guess which one and then use it in the code of the `index.html` page.

Also note, that the image files you are going to use are all located in the directory `%CATALINA_HOME%\webapps\ROOT\8550\images`. For more information on the image files, see the table in “[Exercise 3: Inserting images](#)” on page 19.

Exercise 0: Designing a screen and planning the code

It is always a good idea to do a little bit of planning prior to actually coding.

In this exercise we will design a screen and work out a coding plan that we will then implement during the rest of the exercises.

Once you have copied the file `index.html` from the directory `%CATALINA_HOME%\webapps\ROOT\5100` into the directory `%CATALINA_HOME%\webapps\ROOT\8550`, open it in a text editor. The code initially looks like this:

Figure 3 - 1: initial state of the `index.html` file

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmlpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccemv/iccemv.tml" rev="text/tml"/>
  </head>
  <screen id="initialscr" cancel="#initialscr">
    <display>
      <h1>OE Examples:</h1>
      <a href="/btmlpa/tmlapp.tml">BTMLPA</a>
      <br/>
      <a href="/magcard/magcard.tml">MAGCARD</a>
      <br/>
      <a href="/iccemv/iccemv.tml">ICCEMV</a>
      <br/>
      <a href="emb://embedded.tml">Terminal Config</a>
    </display>
  </screen>
</tml>
```

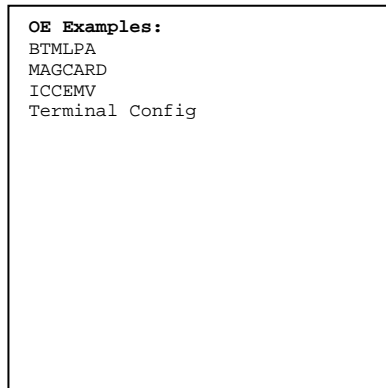
To see the screen that this code defines:

1. Select **Connect to OEGW** in the **Incendo Online MicroBrowser** screen to receive the current version of `index.html` from the server.

The terminal (or the Terminal Simulator) connects to the server and receives the updated versions of the resources (including that of `index.html`). Once this is done, the “Done” message is displayed for a short period of time.

2. Select **Go to Homepage** in the **Incendo Online MicroBrowser** screen to switch to the **Incendo Online Examples** screen.

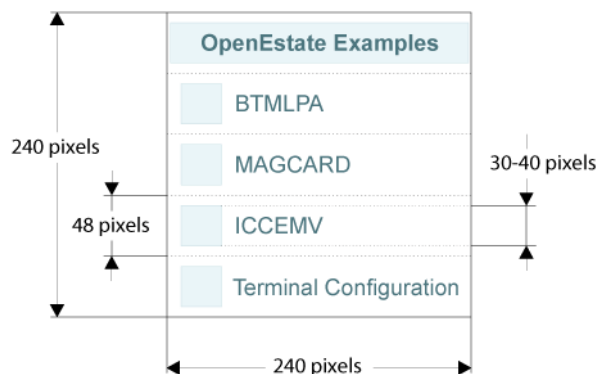
This screen will look like [Figure 3 - 2 on page 14](#).

Figure 3 - 2: initial (unformatted) screen

The obvious disadvantage of this screen is that the screen area is used very inefficiently: its elements are poorly arranged and take only about 20% of available area. The text is tiny so the people who can not see very well may have difficulties reading it. In addition to that, the screen does not look attractive at all, to say the least. (And attractive appearance is not the kind of thing that can be neglected when user interface is concerned!)

Once we have pointed out the main screen's disadvantages, we are ready to think about how to better the situation. So it is time now we drew a sketch of what we want our screen to look like. You can do it on a sheet of paper or use drawings/image editing software.

The solution we came up with after a short while is shown on [Figure 3 - 3 below](#).

Figure 3 - 3: sketch of the initial screen design

As can be seen from the diagram, we decided that we should:

- split the screen area (which for the *Ingenico 8550* terminals is 240 pixels high and 240 pixels wide) into five “stripes” of equal height (48 pixels high each): one for the screen title and the rest four – for the hyperlinks (menu items)
- use images: one in place of the screen title and four – to accompany the text of the hyperlinks; the height of all images may be about the same (30–40 pixels); the top image should be almost as wide as the screen itself (230–235 pixels); the width of the rest of the images should be in the range of 30–50 pixels
- considerably enlarge the text of the hyperlinks (menu items)

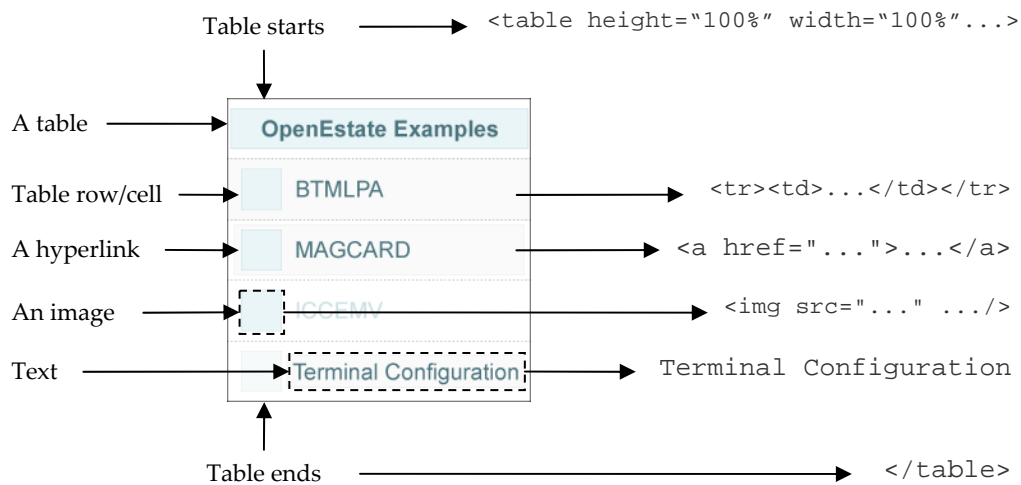
Having decided what our screen is to look like, we now have to “map” our sketch onto the elements of different type – the ones our screen can be made of.

We decided that we use a table with the height and width equal to those of the screen itself. The table will act as a skeleton of the screen, or – in other words – as a container for the screen elements (images and text). So we will be able to control their arrangement on the screen by appropriately defining the table's properties.

It is more or less obvious that the table should contain five rows, and there should be one cell in each row.

Having decided on the screen structure and its elements, we listed the code elements that we are likely to use (see the right part of the diagram). So now we have a plan (a “blueprint” of the code to be developed) that we are going to implement step-by-step during the exercises (Figure 3 - 4 below).

Figure 3 - 4: details of the initial screen design



Note that some of the elements are already in place (see the code of `index.html` in Figure 3 - 1 on page 13) and all we have to do is to build a table around those elements and add images.

Exercise 1: Creating a table

During this exercise you will create a table containing five single-cell rows – to hold the rest of the screen elements in place.

1. Open the file `index.html` located in the directory `%CATALINA_HOME%\webapps\ROOT\8550` in a text editor – if you have not done so yet.
2. Add the opening `<table>` tag after the opening `<display>` tag. Add the closing `</table>` tag before the closing `</display>` tag.
3. Specify the table's size (the height and the width of the table) by defining the `height` and the `width` attributes within the opening `<table>` tag:

```
<table height="100%" width="100%">
```

The table's size is specified as the percentage of the terminal display area. The table's height and width are set to be equal to the height and width of the display (irrespective of how many pixels the display is actually capable of reproducing both in vertical and horizontal directions).

Note: when you specify the height and width of a table as a percentage of the screen, you don't have to worry about the actual number of the pixels in the terminal display.

4. Define table rows and cells:
 - 4.1 Add the sequence `<tr><td>` before `<h1>` and the sequence `</td></tr>` after `</h1>`.
 - 4.2 Insert the sequences `<tr><td>` before all opening `<a>` tags and the sequences `</td></tr>` after all closing `` tags.

In this way, the rows and cells are “built around” the screen title (the heading `<h1>OE Examples:</h1>`) and all hyperlinks (specified by means of the `a` elements).

5. To center the contents of the cells vertically (in relation to the cell borders), set the `valign` attribute of each `tr` element to `"middle"`:

```
<tr valign="middle">
```


Note that vertical alignment for the contents of the cells is defined at the “row level”; if the table rows contained more cells, the vertical alignment specified would be applied to the contents of all cells within the rows.

Tip: if the elements with similar properties form a hierarchy (as in the case of table rows and cells), always try to specify a property at the highest possible level (the way you did it for vertical alignment). The specified property will then “propagate” lower down in the hierarchy of elements so that you won’t have to repeat the property definition separately for each of more specific elements. In this way you will save a lot of time at the coding stage as you will have to type less and your code will be shorter. Code editing will as well become an easier task since you code will be much more readable.

6. Remove all `</br>` tags.

The position of elements within the screen is now defined by the properties of the table and its cells; all line breaks (the `</br>` elements) have become unnecessary.

7. Spell out the word ‘Configuration’ in the ‘Terminal Config’ – as you have probably noticed there is enough space on the screen for the full version of this word – locate the text `Terminal Config` and type `uration` after the `g`.
8. Format the code to improve its readability: break and indent the lines so that the screen structure and the hierarchy of its elements are obvious.

The code now should look something like this:

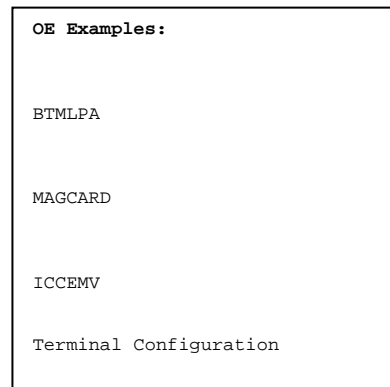
Figure 3 - 5: the code of the `index.tml` file at the end of the Exercise 1

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmlpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccemv/iccemv.tml" rev="text/tml"/>
  </head>
  <screen id="initialscr" cancel="#initialscr">
    <display>
      <table height="100%" width="100%">
        <tr valign="middle"><td>
          <h1>OE Examples:</h1>
        </td></tr>
        <tr valign="middle"><td>
          <a href="/btmlpa/tmlapp.tml">BTMLPA</a>
        </td></tr>
        <tr valign="middle"><td>
          <a href="/magcard/magcard.tml">MAGCARD</a>
        </td></tr>
        <tr valign="middle"><td>
          <a href="/iccemv/iccemv.tml">ICCEMV</a>
        </td></tr>
        <tr valign="middle"><td>
          <a href="emb://embedded.tml">Terminal
Configuration</a>
        </td></tr>
      </table>
    </display>
  </screen>
</tml>
```

9. Save the changes in the file `index.tml`.
10. Update the resources stored in the terminal cache and view the result. For instructions on how to do that, refer to “[Commands you are going to use](#)” on [page 12](#).

The terminal screen should resemble [Figure 3 - 6 below](#).

Figure 3 - 6: terminal screen at the end of the Exercise 1



Exercise 2: Applying a style defined in external style sheet

You can control the appearance and positioning of elements on the screen by explicitly defining attributes of corresponding elements right in your TML code. However, a much more elegant and efficient way of doing this is by means of the formatting styles defined in external style sheets (`.css` files). When using such styles, you have an ability to completely change the look and feel of the user interface without making any changes to the TML code. To do that, you just redefine the properties of the formatting styles in the `.css` file(s).

To be able to apply an external style to an element or a group of elements within your TML page or screen, you have to:

1. Create a `.css` file in which you “describe” the style(s) you are intending to use in a special style definition language (see “TML CSS” in *TML Application Development Guidelines*).
2. Define a link to the `.css` file in a `.tml` file.
3. Make a reference to the style from a TML element to which you want to apply the style.

In this exercise we will skip step 1 of this procedure, since the `.css` file containing the definition of the style that we are going to use is already within your reach. This is the file `style.css` located in the directory `%CATALINA_HOME%\webapps\ROOT\8550`. Once you have found this file, open it in a text editor. The style of interest is `c_href_menu` ([Figure 3 - 7 below](#)).

Figure 3 - 7: `c_href_menu` style definition in the `style.css`

```
.c_href_menu {font-family: helvetica; font-size: large; text-indent: 2%; vertical-align: middle;}
```

This means that the textual content of an element to which this style is applied should be displayed using the font called *helvetica*; the size of the characters should be large; the contents should be left-aligned (if the horizontal alignment is not specified explicitly, the left alignment is assumed by default) and indented by 2% of an element’s size; in vertical direction, the contents should be placed in the middle of an element.

After this very brief introduction, you can come back to coding:

4. Open the file `index.tml` located in the directory `%CATALINA_HOME%\webapps\ROOT\8550` in a text editor – if the file is not open yet.

5. To define a link to the `style.css` file, insert the following link element:
`<link href="style.css" rev="stylesheet"/>`
within the head element (that is, somewhere between the opening and closing `<head>` tags).

Note that since the file `style.css` is located in the same directory as the file `index.tml` you are working with now, the value of the `href` attribute is just `"style.css"` (compare with the values of the `href` attribute for the `.tml` files which are also linked to the `index.tml` file). Also note that the value of the `rev` attribute is `"stylesheet"`. This is the only possible value of this attribute for a linked external style sheet.

6. To apply the style `c_href_menu` to all (child) elements of the table element, add the attribute `class` with the value set to `"c_href_menu"` somewhere within the opening `<table>` tag:
`<table height="100%" width="100%" class="c_href_menu">`
7. Remove the `valign="middle"` within all opening `<tr>` tags: since the style `c_href_menu`, among other properties, specifies the middle alignment in the vertical direction, the specification of vertical alignment in `tr` elements has now become redundant.
8. Format the code: break and indent the lines where and as appropriate.

Figure 3 - 8: the code of the `index.tml` file at the end of the Exercise 2

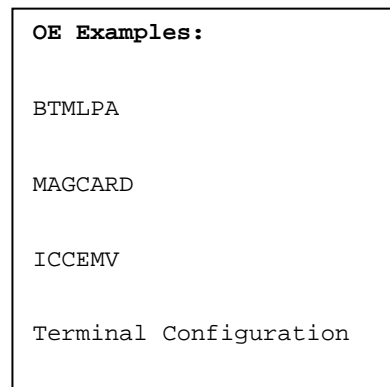
```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmlpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccemv/iccemv.tml" rev="text/tml"/>
    <link href="style.css" rev="stylesheet"/>
  </head>
  <screen id="initialscr" cancel="#initialscr">
    <display>
      <table height="100%" width="100%" class="c_href_menu">
        <tr><td>
          <h1>OE Examples:</h1>
        </td></tr>
        <tr><td>
          <a href="/btmlpa/tmlapp.tml">BTMLPA</a>
        </td></tr>
        <tr><td>
          <a href="/magcard/magcard.tml">MAGCARD</a>
        </td></tr>
        <tr><td>
          <a href="/iccemv/iccemv.tml">ICCEMV</a>
        </td></tr>
        <tr><td>
          <a href="emb://embedded.tml">Terminal
Configuration</a>
        </td></tr>
      </table>
    </display>
  </screen>
</tml>
```

9. Save the changes in the file `index.tml`.

- Update the resources stored in the terminal cache and view the result. For instructions on how to do that, refer to [“Commands you are going to use” on page 12.](#)

The screen you have modified now looks like this:

Figure 3 - 9: terminal screen at the end of the Exercise 21



Exercise 3: Inserting images






To make your screens look visually appealing, you are likely to use graphical images for various different purposes: as controls (for example, as hyperlinks), screen headings, and so on – in place of or in combination with plain text. This exercise demonstrates how you can do that.

You do not need to create the images for this exercise yourself. We suggest that you use the ones located in the directory

`%CATALINA_HOME%\webapps\ROOT\8550\images.`

The details of the images are given in the [Table 3 - 2 below.](#)

Table 3 - 2: list of images used

Image	Width (pixels)	Height (pixels)	File name
	234	32	top.gif
	44	34	btmlpa.gif
	44	34	magcard.gif
	44	34	iccmv.gif
	44	34	conf.gif

Just by the way, note that the images in the table above are arranged the way we want them to appear on the screen. Also note that the dimensions of the images are about the same as we worked out in [“Exercise 0: Designing a screen and planning the code” on page 13.](#)

Having read this short introduction, you have all the necessary information to proceed with the coding:

- Open the file `index.tml` located in the directory `%CATALINA_HOME%\webapps\ROOT\8550` in a text editor – if the file is not open yet.

2. To replace the text **OE Examples:** with the image contained in the file `top.gif`, replace the sequence `<h1>OE Examples:</h1>` with this one:
``.

Since the image file is located in the directory `images` which is a subdirectory of `8550` where `index.tml` itself is located, the value of the `src` attribute is `"images/top.gif"` (rather than just `"top.gif"`).

3. To add images in front of the text of the hyperlinks, insert:
 - o `` before BTMLPA
 - o `` before MAGCARD
 - o `` before ICCEMV
 - o `` before Terminal Configuration

Note that since the `img` elements are placed within the `a` elements, the images will be part of corresponding hyperlinks on the screen and thus they will be “sensitive” to mouse clicks or stylus touches – depending on whether the Terminal Simulator or the terminal equipped with a touch screen is used.

4. Format the code: break and indent the lines where and as appropriate.

The code now should look something like this:

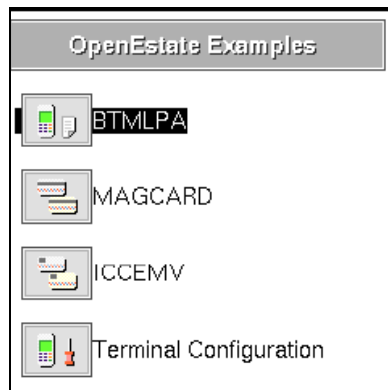
Figure 3 - 10: the code of the `index.tml` file at the end of the Exercise 3

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmlpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccemv/iccemv.tml" rev="text/tml"/>
    <link href="style.css" rev="stylesheet"/>
  </head>
  <screen id="initialscr" cancel="#initialscr">
    <display>
      <table height="100%" width="100%" class="c_href_menu">
        <tr><td>
          
        </td></tr>
        <tr><td>
          <a href="/btmlpa/tmlapp.tml">
            BTMLPA
          </a>
        </td></tr>
        <tr><td>
          <a href="/magcard/magcard.tml">
            MAGCARD
          </a>
        </td></tr>
        <tr><td>
          <a href="/iccemv/iccemv.tml">
            ICCEMV
          </a>
        </td></tr>
        <tr><td>
          <a href="emb://embedded.tml">
            Terminal
            Configuration
          </a>
        </td></tr>
      </table>
    </display>
  </screen>
</tml>
```

5. Save the changes in the file `index.html`.
6. Update the resources stored in the terminal cache and view the result. For instructions on how to do that, refer to [“Commands you are going to use” on page 12](#).

The terminal screen will now look like [Figure 3 - 11 below](#).

Figure 3 - 11: terminal screen at the end of the Exercise 3



Exercise 4: Final adjustments of the screen appearance

In this exercise you will make fine adjustments in the appearance of the screen and its elements:

1. Open the file `index.html` located in the directory `%CATALINA_HOME%\webapps\ROOT\8550` in a text editor – if the file is not open yet.
2. To move all objects on the screen a little bit – mainly in order to provide spacing between the top image and the screen border – insert a padding of two pixels into all cells of the table. To do that, add the `cellpadding` attribute with the value of "2" to the `table` element:

```
<table height="100%" width="100%" class="c_href_menu" cellpadding="2">
```
3. To add spacing between the images and the text that follow within the hyperlinks, insert two non-breaking spaces (` `) after the `img` elements:
 - o ` BTMLPA`
 - o ` MAGCARD`
 - o ` ICCEMV`
 - o ` Terminal Configuration`
4. To make the whole line containing the hyperlink highlighted, when the hyperlink gets into focus, add sequences of non-breaking spaces (` `) after the text of the hyperlinks (before the closing `` tags):
 - o 27 spaces after BTMLPA
 - o 23 spaces after MAGCARD
 - o 27 spaces after ICCEMV
 - o 5 spaces after Terminal Configuration
5. Format the code: break and indent the lines where and as appropriate.

The code now should look something like [Figure 3 - 13 on page 22](#).

6. Save the changes in the file `index.html`.

7. Update the resources stored in the terminal cache and view the result. For instructions on how to do that, refer to [“Commands you are going to use” on page 12.](#)

The exercises are now completed so you can stop all applications that are running, switch off the terminal and disconnect it from the PC – in the case you were using a real *Ingenico 8550* terminal rather than the Terminal Simulator, and, finally, turn your PC off.

The terminal screen will now look like [Figure 3 - 12 below.](#)

Figure 3 - 12: terminal screen at the end of the Exercise 4

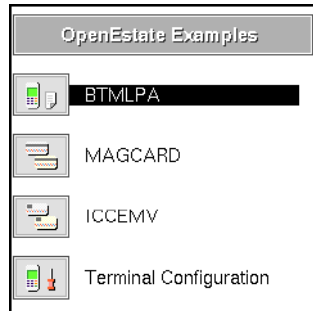


Figure 3 - 13: final version of the index.tml file

```
<?xml version="1.0" encoding="UTF-8"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="#initialscr" cancel="#initialscr"/>
    <link href="/btmlpa/tmlapp.tml" rev="text/tml"/>
    <link href="/magcard/magcard.tml" rev="text/tml"/>
    <link href="/iccemv/iccemv.tml" rev="text/tml"/>
    <link href="style.css" rev="stylesheet"/>
  </head>
  <screen id="initialscr" cancel="#initialscr">
    <display>
      <table height="100%" width="100%" class="c_href_menu"
        cellpadding="2">
        <tr><td>
          
        </td></tr>
        <tr><td>
          <a href="/btmlpa/tmlapp.tml">
            &#160;&#160;
            BTMLPA&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
          </a>
        </td></tr>
        <tr><td>
          <a href="/magcard/magcard.tml">
            &#160;&#160;
            MAGCARD&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;
          </a>
        </td></tr>
        <tr><td>
          <a href="/iccemv/iccemv.tml">
            &#160;&#160;
            ICCMV&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
            &#160;&#160;&#160;&#160;&#160;&#160;&#160;&#160;
          </a>
        </td></tr>
        <tr><td>
          <a href="emb://embedded.tml">
            &#160;&#160;

```

```
Terminal Configuration&#160;&#160;&#160;&#160;&#160;
    </a>
  </td></tr>
</table>
</display>
</screen>
</tml>
```

Payment application examples: overview and prerequisites

4

Incendo Online distribution includes a set of ready-to-use payment application examples which introduce the TML programming essentials and provide the basis for custom TML application development.

Incendo Online examples are standard Java Web applications developed using Java Web services technology. Each application includes the terminal and the host sides. The terminal side implements the user interface functionality. It consists of one or more static TML pages which are downloaded by the terminal from the server, stored in the terminal cache, and can be browsed by the terminal user in the web-like manner.

The host side implements the application business logic. It is built on standard Java Web applications technology and consists of Java servlets that dynamically process terminal requests and construct dynamic TML responses. The Web components run on Apache Tomcat web server.

Overview

The following TML payment application examples are supplied with Incendo Online:

- Magnetic stripe card authorisation example (MAGCARD)
- Smart card authorisation example (ICCEMV)
- Basic TML Payment Application example (BTMLPA)

Magnetic stripe card authorisation example (MAGCARD)

The example is recommended as a good starting point for developers who want to get a basic understanding of TML and Incendo Online functionality.

MAGCARD demonstrates how a simple magnetic stripe card transaction can be processed by means of TML. This example will show you how a TML application can read data from the magnetic stripe card, accept user input from the terminal keypad, use TML variables for collecting data, submit the variables to the host side, and print a simple receipt on the embedded terminal printer.

Smart card authorisation example (ICCEMV)

ICCEMV example is similar to the MAGCARD example but deals with ICC EMV^a also known as smart cards.

Basic TML Payment Application example (BTMLPA)

This is a more complicated example primarily focused on specifics of APACS guidelines implementation. It helps to understand more complex concepts about terminal application design and can be recommended for developers who already have basic experience of using TML.

BTMLPA sticks to the APACS 30 standard which defines the transaction data flow and authorisation methods used for magnetic stripe and smart cards. It supports the following transaction types (according to APACS 30, section 2.2):

- authorisation of a **SALE**
the default option which allows authorising a sale of products or services.

^a ICC is a short for Integrated Circuit Card; EMV – for EuroPay, MasterCard, and Visa.

- authorisation of a **SALE WITH CASHBACK**
a sale where a customer can be given back some cash.
- authorisation of a **REFUND**
opposite to the sale – the customer brings back the goods and is refunded the payment
- authorisation of a **REVERSAL** (or voiding the last transaction)
Allows cancelling any of the above authorisations within the 30 seconds from the time the authorisation took place.

Installation package and source files

The TML application examples are distributed as a .zip file. The name of the file is `oe-examples-[version number].zip`

The distribution for each of the application examples includes a binary release represented by a web archive (.war) file and the source files.

To install the examples, simply unpack the archive into any directory that you wish, and then deploy the .war file on your web server.

The components of the binary release of Incendo Online host software as well as corresponding installation procedure are described in detail in *Incendo Online Desktop Installation Guidelines*. So this section deals solely with the aspects related to the source files not covered elsewhere.

The table that follows contains the lists of the source files for each of the application examples as well as the information about the source files location.

Table 4 - 1: source file locations for each of the application examples

Application	Source files location ^e	Source files list ^f
MAGCARD	<ul style="list-style-type: none"> • For <i>Ingenico 5100</i>: <code>magcard\src\source\web\5100\</code> • For <i>Ingenico 8550</i>: <code>magcard\src\source\web\8550\</code> 	amnt_big auth_ok exception failed magcard
ICCEMV	<ul style="list-style-type: none"> • For <i>Ingenico 5100</i>: <code>iccmv\src\source\web\5100\</code> • For <i>Ingenico 8550</i>: <code>iccmv\src\source\web\8550\</code> 	amnt_big auth_ok exception failed iccmv
BTMLPA	<ul style="list-style-type: none"> • For <i>Ingenico 5100</i>: <code>btmlpa\src\source\web\5100\</code> • For <i>Ingenico 8550</i>: <code>btmlpa\src\source\web\8550\</code> 	error.dtml exception.dtml icc_auth_rslt.dtml mag_auth_rslt.dtml pmtstyle.css reversed tmlapp tmlapp_icc

As you can see, each of the application examples includes two sets of resources: one for *Ingenico 5100* and the other – for *Ingenico 8550* terminals. In terms of the application logic, behaviour and functionality both sets are equivalent. The only difference is in the appearance of the application screens.

^e All locations are relative to the directory where you have unpacked the .zip archive (%OE_EXAMPLES). So, for example, the `magcard\src\source\web\5100\` should be read as `%OE_EXAMPLES\magcard\src\source\web\5100\`.

^f If the file name extension is missing, the .tml extension is meant.

Thanks to *Ingenico 8550* terminals' advanced graphic capabilities, the screens for them can be made visually very rich, with a lot of graphic images as their elements.

Consequently, the resources for *Ingenico 8550* contain more files. In addition to what is listed in the table above, the TML implementation of each of the application examples also includes image files and a style sheet (the `style.css` file). The image files are located in the directories

`%OE_EXAMPLES\[application name]\src\source\web\8550\images` and the style sheet files – in the directories

`%OE_EXAMPLES\[application name]\src\source\web\8550`.

The `[application name]` here stands for the `magcard`, `iccmv`, or `btmplpa`.

Compilation of source files

You may need to compile Incendo Online application examples' source files once you have altered them. After the compilation you should deploy the resulting web archive (`.war`) files on Apache Tomcat web server as described in the *Incendo Online Desktop Installation Guidelines*.

Important: Avoid moving the source files that you are going to compile from their original location, that is from `%OE_EXAMPLES\...`. Otherwise, you will not be able to compile them properly.

The examples source files are compiled using Apache Ant – a Java-based build tool (for installation instructions, refer to “[Installing Apache Ant](#)” below).

The build process is controlled via a set of Ant build files (`build.xml`) created for every application. These files are located in the directories

`%OE_EXAMPLES\[application name]\src`, where the `[application name]` stands for either the `magcard`, `iccmv`, or `btmplpa`.

You should not edit the contents of the `build.xml` files.

To compile the source files:

1. Click **Start** at the bottom left corner of the screen, and then click **Run**.

The **Run** window is displayed.

2. To access Windows command interpreter **cmd.exe**: in the field next to **Open**, type `cmd`, and then click **OK**.

The **cmd.exe** window is displayed.

3. If the current drive (shown at the beginning of the last line as `[drive letter]:`) is not the one on which the file `build.xml` is located, switch to the required drive. Type `[drive letter]:` (for example, `D:`), and then press **Enter**.

The drive you have just specified is shown at the beginning of the current line.

4. To switch to the directory where the file `build.xml` is located, type `cd [%OE_EXAMPLES\[application name]\src]`, for example, `cd C:\oe-examples\magcard\src`, and press **Enter**.
5. To initiate the build process, type `ant.bat` and press **Enter**.

The build process starts. The resulting `.war` file is created in the directory

`%OE_EXAMPLES\[application name]\src\dist` (for example, the `%OE_EXAMPLES\magcard\src\dist` directory).

6. If necessary, repeat steps 4 and 5 for the rest of the application examples.

Installing Apache Ant

1. Download the binary distribution of Apache Ant from <http://ant.apache.org/>.

For Windows® operating system, this is, normally, an archive (`.zip`) file.

2. Decompress the archive. As a result, a new directory called something like `apache-ant-1.6.5` will be created; all the necessary application components will be placed there. This new directory hereafter is referred to as the “Ant directory”.

Create an environment variable `ANT_HOME` to store the local path to the Ant directory (for example, `C:\apache-ant-1.6.5g`). Refer to the Appendix A of the *Incendo Online Desktop Installation Guidelines* for instructions.

3. Add the path to the Ant `bin` directory (for example, `C:\apache-ant-1.6.5\bin`) to the system variable `Path`. Refer to the Appendix A of the *Incendo Online Desktop Installation Guidelines* for instructions.

You can specify the path “literally” or use the reference to the variable `ANT_HOME` in the path definition: `%JAVA_HOME%\bin`.

Prerequisites for running application examples

To be able to run the Incendo Online TML application examples, you have to make sure that:

- Incendo Online host software as well as third-party components required for its functioning are installed on your computer as described in *Incendo Online Desktop Installation Guidelines*.
- MySQL Server, Incendo Online Gateway, and Apache Tomcat are running.
- An Ingenico UNICAPT™ 32 terminal with Incendo Online MicroBrowser installed and running is connected to your computer
or
Incendo Online Terminal Simulator is installed on your computer as described in *Incendo Online Desktop Installation Guidelines* and is running.
- The terminal you are using (or Incendo Online Terminal Simulator) has already been initialised (for instructions on how to initialise a terminal, refer to the *Incendo Online Desktop Installation Guidelines* or the *Configuration and Initialisation of Incendo Online-enabled Terminals*).

To be able to view and edit the contents of TML pages, you, additionally, have to make sure that:

- A text editor is available on your computer.

^g The path on your computer may, obviously, be different.

MAGCARD application example

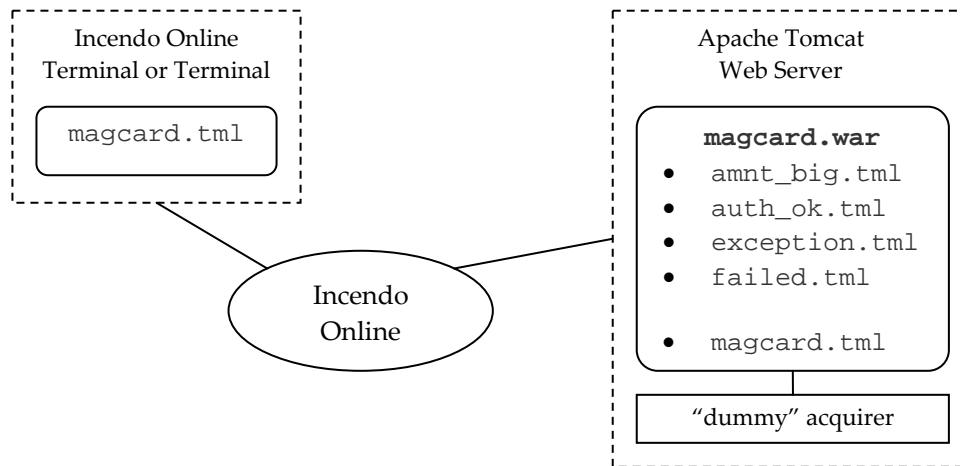
5

There are two types of TML pages that are used by the MAGCARD application example:

- a static TML page that is loaded into the terminal
- dynamic TML pages that are supplied to the terminal by the host, as the result of terminal requests

The terminal-side module is implemented as a single static TML page `magcard.tml`. This page is downloaded by a terminal from the server when a terminal user selects the **Go to Homepage** link in the **Incendo Online MicroBrowser** (start-up) screen.

Figure 5 - 1: MAGCARD application example



There are two separate sets of TML resources for different terminal models: one for *Ingenico 5100* terminals and the other – for *Ingenico 8550*. Selection of the resource version appropriate for the terminal model being used is performed by the `RequestFilter` Java servlet of the host module of MAGCARD application example.

In this chapter, the implementation for *Ingenico 5100* is discussed.

Note: If you are using Incendo Online Terminal Simulator, to be able to see the screens corresponding to the code described in this chapter, switch it into *Ingenico 5100* emulation mode. For instructions on how to do that, refer to the *Incendo Online Desktop Installation Guidelines*.

Application header and variable declarations

The page `magcard.tml` begins with the root TML element that defines the default XML namespace for the elements used on the page.

Figure 5 - 2: magcard.tml application header

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<tml xmlns="http://www.ingenico.co.uk/tml">

    <head>
        <defaults menu="/5100/index.tml" cancel="#init_prompt" />
    </head>
    <!-- Variable for storing result of pam_test -->
    <vardcl name="pam.result" type="string" perms="rwx--" />
    <vardcl name="pam.error_reason" type="string" perms="rwx--" />
    <!-- transaction id -->
    <vardcl name="transid" type="integer" perms="rw---" />
```

The application header following the TML declaration includes `<defaults>` element to define target screens for **Cancel** or **Menu** buttons. Thus, pressing the **Cancel** button will transfer the user to the **Initiate Transaction** application block. Pressing the **Menu** button will switch to the examples selection menu page `index.tml`.

The `<defaults>` element which appears within the `<head>` element has the page-wide scope. You can redefine the target screens for individual screens by means of the corresponding attributes of the `<screen>` element.

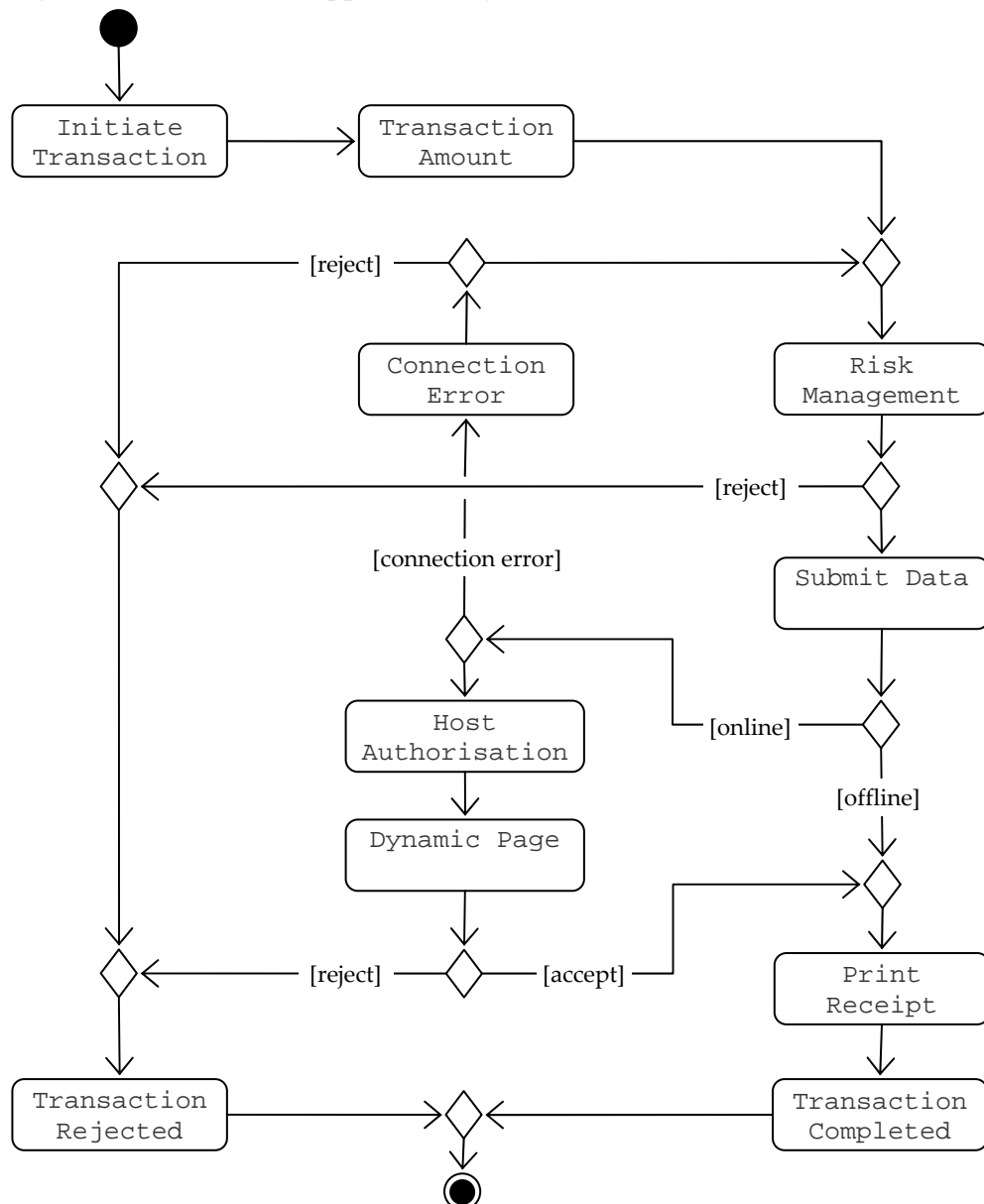
After the header, several variables are declared:

- `pam.result` and `pam.error_reason`
These variables are used for demonstrating the third party application connectivity.
- `transid`
This variable is used to set a unique transaction ID

Application logic

The MAGCARD application layout and user interface follow the APACS guidelines for authorisation of a SALE transaction for magnetic stripe cards (see Figure 5 - 3 below).

Figure 5 - 3: MAGCARD application logic



MAGCARD application example reads the data from the user's magnetic stripe card, accepts the transaction amount entered by the user, submits the data to the host side for transaction authorisation and finally, if the transaction has been approved, prints the receipt.

A flow diagram of the MAGCARD Application Example is shown in the [Figure 5 - 3 on page 29](#).

Each of the application blocks, shown on the diagram is discussed in the following sections. The general purpose of the application block is presented first; and then the specific implementation. The TML screens, used by the application, are referenced by their names defined by the `id` attributes of the `screen` elements.

Initiate Transaction block

The **Initiate Transaction** block is responsible for application initialisation and reading the data from a card. The application prompts the user to swipe a card, initialises a `mag` card parser which waits for the card swiping. If the card has been read successfully, the parser fills predefined card- and `mag` parser-specific variables (see *TML Application Development Guidelines* for details) and the application switches to the **Transaction Amount** block (see [page 31](#)).

This functionality is implemented using the command

```
<card parser="mag" parser_params="read_data"/>
```

Only one TML screen is used - `init_prompt`.

init_prompt screen

This screen implements the **Initiate Transaction** block functionality.

`<tform>` tag is used to accept the user input – in this case the swiping of the card.

The `<prompt>` tag defines the message that will be displayed while the parser waits for the card swipe.

The actual initialisation is performed using the `read_data` function of the `mag` parser:

```
<card parser="mag" parser_params="read_data"/>
```

When this command is initiated, the parser waits for a card swipe. Once a card has been swiped, the parser reads the card data from the magnetic stripe and fills the predefined variables. If there has been an error during the reading of the stripe, or during assigning of the variables, an error message is shown.

This message is defined by the `<baddata>` tag. The parser will automatically fill the `err.baddata_reason` variable with the error message.

If the user presses the **Cancel** button, the application switches to the application selection page `index.tml` that will prompt to choose an example application from the list.

By default, the application suggests online transaction authorisation by setting the variable `card.parser.verdict` to `"online"`.

Figure 5 - 4: #init_prompt screen code

```
<screen id="init_prompt" class="c_center" cancel="/index.tml"
next="#read_amount">
  <setvar name="card.parser.verdict" lo="online"/>
  <tform>
    <baddata class="c_center">
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          <getvar name="err.baddata_reason"/>
        </td></tr>
      </table>
    </baddata>
    <card parser="mag" parser_params="read_data"/>
  </tform>
</screen>
```

```
<table height="100%" width="100%">
  <tr><td align="center" valign="middle">
    Incendo Online<br/>
    Mag Cards Processing Example<br/>
    Please Swipe Card
  </td></tr>
</table>
</prompt>
</tform>
</screen>
```

The next application block is the **Transaction Amount**.

Transaction Amount block

This application block begins the actual transaction. The two main functions of this application block are to get the user to enter the transaction amount, and to set the unique transaction ID. **Transaction Amount** block can also be used to select the transaction type. However, our application example assumes that the transaction is of a generic **Sale** type.

The transaction amount, entered by the user, is assigned to the `payment.amount` variable. This is one of the predefined variables, used by the `mag` parser.

`transid` variable was declared at the start of the `magcard.tml` file (see [“Application header and variable declarations” on page 28](#)). This variable is used as a transaction ID by the MAGCARD Application Example. It is assigned the value of a predefined variable `oebr.unique_id`. This variable is automatically incremented by the MicroBrowser every time a transaction is processed. It is better to use a locally declared variable for the transaction ID, in case the transaction must be reversed or cancelled.

The next step in the application flow is the **Risk Management** block (see [page 32](#)).

Only one TML screen is used by the Transaction Amount application block – `read_amount`.

read_amount screen

This screen displays a simple form for entering the amount of money using the `<display>` and `<form>` tags.

`<input>` command is used to accept the user input. The data is assigned to the `payment.amount` variable, set by the command's `name` parameter.

The command

```
<setvar name="transid" lo="tmlvar:oebr.unique_id"/>
```

sets the `transid` variable to a unique value, which will be used as a transaction ID.

Figure 5 - 5: #read_amount screen code

```
<screen id="read_amount" next="#mag_rm">
<!-- Increase transaction counter -->
  <setvar name="transid" lo="tmlvar:oebr.unique_id"/>
  <display>
    <form>
      <table height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Amount:<br/>
          <input alt="Amount:" type="number"
name="payment.amount" size="10" format="^*0.00"/>
        </td></tr>
      </table>
    </form>
  </display>
</screen>
```

The next application block is the **Risk Management**.

Risk Management block

Risk Management block performs the risk management assessment and decides whether the transaction should be processed online, offline or rejected.

MAGCARD application example uses only the `risk_mgmt` function of the `mag` parser to perform the risk assessment:

```
<card parser="mag" parser_params="risk_mgmt"/>
```

The parser does the following:

- analyses the transaction amount and transaction type
- checks the previous submission attempt (`oebr.econn` value)
- updates `card.parser.verdict` variable
- fills `card.parser.reject_reason` variable, if necessary.

Your application could employ additional risk management.

As the result of the risk management process, the `card.parser.verdict` variable should be assigned one of the following values:

- "online"
The transaction should be authorised online.
- "offline"
The transaction may be processed offline.
- "reject"
The transaction should be rejected. `card.parser.reject_reason` variable should be filled with the explanation for the rejection.

If the transaction is rejected, the application proceeds to the **Transaction Rejected** block (see page 41). Otherwise, the next step is the **Submit Data** application block (see page 33).

The **Risk Management** block of the MAGCARD Application Example uses only the `mag_rm` screen.

mag_rm screen

This screen performs the function of the **Risk Management** application block by using the `risk_mgmt` command of the `mag` parser. It is executed from within the `<tform>` tags.

The `<variant>` tag defines the TML screens the application will go to depending on risk management outcome.

Figure 5 - 6: #mag_rm screen code

```
<screen id="mag_rm">
  <next uri="#mag_submit">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
      ro="reject" uri="#reject_trans"/>
  </next>
</tform>
  <card parser="mag" parser_params="risk_mgmt"/>
</tform>
</screen>
```

The application will continue to either the **Submit Data** block (`mag_submit` screen) or to the **Transaction Rejected** block (`reject_trans` screen).

Submit Data block

The purpose of the **Submit Data** program block is to submit to the host the variables necessary for the transaction processing.

Our application example assumes that the following information is required for the magnetic card transaction processing:

- `card.parser.type`
- `card.cardholder_name`
- `card.pan`
- `card.issuer_name`
- `card.issue_number`
- `card.scheme`
- `card.effective_date`
- `card.expiry_date`
- `card.mag.iso2_track`
- `transid`
- `payment.amount`

As discussed in the [“Risk Management block” on page 32](#), risk management verdict is contained in the `card.parser.verdict` variable. It can be one of the following:

- `"offline"`
- `"online"`
- `"reject"`

If the risk management verdict was `"online"`, host authorisation is required. Therefore, the submission must be done online and the next step of the application will be the **Host Authorisation** block (see page 35).

If the risk management verdict was `"offline"`, authorisation is not necessary. The host can process the transaction later and the submission can be done offline.

If the risk management verdict was `"reject"`, the transaction was rejected, and there is no need to submit data to host.

Data submission to the host is done using the `<submit>` TML command. It generates a form that will be sent to the host module component that is responsible for accepting terminal requests. This component is specified by the `tgt` attribute of the `<submit>` tag. The page that the application will go to in case of a connection error is specified by the `econn` attribute.

Note: for more information on the `<submit>` command, see *TML Application Development Guidelines*.

Incendo Online Microbrowser variable that is responsible for setting the submit mode is `oebr.submit_mode`. It is set to `"online"` if the submission must be done online, and `"offline"` if otherwise. The easiest way to do that is to set it to the value of the risk management verdict:

```
<setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict"/>
```

In case of the online submission, application proceeds to the **Host Authorisation** block (see page 35).

If the submission is to be done offline, **Print Receipt** block is next (see page 37).

If there is a connection error during the submission, the MAGCARD application proceeds to the **Connection Error** block (see page 34).

Submit Data block consists of the `mag_submit` screen.

mag_submit screen

The path to the host and the screen that is accessed in case of the connection error are set by the attributes of the <submit> tag:

```
<submit tgt="/iccmv/authtxn" econn="#icc_conn_fld">
```

In our application example, the path "/iccmv/authtxn" sends the data to the MagCardTxnProcessingServlet which emulates the interaction with the acquirer.

MagCardTxnData works with the data structure that was sent by the <submit> command. check_err screen is part of the **Connection Error** block (see page 34).

Figure 5 - 7: #mag_submit screen code

```
<screen id="mag_submit" next="#receipt">
  <setvar name="oebr.submit_mode"
  lo="tmlvar:card.parser.verdict"/>
  <submit tgt="/magcard/authtxn" econn="#check_err">
    <getvar name="card.parser.type"/>
    <getvar name="card.cardholder_name"/>
    <getvar name="card.pan"/>
    <getvar name="card.issuer_name"/>
    <getvar name="card.issue_number"/>
    <getvar name="card.scheme"/>
    <getvar name="card.effective_date"/>
    <getvar name="card.expiry_date"/>
    <getvar name="card.mag.iso2_track"/>
    <getvar name="transid"/>
    <getvar name="payment.amount"/>
  </submit>
</screen>
```

Connection Error block

This application block deals with the connection errors that occurred during the execution of the **Submit Data** application block (see page 33).

If the connection error occurred during the offline submission, it means that the offline transaction pool is full. Our application example assumes that if the offline transaction pull is full, the transaction should be rejected. The application proceeds to the Transaction rejected block (see page 41).

If the connection error occurred during the online submission, application control is transferred back to the **Risk Management** block, which decides whether the online submission should be repeated, or an offline submission can be performed (see page 32). If an online submission is required, but it can not be done because of the connection errors, **Risk Management** block should reject the transaction.

Connection Error block consists of the two TML screens:

- check_err screen
- pool_full screen

check_err screen

This screen transfers control to either the pool_full screen if the connection error occurred during the offline submission; or to the **Risk Management** block (mag_rm screen) otherwise.

Figure 5 - 8: #check_err screen code

```
<screen id="check_err">
  <next uri="#mag_rm">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
  ro="offline" uri="#pool_full" />
    <variant lo="tmlvar:oebr.submit_mode" op="equal"
  ro="offline" uri="#pool_full" />
  </next>
</screen>
```

pool_full screen

This screen displays an error message and transfers the application control to the `void_trans` screen, part of the **Transaction Rejected** block (see page 41).

Figure 5 - 9: #pool_full screen code

```
<screen id="pool_full" class="c_center" next="#void_trans">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Transactions offline pool is full and connection to
server fails. Transaction will be rejected.
      </td></tr>
    </table>
  </display>
</screen>
```

Host Authorisation block

The purpose of the **Host Authorisation** program block is to conduct online authorisation. The terminal application host-side module should do the following:

1. parse the TML data sent by the **Submit Data** block (see page 33)
2. generate an appropriate authorisation request and send it to the card acquirer
3. process the response from the acquirer
4. send an appropriate dynamic TML page to the terminal, depending on the acquirer's verdict.

In our MAGCARD Application Example, two Java servlets are used to emulate this process:

- `MagCardTxnData` that works with the data structure that was sent by the **Submit Data** block, and
- `MagCardTxnProcessingServlet` that processes the information

`MagCardTxnProcessingServlet` emulates the interaction with the acquirer, and does the following:

- parses the TML post request.
- checks the card expiry date against the current date.
- checks the transaction amount against the defined top limit.
- selects the dynamic page that will be sent to the terminal.

The function of the dynamic TML page is discussed in the [“Dynamic Page block”](#) below.

Dynamic Page block

The dynamic TML pages described in this section are sent by the host module of MAGCARD to the terminals in response to their transaction authorisation requests.

The purpose of these pages is to inform the user of the results of the Host Authorisation, and to redirect the program to the **Print Receipt** block if the authorisation has been successful, or to the **Transaction Rejected** block if the transaction has been denied.

The pages that are sent to the terminal are:

- if the host approves the transaction, `auth_ok.tml` is used (see page 36)
- if the transaction amount exceeds the upper limit, `amnt_big.tml` is sent (see page 36)
- if the transaction has been declined, `failed.tml` is utilised (see page 36)

***auth_ok.tml* page**

This page is sent to the terminal when the transaction has been approved.

This is a complete TML page, and not just a <screen> element. Therefore, the page contains the <?xml?> header and the <tml> element.

Notice that the cache parameter of the <tml> tag is set to "deny". This instructs the MicroBrowser not to cache this TML page within the terminal.

The page displays a Transaction Approved message for a brief time. The time is set by the timeout parameter of the <screen> element, and the message is contained within the <display> tags.

Figure 5 - 10: auth_tml page code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="dynamic" class="c_center" timeout="1"
next="/magcard/magcard.tml#receipt">
    <display>
      <table height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Transaction Approved
        </td></tr>
      </table>
    </display>
  </screen>
</tml>
```

Once the message is displayed for a required amount of time, the application proceeds to the receipt screen of the magcard.tml page. This is defined by the next parameter of the <screen> element. The receipt screen is contained within the **Print Receipt** application block (see page 37).

***amnt_big.tml* page**

This page is sent by the host if the transaction amount exceeds a predefined top limit.

Figure 5 - 11: amnt_big.tml page code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="dynamic" class="c_center" timeout="1"
next="/magcard/magcard.tml#void_trans">
    <display>
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Amount Exceeds Top Limit
        </td></tr>
      </table>
    </display>
  </screen>
</tml>
```

The application proceeds to the void_trans screen of the magcard.tml file. The screen is contained within the **Transaction Rejected** program block (see page 41).

Note: see "[auth_ok.tml page](#)" above for a more detailed description of a dynamic page sent by the host.

***failed.tml* page**

This page is sent by the host if it rejected the transaction.

Figure 5 - 12: failed.tml page code

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="dynamic" class="c_center" timeout="1"
next="/magcard/magcard.tml#void_trans">
    <display>
      <table border="2" height="100%" width="100%">
```

```

        <tr><td align="center" valign="middle">
            Not Authorized
        </td></tr>
    </table>
</display>
</screen>
</tml>

```

The application proceeds to the `void_trans` screen of the `magcard.tml` file. The screen is contained within the **Transaction Rejected** program block (see page 41).

Note: see [“auth_ok.tml page” on page 36](#) for a more detailed description of a dynamic page sent by the host.

Print Receipt block

The **Print Receipt** application block is responsible for printing the receipt.

It is assumed that the receipt will only be printed if the transaction has been authorised.

MAGCARD application example provides two different ways of printing a receipt:

- using an internal terminal printer
- using an external printer

The external print function shows an example of integrating a third-party terminal application (the interface with an external printer) within a TML program.

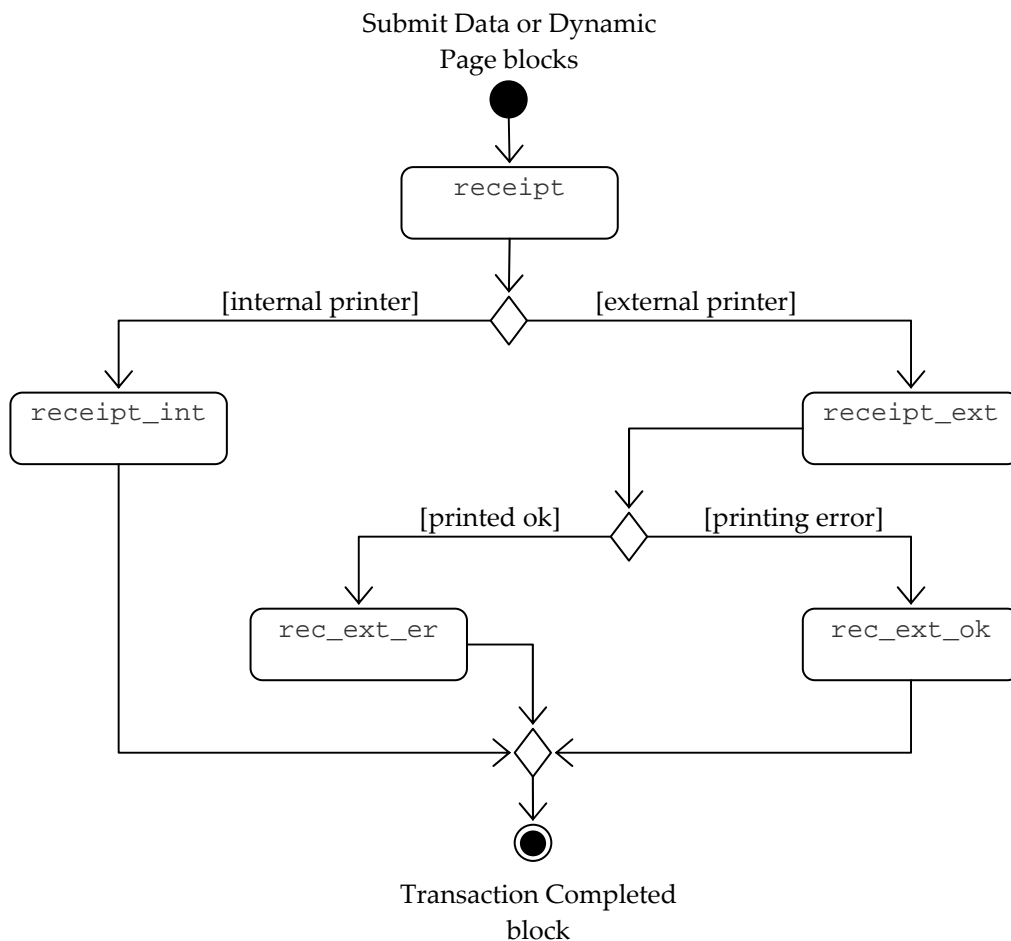
This is done using the

```
<card parser="pam" parser_params="read_data" />
```

command. For more details, see [“receipt_ext screen” on page 39](#) and *Technical Note 3: Integrating Incendo Online with third-party terminal applications*.

The application logic for the **Print Receipt** block is shown in the [Figure 5 - 13 below](#).

Figure 5 - 13: application logic of the Print Receipt block



Note: after the receipt is printed, a signature check can be conducted. To do that, the customer should be asked to sign the receipt. If the signature does not match the example on the card, the transaction should be cancelled. However, this functionality is not implemented in the MAGCARD application example.

receipt screen

This screen queries the user whether the receipt should be printed using an internal terminal printer, or whether an external printer should be used.

The screen uses hyperlinks (`<a>` tags) to transfer the user to the:

- `receipt_int` screen if the user wishes to use the internal printer (see page 38)
- `receipt_ext` screen if the user wants to use the external printer (see page 39)

Figure 5 - 14: #receipt screen code

```
<screen id="receipt" next="#receipt_int">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Print receipt on external printer?<br />
        <a href="#receipt_ext">Yes</a>
        <a href="#receipt_int">No</a>
      </td></tr>
    </table>
  </display>
</screen>
```

receipt_int screen

The screen prints a simple receipt using the internal terminal printer.

This screen uses the `<print>` TML tag to print some basic information. The following information is printed:

- Current time
the value is taken from the `terminal.datetime` variable, which is updated by the terminal.
- Card number
`card.pan` variable was filled earlier by the `mag` parser. This was done during the execution of the **Initiate Transaction** block (see page 30)
- Transaction ID
`transid` variable set earlier
- Payment amount
the value of the `payment.amount` variable was entered by the user during the execution of the **Transaction Amount** block (see page 31)

Figure 5 - 15: #receipt_int screen code

```
<screen id="receipt_int" next="#compl_trans">
  <print>
    <table>
      <tr>
        <td>Date:</td>
        <td>
          <getvar name="terminal.datetime" />
        </td>
      </tr>
      <tr>
        <td>Card:</td>
        <td>
          <getvar name="card.pan" format="n4n#*n4" />
        </td>
      </tr>
    </table>
  </print>
</screen>
```

```

        <td>Transaction ID:</td>
        <td>
            <getvar name="transid" />
        </td>
    </tr>
    <tr>
        <td>Amount:</td>
        <td>
            <getvar name="payment.amount" format="^*0.00" />
        </td>
    </tr>
</table>
<br />
-----
<br /><br /><br />
</print>
</screen>

```

The application then proceeds to the `compl_trans` screen of the **Transaction Completed** block (see page 40).

***receipt_ext* screen**

This screen uses a third-party application to print a receipt on an external printer.

The name of the third-party application is assigned to `oebr.3rdparty.app_name`:

```
<setvar name="oebr.3rdparty.app_name" lo="GB000400_Prn" />
```

The name `GB000400_Prn` was assigned to that application by the *IngEstate*.

Several parameters are passed to using the `oebr.3rdparty.var_list` pre-defined variable:

```
<setvar name="oebr.3rdparty.var_list"
lo="terminal.datetime;card.pan;transid;payment.amount" />
```

This information will be included in the receipt.

Once `oebr.3rdparty.app_name` and `oebr.3rdparty.var_list` variables are filled with the required information

```
<card parser="pam" parser_params="read_data" />
```

is executed.

`GB000400_Prn` should process the information, passed to it by the `oebr.3rdparty.var_list` pre-defined variable, and print a receipt.

Figure 5 - 16: #receipt_ext screen code

```

<screen id="receipt_ext">
    <setvar name="pam.result" lo="error" />
    <setvar name="pam.error_reason" lo="No reply from external
printer" />
    <setvar name="oebr.3rdparty.app_name" lo="GB000400_Prn" />
    <setvar name="oebr.3rdparty.var_list"
lo="terminal.datetime;card.pan;transid;payment.amount" />
    <next uri="#rec_ext_ok">
        <variant lo="tmlvar:pam.result" op="equal" ro="error"
uri="#rec_ext_er" />
    </next>
    <tform>
        <baddata class="c_center">
            <table border="2" height="100%" width="100%">
                <tr><td align="center" valign="middle">
                    <getvar name="err.baddata_reason" />
                </td></tr>
            </table>
        </baddata>
        <card parser="pam" parser_params="read_data" />
        <prompt>
            <table height="100%" width="100%">
                <tr><td align="center" valign="middle">

```

```
        Printing ...<br />
      </td></tr>
    </table>
  </prompt>
</tform>
</screen>
```

rec_ext_er screen

This screen is used if there have been an error during the external printing.

Figure 5 - 17: #rec_ext_er screen code

```
<screen id="rec_ext_er" next="#compl_trans">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Error while printing:<br />
        <getvar name="pam.error_reason" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

The next step is the `compl_trans` screen of the **Transaction Completed** block (see page 40).

rec_ext_ok screen

This screen is used if the receipt was printed successfully on an external printer.

Figure 5 - 18: #rec_ext_ok screen code

```
<screen id="rec_ext_ok" next="#compl_trans">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Printed ok <br />
      </td></tr>
    </table>
  </display>
</screen>
```

The next step is the `compl_trans` screen of the **Transaction Completed** block (see page 40).

Transaction Completed block

This application block is reached if the transaction has been completed successfully. The user is informed that the transaction has been completed. The application returns back to the **Initiate Transaction** block (see page 30).

compl_trans screen

This screen displays the `Transaction Completed` message for a brief period, and then transfers application control to the **Initiate Transaction** block (`init_prompt` screen).

The length of time the message is displayed is set by the `timeout` attribute of the `<screen>` element.

Figure 5 - 19: #compl_trans screen code

```
<screen id="compl_trans" class="c_center" timeout="1"
next="#init_prompt">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">Transaction
Completed
      </td></tr>
    </table>
  </display>
```

</screen>

Transaction Rejected block

If the transaction is rejected, the application should do the following:

- notify the user of the rejection and give the rejection reason,
- if the transaction has already been processed, void the transaction

If the transaction has been rejected in one of the previous application blocks, `card.parser.verdict` variable was set to "reject" and `card.parser.reject_reason` was filled with the explanation of the rejection.

Therefore, you can simply display the `card.parser.reject_reason` variable to inform the user. (see "[reject_trans screen](#)" below).

The application should also void the transaction, if it has been processed. Our basic application does not implement this functionality. It assumes that the transaction was rejected or cancelled before transaction data had a chance to be processed, and simply informs the user that the transaction has been declined (see "[void_trans screen](#)" below).

Once the **Transaction Rejected** block has been processed, the application returns to the **Initiate Transaction** block (see page 30).

reject_trans screen

The screen displays the message that the transaction is rejected mentioning the reason set by the magcard parser. Then the application proceeds to the `void_trans` screen.

Figure 5 - 20: #reject_trans screen code

```
<screen id="reject_trans" class="c_center" next="#void_trans">
<display>
<table height="100%" width="100%">
<tr><td align="center" valign="middle">
<getvar name="card.parser.reject_reason"/>
</td></tr>
</table>
</display>
</screen>
```

void_trans screen

The screen displays a message that the transaction has been declined and switches to the Initiate Transaction block (`init_prompt` screen). If the transaction has been rejected by the host side, this screen will be displayed after either the `amnt_big.tml` or `failed.tml` (see "[Dynamic Page block](#)" on page 35)

Figure 5 - 21: #void_trans screen code

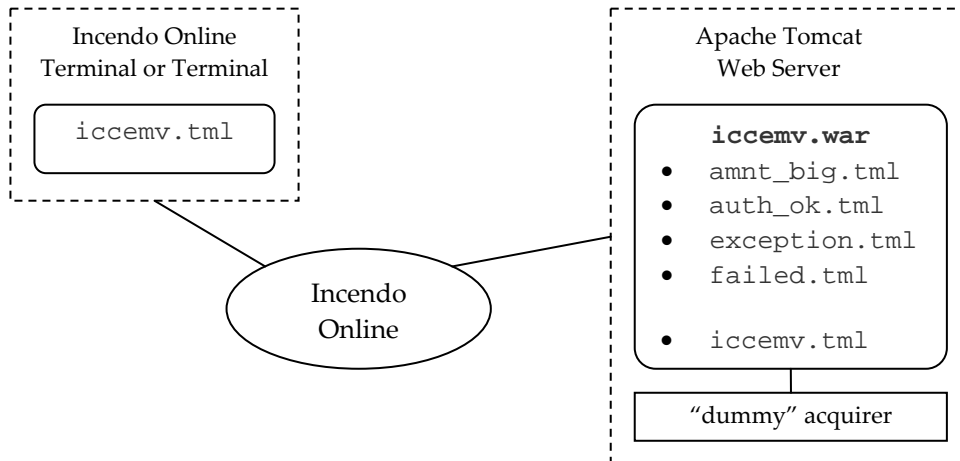
```
<screen id="void_trans" class="c_center" timeout="1"
next="#init_prompt">
<display>
<table height="100%" width="100%">
<tr><td align="center" valign="middle">Transaction
Declined
</td></tr>
</table>
</display>
</screen>
```

ICCEMV application example

6

The terminal-side module is implemented as a single static TML page `iccemv.tml`. This page is downloaded by a terminal from the server when a terminal user selects the **Go to Homepage** link in the **Incendo Online MicroBrowser** (start-up) screen.

Figure 6 - 1: ICCEMV application example



There are two separate sets of TML resources for different terminal models: one for *Ingenico 5100* terminals and the other – for *Ingenico 8550*. Selection of the resource version appropriate for the terminal model being used is performed by the `RequestFilter` Java servlet of the host module of ICCEMV application example. In this chapter, the implementation for *Ingenico 5100* is discussed.

Note: If you are using Incendo Online Terminal Simulator, to be able to see the screens corresponding to the code described in this chapter, switch it into *Ingenico 5100* emulation mode. For instructions on how to do that, refer to the *Incendo Online Desktop Installation Guidelines*.

Application header and variable declarations

The TML page begins with the xml declaration and the root TML element that defines the namespace for the elements used on the page.

Figure 6 - 2: ICCEMV TML header and variable declarations

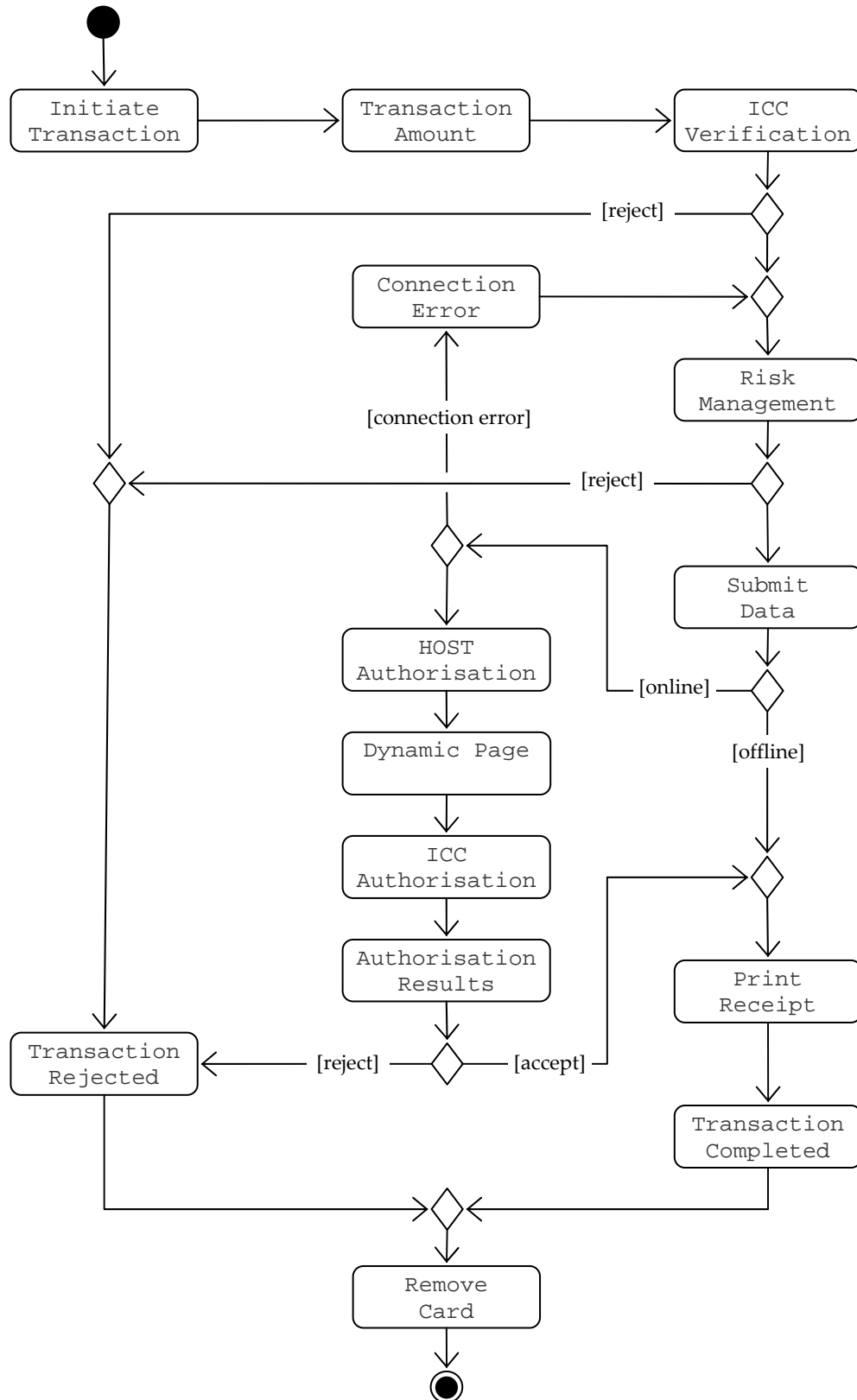
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tml xmlns="http://www.ingenico.co.uk/tml">
  <head>
    <defaults menu="/5100/index.tml" cancel="#init_prompt"/>
  </head>
  <!-- transaction id -->
  <vardcl name="transid" type="integer" perms="rw---"/>
```

The application header following the TML declaration includes the `<defaults>` element. The element specifies that by default, pressing the **Cancel** button while browsing the page will switch the user to the `init_prompt` screen, which corresponds to the **Initiate Transaction** application block (page 44). The default action for pressing the **Menu** button is to switch to the application examples selection page `index.tml`. Additionally, a variable `transid` is declared for storing the transaction ID information.

Application logic

The application logic implemented by the `iccmv.tml` page is shown on this diagram:

Figure 6 - 3: ICCEMV application logic



Initiate Transaction block

The purpose of this application block is to initialise the ICC once a card is inserted into the terminal. This is done in two steps:

- wait for card insertion and start ICC initialisation - `init_prompt` screen
- select the ICC application that will be used for the transaction - `final_init` screen

ICC is initialised by the following code:

```
<card parser="icc_emv" parser_params="init_app"/>
```

As the result the parser will fill the following TML variables:

- `card.cardholder_name`
- `card.effective_date`
- `card.expiry_date`
- `card.scheme`
- `card.pan`
- `card.pan_pr`
- `card.input_type`
- `card.parser_type`
- `card.emv.app_pan_seq`
- `card.emv.aid`
- `card.emv.aip`
- `card.emv.auc`
- `card.emv.iac_default`
- `card.emv.iac_denial`
- `card.emv.iac_online`
- `card.emv.iad`
- `card.emv.iso_track2`

Once ICC is initialised, an appropriate ICC application must be selected. This is done by first setting the `card.emv.selected_app` variable to a particular value and then using the command

```
<card parser="icc_emv" parser_params="final_select"/>
```

ICC application selection can be either manual or automatic.

Our ICCEMV application example implements only automatic application selection, thus `card.emv.selected_app` variable is set to `" "`.

Note: the rest of the transaction process requires access to the ICC. Thus, the card should remain in the card reader. If the card is removed from the card reader, the parser interrupts the command processing and assigns `"reject"` to the `card.parser.verdict` variable. `card.parser.reject_reason` is set to the `"The ICC is removed"`.

Once **Initiate Transaction** screens are processed, the application then goes to the **Transaction Amount** block (see page [45](#)).

init_prompt screen

This is the initial application screen. The application initialises the `icc_emv` parser and prompts a user to insert an ICC card.

Then application proceeds to the `final_init` screen.

If a user presses the **Cancel** button, the application switches to the application examples selection page `index.tml`.

If there is an error, an error screen is displayed and the control passes to the **Remove Card** block (see page [59](#)).

Figure 6 - 4: #init_prompt screen code

```

<!-- Initial screen: prompt for merchant -->
<screen id="init_prompt" class="c_center"
cancel="/5100/index.tml" next="#final_init">
  <tform>
    <baddata class="c_center" max="1" next="#remove_card">
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          <getvar name="err.baddata_reason"/>
        </td></tr>
      </table>
    </baddata>
    <card parser="icc_emv" parser_params="init_app"/>
    <prompt>
      <table height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Incendo Online ICCEMV 2.0.2.6<br/>
          Smartcards Processing Example<br/>
          Please Insert Card
        </td></tr>
      </table>
    </prompt>
  </tform>
</screen>

```

final_init screen

This screen is used to select the ICC application that will be used for the transaction. This done by setting the value of the `card.emv.selected_app` variable to "" (automatic selection) and then calling the card parser:

```
<card parser="icc_emv" parser_params="final_select"/>
```

The application then goes to the **Transaction Amount** block.

Figure 6 - 5: #final_init screen code

```

<!-- application selection (automatic) -->
<screen id="final_init" next="#read_amount">
  <setvar name="card.emv.selected_app" lo="" />
  <tform>
    <baddata class="c_center" max="1" next="#remove_card">
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          <getvar name="err.baddata_reason"/>
        </td></tr>
      </table>
    </baddata>
    <card parser="icc_emv" parser_params="final_select"/>
  </tform>
</screen>

```

Transaction Amount block

This program block is responsible for setting the unique transaction ID and getting the transaction amount information from the user. The transaction amount will be used during the Risk Management process; therefore it must be acquired before that application step. However, this information is not required for ICC Verification. **Transaction Amount** block can be either before or after it.

It contains only the `read_amount` screen.

The next application block is the **ICC Verification** (see page 46).

read_amount screen

This screen sets the `transid` variable to the unique transaction ID. The value of the `oebr.unique_id` is set by the MicroBrowser.

The user is asked for the transaction amount and user input is assigned to the pre-defined variable `payment.amount`.

Figure 6 - 6: #read_amount screen code

```
<screen id="read_amount" next="#get_cvm">
<!-- Increase transaction counter -->
  <setvar name="transid" lo="tmlvar:oebr.unique_id"/>
  <display>
    <form>
      <table height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Amount:<br/><input alt="Amount:" type="number"
name="payment.amount" size="10" format="^^*0.00"/>
        </td></tr>
      </table>
    </form>
  </display>
</screen>
```

Application control is then passed to the **ICC Verification** program block.

ICC Verification block

The purpose of this application block is to verify the card and the user. ICC goes through the applicable cardholder verification methods (CVM). The cardholder verification methods are obtained by using `get_cvm` command of the `icc_emv` parser.

Method suggestions are put into the `card.parser.cvm` variable. Its possible values are:

- `"pin_online"` - PIN should be sent to the host for verification.
- `"pin"` - PIN should be verified offline by the ICC.
- `"no_cv"` - no cardholder verification is required
- empty string (`" "`) - no more cardholder verification methods are available. This result means that all CVM checks have been made and the application can progress to the Risk Management stage.

One of the card verification methods requested by the parser could be signature verification. In this case `icc_emv` parser will set the value of the `card.emv.signature` variable to 1. Signature verification will have to be done towards the end of the transaction process, after the receipt has been printed. Remember that the card must be removed from the terminal before the signatures can be verified. This basic application example does not implement this verification method.

When a verification method is executed, its outcome (cardholder verification result) is put into the `card.parser.cvr` variable. This variable can be filled by the TML application or be updated by the card parser. It can have the following values:

- `"bypassed"` – the method is not supported by the TML application
- `"ok"` – verification done by this method was a success
- `"ok_msg"` – same as `"ok"`, but additionally `"PIN OK"` message should be displayed
- `"failed"` – verification was unsuccessful
- `"pin_tries"` – wrong PIN was entered several times, and the number of tries exceeds PIN try limit

Their results are used during the Risk Management process.

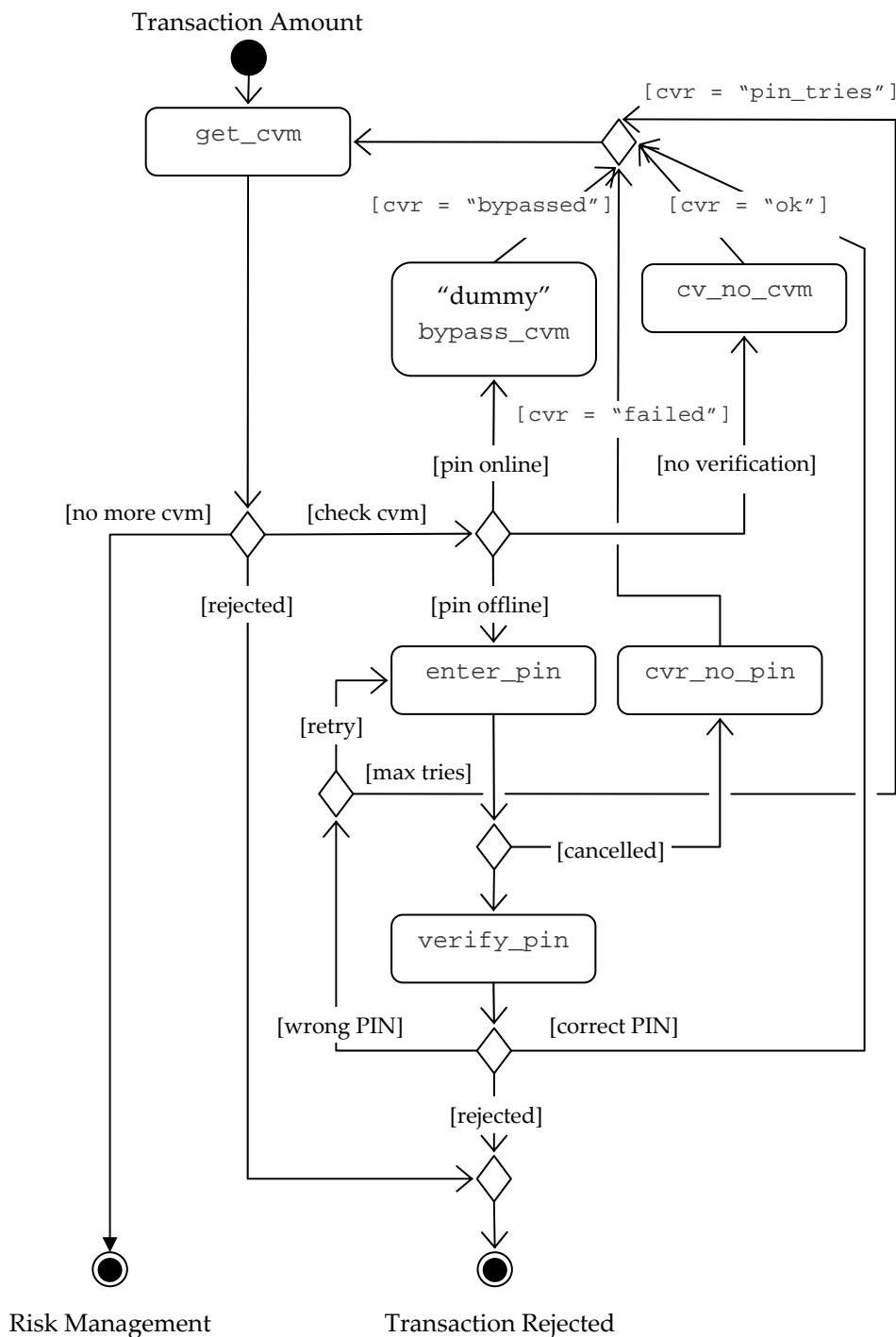
ICC Authorisation block contains the following screens:

- `get_cvm` ([on page 47](#))
- `enter_pin` ([on page 48](#))

- [verify_pin \(on page 49\)](#)
- [bypass_cvm \(on page 49\)](#)
- [cv_no_cvm \(on page 50\)](#)
- [cvr_no_pin \(on page 50\)](#)
- [assert \(on page 50\)](#)

When ICC authorisation is completed, the program proceeds to either the **Risk Management** block (see page 50) or the **Transaction Rejected** block (see page 57).

Figure 6 - 7: application logic of the ICC Authorisation program block



get_cvm screen

The screen requests the ICC parser for the appropriate cardholder verification method:

```
<card parser="icc_emv" parser_params="get_cvm"/>
```

The parser responds by assigning a value to the predefined variable `card.parser.cvm`. The parser might do the following:

- reject the transaction – the value of the `card.parser.verdict` is set to "reject"
This happens if the card was removed from the card reader before transaction is completed. The application moves to the **Transaction Rejected** block (see page 57)
- ask for offline pin verification - `card.parser.cvm` is set to "pin"
The application proceeds to the `enter_pin` screen (described below)
- report that the PIN should be verified online - `card.parser.cvm` is set to "pin_online"
To keep things simple, basic ICCEMV application examples do not implement online pin verification. Instead, the next step is the `bypass_cvm` screen (described on page 49)
- report that no cardholder verification required - `card.parser.cvm` is set to "no_cv"
The application moves to the `cv_no_cvm` screen (described on page 50)
- report that no more cardholder verification methods are available
This result from the `icc_emv` parser means that all cardholder verification methods have been completed, and the application can proceed to the **Risk Management** block (see page 50)

Figure 6 - 8: #get_cvm screen code

```
<screen id="get_cvm">
  <next uri="#assert">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="#reject_trans"/>
    <variant lo="tmlvar:card.parser.cvm" op="equal" ro="pin"
uri="#enter_pin"/>
    <variant lo="tmlvar:card.parser.cvm" op="equal"
ro="pin_online" uri="#bypass_cvm"/>
    <variant lo="tmlvar:card.parser.cvm" op="equal" ro="no_cv"
uri="#cv_no_cvm"/>
    <variant lo="tmlvar:card.parser.cvm" op="equal" ro=""
uri="#risk_mgmt"/>
  </next>
<tform>
  <card parser="icc_emv" parser_params="get_cvm"/>
</tform>
</screen>
```

The application returns to this screen repeatedly until all the cardholder verification methods have been processed, or the card is removed from the card reader.

An unexpected outcome would indicate bad program design, and application control will pass to the `assert` screen (see page 50).

enter_pin screen

The screen displays a simple form for entering the PIN. This is done using the command

```
<pinentry type="icc" prompt="Enter PIN"/>
```

The application then proceeds to the `verify_pin` screen.

Figure 6 - 9: #enter_pin screen code

```
<screen id="enter_pin" cancel="#cvm_no_pin" next="#verify_pin">
  <tform>
    <baddata class="c_center" max="3" next="#init_prompt">
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          <getvar name="err.baddata_reason"/>
        </td></tr>
      </table>
    </baddata>
    <pinentry type="icc" prompt="Enter PIN"/>
  </tform>
```

```
</tform>
</screen>
```

The user can also cancel the PIN entry. In that case, application accesses the `cvr_no_pin` screen (see page 50).

verify_pin screen

The screen requests the ICC parser to verify the entered PIN. This is done using the command

```
<card parser="icc_emv" parser_params="verify"/>
```

The parser responds by assigning a value to the predefined variables `card.parser.verdict` and `card.parser.cvr`. The parser might:

- reject the transaction - the value of the `card.parser.verdict` is set to "reject"
This happens if the card was removed from the card reader before transaction is completed. The application moves to the **Transaction Rejected** block (see page 57)
- report that the PIN has been verified - `card.parser.cvr` is set to "ok" or "ok_msg"
The application returns to the `get_cvm` screen.
- report that the verification failed - `card.parser.cvr` is set to "failed"
The user is asked to enter the PIN again, from the `enter_pin` screen (see page 48).
- report that the maximum number of tries for entering the PIN is exceeded - `card.parser.cvr` is set to "failed"
`get_cvm` screen is accessed again.

Figure 6 - 10: #verify_pin screen code

```
<screen id="verify_pin">
  <next uri="#assert">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="#reject_trans"/>
    <variant lo="tmlvar:card.parser.cvr" op="equal" ro="ok"
uri="#get_cvm"/>
    <variant lo="tmlvar:card.parser.cvr" op="equal" ro="ok_msg"
uri="#get_cvm"/>
    <variant lo="tmlvar:card.parser.cvr" op="equal" ro="failed"
uri="#enter_pin"/>
    <variant lo="tmlvar:card.parser.cvr" op="equal"
ro="pin_tries" uri="#get_cvm"/>
  </next>
<tform>
  <card parser="icc_emv" parser_params="verify"/>
</tform>
</screen>
```

An unexpected outcome would indicate bad program design, and application control will pass to the `assert` screen (see page 50).

bypass_cvm screen

Online pin verification is not implemented in the current application example. This screen simply sets the card verification result variable `card.parser.cvr` to "bypassed" and transfers control back to the `get_cvm` screen (see page 47).

Figure 6 - 11: #bypass_cvm screen code

```
<screen id="bypass_cvm" next="#get_cvm">
  <setvar name="card.parser.cvr" lo="bypassed"/>
</screen>
```

cv_no_cvm screen

If no cardholder verification is required, verification is assumed to have been passed successfully. Cardholder verification result (`card.parser.cvr`) is set to "ok" and the application returns to the `get_cvm` screen (see page 47).

Figure 6 - 12: #cv_no_cvm screen code

```
<screen id="cv_no_cvm" next="#get_cvm">
  <setvar name="card.parser.cvr" lo="ok"/>
</screen>
```

cvr_no_pin screen

This screen is used if the user for some reason cancelled the PIN entry. In this example, PIN verification is assumed to have been failed.

Figure 6 - 13: #cvr_no_pin screen code

```
<screen id="cvr_no_pin" next="#get_cvm">
  <setvar name="card.parser.cvr" lo="failed"/>
</screen>
```

`card.parser.cvr` variable is set to "failed" and the application returns to the `get_cvm` screen (see page 47).

assert screen

This screen is used to catch programming errors and to improve the TML code. If, for instance, `get_cvm` screen (see page 47) does not take into account all possible outcomes of the cardholder verification, application will use the `assert` screen to notify the user (and the programmer) of the assertion error.

Figure 6 - 14: #assert screen code

```
<screen id="assert" timeout="3" class="c_center"
next="#init_prompt">
  <display>
    <table border="2" height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Assertion Error<br/>
        on screen "<getvar name="oebr.prev_screen"/>"
      </td></tr>
    </table>
  </display>
</screen>
```

Risk Management block

Risk management is performed by the card, the terminal, and the host module in order to protect the system from fraud.

Part of the risk management – cardholder verification, has already been conducted during the execution of the **ICC Verification** block (see page 46).

Risk Management block should analyse the results of the cardholder verification methods. The application should also check the transaction floor limit, and the policies regarding online transaction processing.

The following command asks the `icc_emv` card parser to perform the risk management:

```
<card parser="icc_emv" parser_params="risk_mgmt"/>
```

As the result of this command, the card parser sets the `card.parser.verdict` variable to one of the following values:

- "offline" – the transaction can be processed offline
`card.emv.tc` variable will also be filled
- "online" – the transaction must be authorised online
`card.emv.arqc` variable will also be filled

- "reject" – the transaction is rejected
card.emv.aac will also be filled

Additionally, the parser sets the following variables:

- card.emv.atc
- card.emv.cvmr
- card.emv.iad
- card.emv.tvr
- card.emv.unumber

Obviously, only one of card.emv.aac, card.emv.tc and card.emv.arqc will be non-empty, depending on the risk management verdict.

Risk Management block should also take into account previous submission attempts to host. For example, if the card parser suggests that the transaction should be authorised online, but the connection can not be established, the Risk Management block should decide whether the application should:

- try to submit online again
- process the transaction offline
- reject the transaction

This can be done in conjunction with the host side, using the oebr.econn variable – the reason for the submission failure filled by the HTTP client.

Alternatively, you can use the econn attribute of the <submit> command. This will specify the screen that the application will proceed to in case there is a connection error during the execution of the **Submit Data** block (see ["Connection Error block" on page 53](#)).

Our ICCEMV example application uses the risk_mgmt screen to perform the initial risk management assessment (see page 51).

risk_mgmt screen

The screen requests the ICC parser to perform the risk management evaluation. The parser analyses the amount and transaction type and previous submission attempts for the transaction and responds by assigning a value to the predefined variables card.parser.verdict and card.parser.reject_reason (if necessary). If the evaluation is successful, the application proceeds to **Submit Data** block (see page 51), if not – to the **Transaction Rejected** block (see page 57).

Figure 6 - 15: #risk_mgmt screen code

```
<screen id="risk_mgmt">
  <next uri="#submit">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="#reject_trans"/>
  </next>
  <tform>
    <card parser="icc_emv" parser_params="risk_mgmt"/>
  </tform>
</screen>
```

Submit Data block

The purpose of the **Submit Data** program block is to submit to the host the variables necessary for the transaction processing.

Our application example assumes that the following information is required for the transaction processing:

- oebr.submit_mode
- card.pan
- card.effective_date
- card.expiry_date
- card.emv.app_pan_seq

- `card.emv.aip`
- `card.emv.auc`
- `card.emv.atc`
- `card.emv.aac`
- `card.emv.tc`
- `card.emv.arqc`
- `card.emv.iad`
- `card.emv.tvr`
- `card.emv.unumber`
- `transid`
- `payment.amount`

As discussed in the [“Risk Management block” on page 50](#), risk management verdict is contained in the `card.parser.verdict` variable. It can be one of the following:

- `"offline"`
- `"online"`
- `"reject"`

If the Risk Management verdict is `"online"`, host authorisation is required. Therefore, the submission must be done online and the next step of the application will be the **Host Authorisation** block (see [page 54](#)).

If the Risk Management verdict is `"offline"`, authorisation is not necessary. The host can process the transaction later and the submission can be done offline.

If the Risk Management verdict was `"reject"`, the transaction was rejected, and there is no need to submit data to host.

Data submission to the host is done using the `<submit>` TML command. It generates a form that will be sent to the host module component that is responsible for accepting terminal requests. This component is specified by the `tgt` attribute of the `<submit>` tag. The page that the application will go to in case of a connection error is specified by the `econn` attribute.

Note: for more information on the `<submit>` command, see *TML Application Development Guidelines*.

Incendo Online Microbrowser variable that is responsible for setting the submit mode is `oebr.submit_mode`. It is set to `"online"` if the submission must be done online, and `"offline"` if otherwise. The easiest way to do that is to set it to the value of the Risk Management verdict:

```
<setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict"/>
```

In case of the online submission, application proceeds to the **Host Authorisation** block (see [page 54](#)).

If the submission is to be done offline, **Print Receipt** block is next (see [page 58](#)).

If there is a connection error during online submission, **Risk Management** must be performed again (see [page 50](#)).

submit screen

The path to the host and the screen that is accessed in case of the connection error are set by the attributes of the `<submit>` tag:

```
<submit tgt="/iccemv/authtxn" econn="#icc_conn_fld">
```

In our application example, the path `" /iccemv/authtxn"` sends the data to the `ICCEMVTxnProcessingServlet` which emulates the interaction with the acquirer.

`ICCEMVTxnData` works with the data structure that was sent by the `<submit>` command. `icc_conn_fld` screen is part of the **Risk Management** block (see [page 50](#)).

Figure 6 - 16: #submit screen code

```

<screen id="submit">
  <setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict"/>
  <next uri="#receipt">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="#void_trans"/>
  </next>
  <submit tgt="/iccemv/authtxn" econn="#icc_conn_fld">
    <getvar name="oebr.submit_mode"/>
    <getvar name="card.pan"/>
    <getvar name="card.effective_date"/>
    <getvar name="card.expiry_date"/>
    <getvar name="card.emv.app_pan_seq"/>
    <getvar name="card.emv.aip"/>
    <getvar name="card.emv.auc"/>
    <getvar name="card.emv.atc"/>
    <getvar name="card.emv.aac"/>
    <getvar name="card.emv.tc"/>
    <getvar name="card.emv.arqc"/>
    <getvar name="card.emv.iad"/>
    <getvar name="card.emv.tvr"/>
    <getvar name="card.emv.unnumber"/>
    <getvar name="transid"/>
    <getvar name="payment.amount"/>
  </submit>
</screen>

```

Connection Error block

This application block deals with the connection errors that occurred during the execution of the **Submit Data** application block (see page 51).

If the connection error occurred during the offline submission, it means that the offline transaction pool is full. Our application example assumes that if the offline transaction pull is full, the transaction should be rejected. The application proceeds to the **Transaction Rejected** block (see page 57).

If the connection error occurred during the online submission, application control is transferred back to the **Risk Management** block, which decides whether the online submission should be repeated, or an offline submission can be performed (see page 50). If an online submission is required, but it can not be done because of the connection errors, **Risk Management** block should reject the transaction.

Connection Error block consists of:

- `icc_conn_fld` screen

icc_conn_fld screen

The purpose of this screen is to process the connection errors that occurred during the online data submission (done by the **Submit Data** block).

Our ICCEMV application example implements a very simple risk management algorithm. It simply redirects the application to the start of the **Risk Management** block.

Figure 6 - 17: #icc_conn_fld screen code

```

<screen id="icc_conn_fld">
  <next uri="#risk_mgmt">
    <variant lo="tmlvar:oebr.submit_mode" op="equal"
ro="offline" uri="#void_trans" />
  </next>
</screen>

```

Host Authorisation block

The purpose of the Host Authorisation program block is to conduct online authorisation. The terminal application host-side module should do the following:

1. parse the TML data sent by the **Submit Data** block (see page 51)
2. generate an appropriate authorisation request and send it to the card acquirer
3. process the response from the acquirer
4. send an appropriate dynamic TML page to the terminal, depending on the acquirer's verdict.

In our ICCEMV application example, two Java servlets are used to emulate this process:

- `ICCEMVTxnData` that works with the data structure that was sent by the **Submit Data** block, and
- `ICCEMVTxnProcessingServlet` that processes the information

`ICCEMVTxnProcessingServlet` emulates the interaction with the acquirer.

The transaction processing servlet does the following:

- parses the TML post request.
- checks the card expiry date against the current date.
- checks the transaction amount against the defined top limit.
- selects the dynamic page that will be sent to the terminal.

The function of the dynamic TML page is discussed in the [“Dynamic Page block”](#) below.

Dynamic Page block

The dynamic TML pages described in this section are sent by the host module of ICCEMV to the terminals in response to their transaction authorisation requests.

The purpose of these pages is to set up the TML variables that will be used by the **ICC Authorisation** block, and to initiate ICC authorisation. Additionally, a dynamic TML page may inform the user of the results of the Host Authorisation.

The card parser uses the following variables to decide whether to authorise the transaction:

- `payment.auth_code`
- `payment.auth_resp_code`
- `payment.txn_result`
- `payment.trans_type`
- `payment.emv.issuer_auth`
- `payment.emv.issuer_script1`
- `payment.emv.issuer_script2`

Therefore, the dynamic TML page should fill some of these variables before **ICC Authorisation** is conducted.

The pages that are sent to the terminal are:

- if the host approves the transaction, `auth_ok.tml` is used (see page 55)
- if the transaction amount exceeds the upper limit, `amnt_big.tml` is sent (see page 55)
- if the transaction has been declined, `failed.tml` is utilised (see page 55)

auth_ok.tml page

The page is sent to the terminal when the transaction has been approved.

To indicate that the Host Authorisation process was successful, `payment.txn_result` variable is set to 1:

```
<setvar name="payment.txn_result" lo="1"/>
```

The page then initiates the **ICC Authorisation** (see page 56).

Figure 6 - 18: auth_ok.tml code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="auth_ok" next="/iccmv/iccmv.tml#subm_tc_aac">
<!-- authorization approved -->
    <setvar name="payment.txn_result" lo="1"/>
<!-- auth_code, issue_auth and issuer_script may be filled -->
    <tform>
      <card parser="icc_emv" parser_params="auth"/>
    </tform>
  </screen>
</tml>
```

failed.tml page

The page is sent to the terminal when the transaction has been rejected by the host.

To indicate that the Host Authorisation process has failed, `payment.txn_result` variable is set to 0:

```
<setvar name="payment.txn_result" lo="0"/>
```

The page then initiates the **ICC Authorisation** (see page 56).

Figure 6 - 19: failed.tml code

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="failed" next="/iccmv/iccmv.tml#subm_tc_aac">
<!-- authorization declined -->
    <setvar name="payment.txn_result" lo="0"/>
<!-- auth_code, issue_auth and issuer_script may be filled -->
    <tform>
      <card parser="icc_emv" parser_params="auth"/>
    </tform>
  </screen>
</tml>
```

amnt_big.tml page

The page is sent as a host-side reply when the transaction amount exceeds the top limit.

The page contains two screen – one outputs a message to the user, and the other sets the variables and calls the card parser.

To indicate that the Host Authorisation process has failed, `payment.txn_result` variable is set to 0:

```
<setvar name="payment.txn_result" lo="0"/>
```

The page then initiates the **ICC Authorisation** (see page 56).

Figure 6 - 20: amnt_big.tml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">

  <screen id="reason" class="c_center" timeout="1"
next="#failed">
    <display>
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Amount Exceeds Top Limit
        </td></tr>
```

```
        </table>
    </display>
</screen>

    <screen id="failed" next="/iccmv/iccmv.tml#subm_tc_aac">
<!-- authorization declined -->
        <setvar name="payment.txn_result" lo="0"/>
<!-- auth_code, issue_auth and issuer_script may be filled -->
        <tform>
            <card parser="icc_emv" parser_params="auth"/>
        </tform>
    </screen>

</tml>
```

ICC Authorisation block

As mentioned previously, as the result of the **Host Authorisation** process (see page 54) the host sends a dynamic TML page to the terminal.

ICC Authorisation is performed from within the dynamic page using the following command:

```
<card parser="icc_emv" parser_params="auth"/>
```

The card parser uses the following variables to decide whether to authorise the transaction:

- `payment.auth_code`
- `payment.auth_resp_code`
- `payment.txn_result`
- `payment.trans_type`
- `payment.emv.issuer_auth`
- `payment.emv.issuer_script1`
- `payment.emv.issuer_script2`

Therefore, the dynamic TML page should fill some of these variables before **ICC Authorisation** is conducted.

The example ICCEMV application fills only one variable - `payment.txn_result`. It is set to 1 if the host authorised the transaction and to 0 if the transaction was denied. See “[Dynamic Page block](#)” on page 54 for the code of various dynamic TML pages.

Once the `icc_emv` parser processes the “auth” command, it sets the following variables:

- `card.emv.aac`
This variable is filled if the transaction was declined
- `card.emv.tc`
This variable is filled if the transaction was accepted
- `payment.emv.script_results`

These variables should be submitted to the host. This is done by the **Authorisation Results** block.

Authorisation Results block

As described above, the dynamic TML page, generated by the host, has called authorisation command of the `icc_emv` parser. The results of this authorisation must be submitted to the database. Then, depending on the outcome of authorisation, the application proceeds to either **Print Receipt** or **Transaction Rejected** block.

This functionality is implemented using two screens:

- `subm_tc_aac`
- `end_trans`

subm_tc_aac screen

This page submits to host variables `card.emv.aac` and `card.emv.tc`. One of these variables was filled as the result of the authorisation (`auth`) command executed by the `icc_emv` parser previously. Additionally, the transaction ID (`transid` variable) is also submitted, to be used for reference.

Figure 6 - 21: #subm_tc_aac screen code

```
<screen id="subm_tc_aac">
  <setvar name="oebr.submit_mode" lo="offline"/>
  <next uri="#end_trans"/>
  <submit tgt="/iccemv/subm_tc_aac" econn="#end_trans">
    <getvar name="oebr.submit_mode"/>
    <getvar name="card.emv.aac"/>
    <getvar name="card.emv.tc"/>
    <getvar name="transid"/>
  </submit>
</screen>
```

Submission process is set up to be always performed in the offline mode. This is done by assigning "offline" value to the `oebr.submit_mode` variable.

On the host side, two Java servlets are used to process the submission:

- `ICCEMVTxn2Data` that defines the data structure, and
- `ICCEMVTxn2ProcessingServlet` that processes the information

After the submission is performed, application proceeds to the `end_trans` screen.

end_trans screen

This screen is used to transfer control to either **Print Receipt** or **Transaction Rejected** block, depending on the value of the `payment.txt_result` variable. This variable was set by the host during the authorisation process (see [“Host Authorisation block” on page 54](#)). The value is 0 if the transaction has failed.

Figure 6 - 22: #end_trans screen data

```
<screen id="end_trans">
  <next uri="#receipt">
    <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="#void_trans"/>
    <variant lo="tmlvar:payment.txn_result" op="equal" ro="0"
uri="#void_trans"/>
  </next>
</screen>
```

Transaction Rejected block

If the transaction is rejected, the application should do the following:

- notify the user of the rejection and give the rejection reason,
- if the transaction has already been processed, void the transaction

If the transaction has been rejected in one of the previous application blocks, `card.parser.verdict` variable was set to "reject" and `card.parser.reject_reason` was filled with the explanation of the rejection.

Therefore, you can simply display the `card.parser.reject_reason` variable to inform the user. ([“reject_trans screen” on page 58](#)).

The application should also void the transaction, if it has been processed. Our basic application does not implement this functionality. It assumes that the transaction was rejected or cancelled before transaction data had a chance to be processed, and simply informs the user that the transaction has been declined. ([“void_trans screen” on page 58](#)).

The next step in the application flow is the **Remove Card** block (see [page 59](#)).

reject_trans screen

The screen displays the message that the transaction is rejected mentioning the reason. Then the application switches to the `void_trans` screen.

Figure 6 - 23: #reject_trans screen code

```
<screen id="reject_trans" class="c_center" next="#void_trans">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        <getvar name="card.parser.reject_reason"/>
      </td></tr>
    </table>
  </display>
</screen>
```

void_trans screen

The screen displays a message that the transaction is declined.

Figure 6 - 24: #void_trans screen code

```
<screen id="void_trans" class="c_center" timeout="1"
next="#remove_card">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Transaction Declined
      </td></tr>
    </table>
  </display>
</screen>
```

The next step in the application flow is the **Remove Card** block (see page 59).

Print Receipt block

If the transaction was successful up to this point, a receipt is printed.

Note: if the value of the `card.emv.signature` variable was set to 1 during the previous application steps, a signature check is required. However, PIN entry is the most common verification method for an ICC transaction. Therefore, our ICC application example does not support signature check functionality.

The `receipt` screen is used for printing a receipt.

The application then proceeds to the **Transaction Completed** block (see page 59).

receipt screen

This screen uses the `<print>` TML tag to print some basic information. The following information is printed:

- Current time
the value is taken from the `terminal.datetime` variable, which is updated by the terminal.
- Card number
`card.pan` variable was filled earlier by the `icc_emv` parser. This was done during the execution of the **Initiate Transaction** block (see page 44)
- Transaction ID
`transid` variable set earlier
- Payment amount
the value of the `payment.amount` variable was entered by the user during the execution of the **Transaction Amount** block (see page 45)

Figure 6 - 25: #receipt screen code

```
<screen id="receipt" next="#compl_trans">
  <print>
    <table>
      <tr>
        <td>Date:</td>
        <td><getvar name="terminal.datetime"/></td>
      </tr>
      <tr>
        <td>Card:</td>
        <td><getvar name="card.pan" format="n4n#*n4"/></td>
      </tr>
      <tr>
        <td>Transaction ID:</td>
        <td><getvar name="transid"/></td>
      </tr>
      <tr>
        <td>Amount:</td>
        <td><getvar name="payment.amount "
format="^*0.00"/></td>
      </tr>
    </table>
    <br/>
    -----
    <br/><br/><br/>
  </print>
</screen>
```

Transaction Completed block

This application block is reached if the transaction was completed successfully.

The next step is the **Remove Card** block.

compl_trans screen

The screen displays a message that the transaction has been completed. The application then switches to the **Remove Card** block (see page 59).

Figure 6 - 26: #compl_trans screen code

```
<screen id="compl_trans" class="c_center" timeout="1"
next="#remove_card">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Transaction Completed
      </td></tr>
    </table>
  </display>
</screen>
```

Remove Card block

This application block finalizes the transaction. The user is asked to remove the card, and the application waits for the card to be removed.

This is done using the `wait_remove_card` command of the `icc_emv` parser:

```
<card parser="icc_emv" parser_params="wait_remove_card"/>
```

Once the card removed, the transaction is finished.

In our application example, the application proceeds back to the **Initiate Transaction** block, where it waits for the user to initiate another transaction (see [page 44](#)).

remove_card screen

`<tform>` command is used - the screen waits for the user input. In this case the input will be the action of removing the card. The `<prompt>` element is used to display a short message on the terminal screen during this wait.

Figure 6 - 27: #remove_card screen code

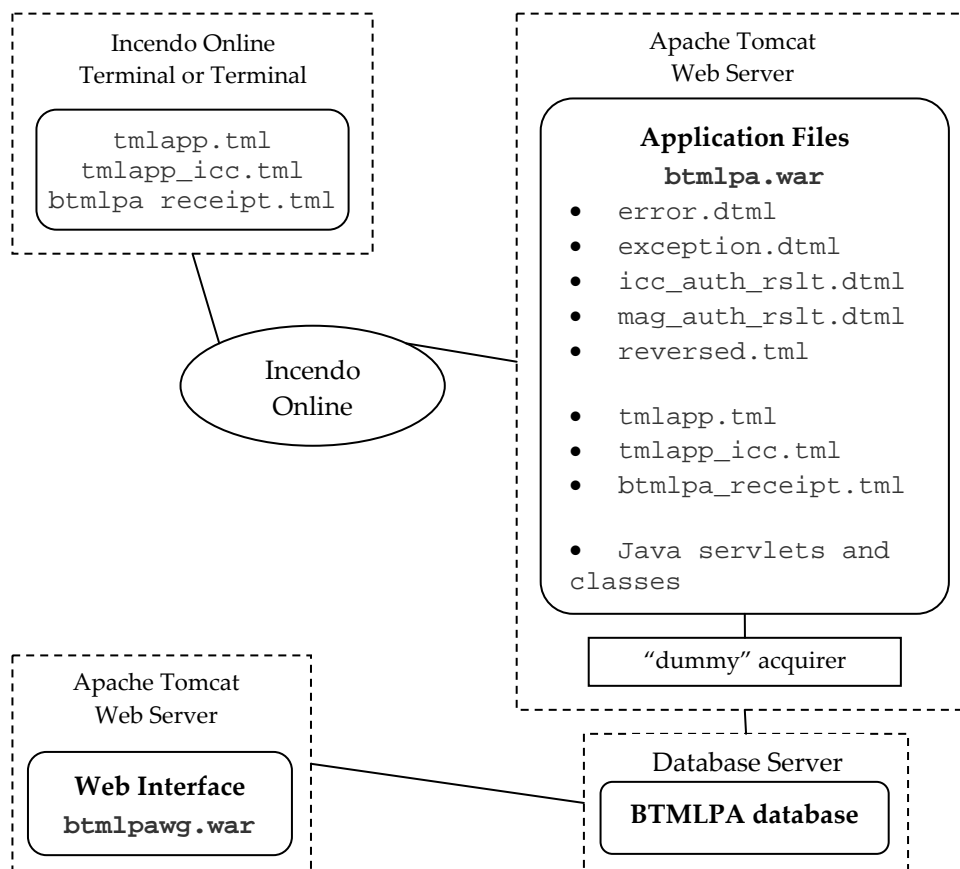
```
<screen id="remove_card" class="c_center" next="#init_prompt">
  <tform>
    <card parser="icc_emv" parser_params="wait_remove_card"/>
    <prompt>
      <table height="100%" width="100%">
        <tr><td valign="middle" align="center">
          Please, Remove Card
        </td></tr>
      </table>
    </prompt>
  </tform>
</screen>
```

BTMLPA architecture



Basic TML Payment Application (BTMLPA) builds on the MAGCARD and ICCEMV application examples.

Figure 7 - 1: BTMLPA outline



It consists of four broad parts (Figure 7 - 1 above):

- terminal-side module (see page 61)
- host-side module (see page 62)
- BTMLPA database (see page 63)
- BTMLPA web interface (see page 63)

The terminal-side and the host-side modules interact by the means of the Incendo Online Gateway. The host-side module uses the BTMLPA database to store transaction information. The web interface is used to view and modify the BTMLPA database.

These parts interact to conduct magnetic card and ICCEMV transactions (see page 64).

BTMLPA terminal-side module

The terminal-side module is implemented as three linked static TML pages (`tmlapp.tml`, `tmlapp_icc.tml` and `btmlpa_receipt.tml`) accompanied with the style sheet files. (`receipt.css`, and `5100.css` or `8550.css`).

These resources are downloaded by a terminal from the host server when a terminal user selects the Go to Homepage link in the **Incendo Online MicroBrowser** (start-up) screen.

The `tmlapp.tml` contains all of the common TML screens, and screens that are necessary for processing a magnetic card transaction.

The `tmlapp_icc.tml` has only the screens specific to the EMV transaction.

The `btmlpa_receipt.tml` deals with the receipt printing.

`receipt.css`

The `5100.css` describes the styles that are used by the monochrome screen terminals.

The `8550.css` contains the styles that are used by the color screen terminals.

The `receipt.css` is used for the receipt styles.

BTMLPA host-side module

The purpose of the host-side module of BTMLPA is to process transaction authorisation requests coming from the terminals via the Incendo Online Gateway.

The host module is responsible for:

- accepting, parsing and forwarding terminal transaction authorisation requests to the acquirer host
- accepting and parsing the acquirer host's replies
- generating dynamic TML pages and sending them to the terminals as the replies to the authorisation requests

Note: The BTMLPA host module supports only “dummy” transaction authorisation. This means that the module does not actually communicate with an external acquirer host when processing transaction authorisation requests. For every authorisation request, it checks the card expiry date, validates the transaction amount by comparing it with the predefined floor limit, and generates an authorisation verdict.

The host module includes:

- software components (implemented as Java servlets) whose main function is to select, construct and send to the terminals the TML pages appropriate for the terminal model being used and the received authorisation request(s) (see [“BTMLPA Java servlets”](#) below)
- a set of dynamic TML pages that are sent to the terminals in response to their transaction authorisation requests (see [“Dynamic TML pages”](#) on page 63)

BTMLPA Java servlets

The host functionality is implemented using the following Java servlets:

- `AuthResultData.java`
defines the data structure that is used by the `TxnAuthServlet.java` for the authorisation results
- `ReversalServlet.java`
processes the Reversal transaction request
- `ReversalData.java`
defines the data structure used by the `ReversalServlet.java`
- `BaseServlet.java`
- `ICC2OfflineSubmitData.java`
used by the `ICC2OfflineSubmitServlet.java`
- `ICC2OfflineSubmitServlet.java`
responsible for processing the ICC authorisation results submission
- `TxnAuthData.java`
defines the data structure that is received from the terminal. It is used by the `TxnAuthServlet.java`
- `TxnAuthServlet.java`
processes the Sale, Sale With Cashback and Refund transaction

requests. It emulates interaction with the acquirer and supplies the authorisation results.

- `RequestFilter.java`
selects the TML resource version appropriate for the model of the terminal (5100 or 8550) that originated the corresponding request.

Dynamic TML pages

As the response for a transaction request, the host will either send a pre-made TML page (such as `reversed.tml`) or generate a TML page using Freemarker from a template (such as `icc_auth.rslt.dtml`).

No matter how this page is created, the end result is a valid TML page that is loaded into the terminal.

The terminal will then execute this TML page.

Note: for more information on the dynamic pages see [“Dynamic Page block” on page 82](#).

BTMLPA database

BTMLPA database is used to store:

- terminal information, such as Ingenico terminal IDs (ITIDs) and APACS terminal IDs (APACS TIDs)^{*h*}
- transactions data, such as transaction date and time, information about the card used, transaction type and amount(s), authorisation result and so on – for each transaction processed by the host module
- the information about the users authorised to work with BTMLPA web interface (GUI).

BTMLPA web interface (GUI)

BTMLPA Web GUI is an application intended for remote monitoring and control of BTMLPA’s operation. It allows managing the application users, connected terminals and viewing session information.

With Incendo Online, developers can easily provide their TML applications with the possibilities of remote administration, allowing help desk personnel to view the application statistics, modify various application parameters and settings.

BTMLPA Web interface is intended as an example of how you can provide web access to a TML application using an open standards-based Java Servlet/JSP technology.

The interface implementation relies on the Apache Struts, an open source framework for building Java web applications, see the Apache Struts framework home page at <http://struts.apache.org/> for more information.

BTMLPA Web GUI is supplied as part of Incendo Online host software. The corresponding web archive file `btmlpawg.war` can be found in the directory `%OE_HOME%\examples\btmlpa`. The archive can be deployed on any web server capable of running Java servlets.

For deployment instructions as well as for basic information on how to use the application, refer to *Incendo Online Desktop Installation Guidelines* (the sections “Deploying web components” and “Testing the installation” respectively).

^{*h*} Within the Incendo Online, the terminals are identified by their ITIDs. However, ITIDs can only be used within Ingenico and should not be exposed to any external entity (such as an acquirer host). The acquirer host recognises the terminals only by their APACS TIDs and does not know anything about their ITIDs.

BTMLPA details

8

BTMLPA builds upon the MAGCARD and ICCEMV application examples, so it is a good idea to go through the “[MAGCARD application example](#)” on page 28 and “[ICCEMV application example](#)” on page 42 before the BTMLPA.

There are two separate sets of TML resources for different terminal models: one for *Ingenico 5100* terminals and the other – for *Ingenico 8550*. Like “[MAGCARD application example](#)” chapter on page 28 and “[ICCEMV application example](#)” on page 42, this chapter deals with the TML implementation for the *Ingenico 5100* terminals.

Important: see the content of the `tmlapp.tml` and `tmlapp_icc.tml` files for the TML code of a particular screen. These files are in the directory `%OE_HOME%\examples\btmlpa\src\source\web\5100\`

The added functionality is:

- several transaction types
- manual entry of the magnetic card data
- improved receipt printing
- signature checking

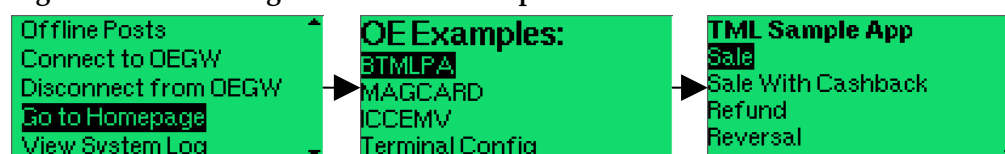
BTMLPA preliminaries

This section will cover general transaction flow, **Transaction Preparation** block of TML screens, and application-specific variables that are used by the BTMLPA.

Starting a BTMLPA transaction

To access the BTMLPA examples, from the Incendo Online MicroBrowser start-up screen select `Go to Homepage` link. Then, use the BTMLPA link. The terminal will access the `tmlapp.tml` page ([Figure 8 - 1 below](#)).

Figure 8 - 1: Accessing the BTMLPA examples



Note: the terminal must be already activated by the Incendo Online Gateway.

The links `Sale`, `Sale With Cashback` and `Refund` will initiate a magnetic card or an ICCEMV transaction of the appropriate type.

`Reversal` will reverse a previously executed transaction.

`Duplicate` will print a duplicate receipt for a previously executed transaction.

`Examples` will take you back to the OE Examples screen ([Figure 8 - 1 above](#)).

`About` link will display the BTMLPA version number.

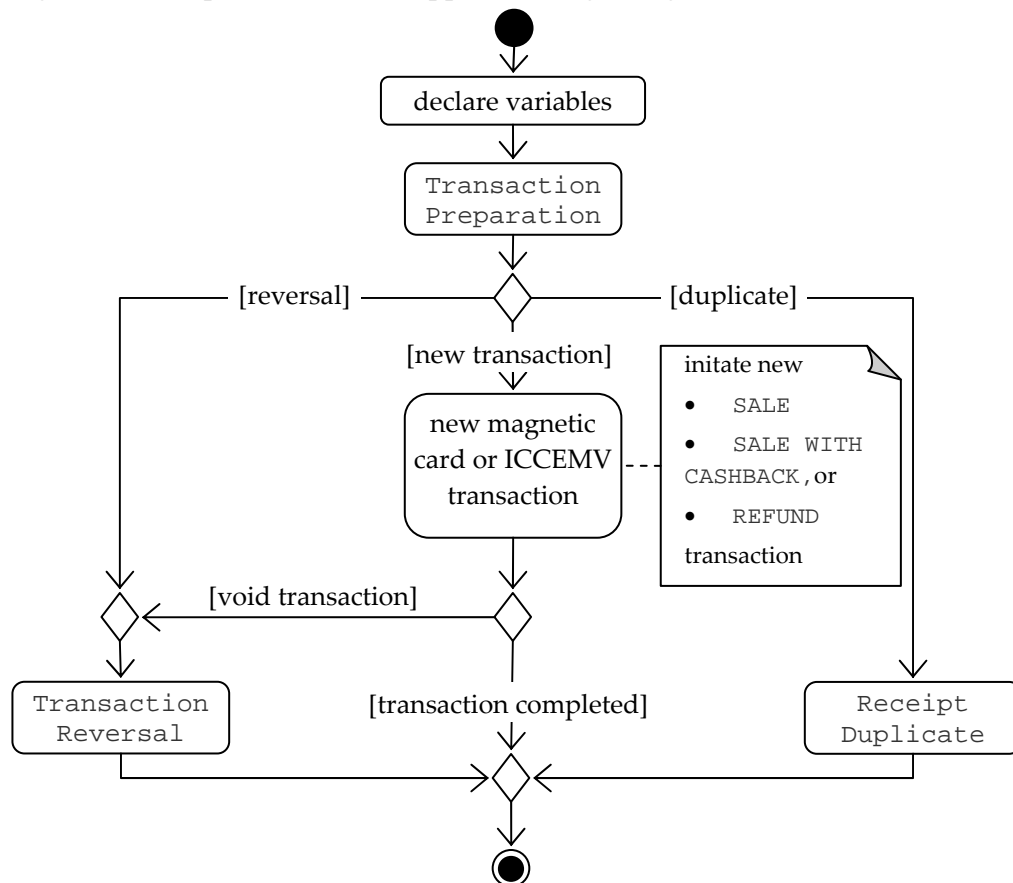
General transaction flow

The general transaction flow is shown in the [Figure 8 - 2 on page 65](#).

First, application-specific variables are declared (see page [Error! Bookmark not defined.](#)).

Transaction Preparation block (see page [66](#)) of TML screens is executed before the actual transaction is started. It is common to all transaction types.

Figure 8 - 2: simplified BTMLPA application logic diagram



The user is prompted to choose a particular transaction. Depending on the user selection, a new magnetic card or ICCEMV transaction is executed (), a previous transaction is reversed (see page 67), or a receipt duplicate is printed (see page 68).

Supported transaction types are discussed in more detail [below](#).

A list of application-specific variables is given [Error! Bookmark not defined.](#)

Supported transaction types

BTMLPA supports the following transaction types:

- **Sale** (`payment.trans_type="debit"`)
The sale transaction assumes that the amount entered by the merchant is withdrawn from the cardholder's account as a charge for goods or services.
- **Sale With Cashback** (`payment.trans_type="cashback"`)
The merchant specifies the amount of money the customer (cardholder) is charged for the goods or services, and part of that amount that will be given to the customer as cash. In effect, this is a combination of a sale and a cash withdrawal.
- **Refund** (`payment.trans_type="credit"`)
The merchant returns a certain amount of money back to the customer according to the conditions of a prior sale.
- **Reversal** (`payment.trans_type="reversal"`)
The previous successful **Sale**, **Sale With Cashback** or **Refund** transaction is cancelled or reversed at the merchant's request.

Additionally, it is possible to print a duplicate receipt of the previous transaction.

Transaction Preparation block

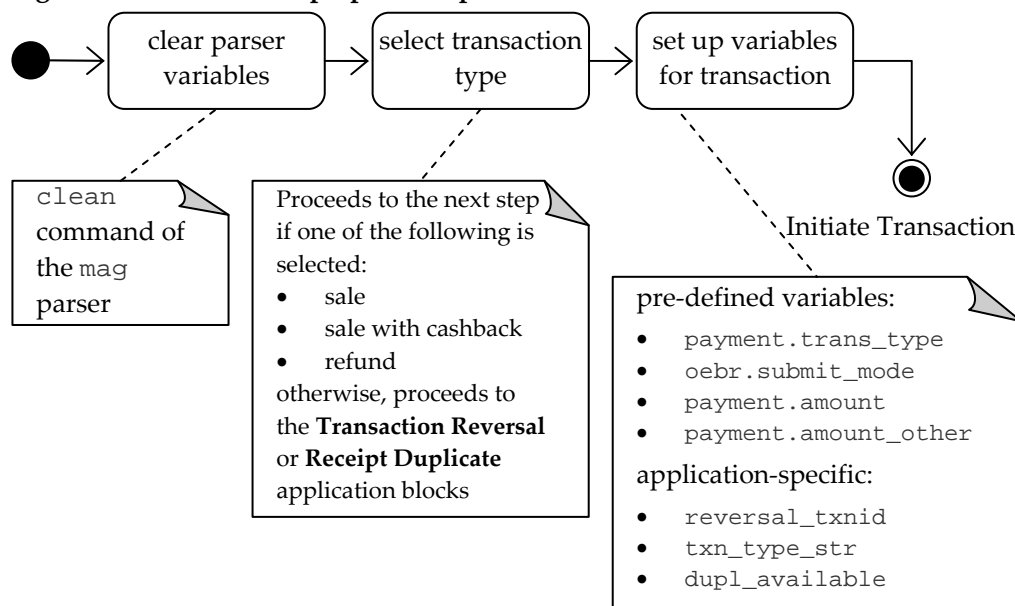
This logic block prepares the program for the transaction. The application clears the parser-related variables, and asks the user to select the required transaction type (see page 65 for a description of the transaction types).

Then, TML application assigns values to several application-specific and pre-defined variables, based on the transaction type (Figure 8 - 3 below).

The pre-defined variables that are used by the mag parser are cleared using the

```
<card parser="mag" parser_params="clean" />
```

Figure 8 - 3: Transaction preparation process



Transaction Preparation block consists of the following:

- `init_prompt`
clear the parser-related variables.
- `main_menu`
The user selects the transaction type by selecting a link on the terminal display. For description of the transaction types supported by the BTMLPA, see [“Supported transaction types” on page 65](#).
- `sale`
sets up the variables for Sale transaction
- `sale_cb`
sets up the variables for Sale With Cashback transaction
- `refund`
sets up the variables for Refund transaction

REVERSAL transaction and Duplicate receipt

Both the `Reversal` transaction and duplicate receipt can only be done after a successful transaction of the `Sale`, `Sale With Cashback` or `Refund` type (see page 69).

Transaction Reversal block

Transaction reversal is initiated if the user selects **Reversal** link from the `main_menu` screen of the **Transaction Preparation** block (see page 66).

The main function of the **Transaction Reversal** block is to submit to host the value of the transaction ID (`transid` variable) of the transaction that should be reversed. The host should take care of the rest.

Figure 8 - 4: submitting the reversal data to the host – #submit_rev screen

```
<screen id="submit_rev" next="#receipt" cancel="#trans_abrt">
  <setvar name="sbmt_url" lo="#submit_rev" />
  <submit tgt="/btmlpa/reversal" econn="#conn_fld">
    <getvar name="transid" />
    <getvar name="oebr.submit_mode" />
  </submit>
</screen>
```

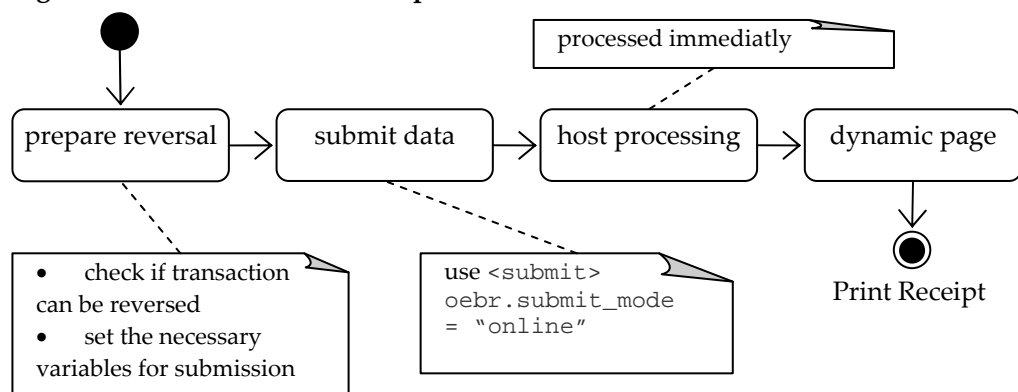
Submission is done using the `<submit>` element of the `submit_rev` screen (Figure 8 - 4 above). The `"/btmlpa/reversal"` value of the `tgt` attribute is processed by the host and the data is referred to the `ReversalServlet.java`.

The data structure that the host requires for the reversal is defined within the `ReversalData.java`. On the terminal side, it is set using `<getvar>` elements (Figure 8 - 4 above).

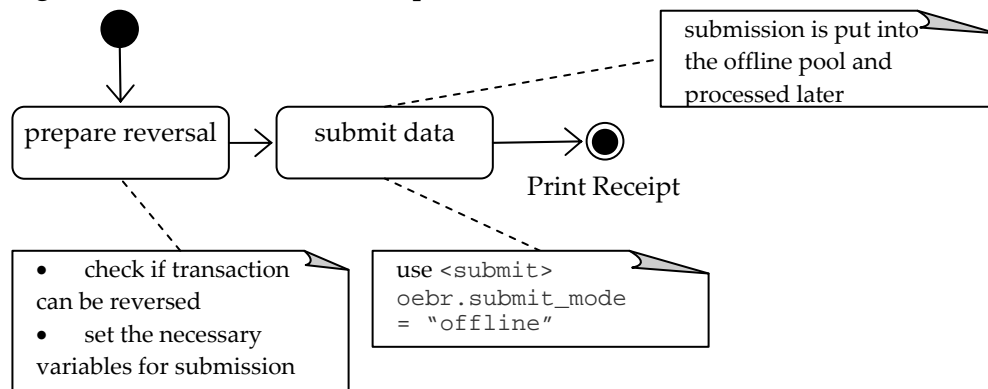
The data can be submitted to host using either online or offline mode (see Figure 8 - 5 below and Figure 8 - 6 on page 68), depending on the value of the pre-defined variable `oebr.submit_mode`.

In case of the online submission, the control is passed to host. Once the host processes the transaction, it sends a TML page to the terminal. In our case this page is the `reversed.tml`. This dynamic page determines what the application does next – in our case the application proceeds to the **Print Receipt** program block (see page 85).

Figure 8 - 5: Transaction reversal process – online submission



If submission is done offline, the next screen the application will access is defined by the next attribute of the screen that originated the data submission (see Figure 8 - 4 above). In our case, the application will proceed to the receipt screen - **Print Receipt** program block (see page 85).

Figure 8 - 6: Transaction reversal process - offline submission

Transaction Reversal block consists of the following screens:

- `check_rev`
Checks if the reversal can be performed (i.e. the previous transaction was Sale, Sale With Cashback, or Refund type)
- `confirm_rev`
The merchant is asked to confirm the reversal
- `sub_rev_prep`
Sets various variables in preparation for submission
- `submit_rev`
This screen submits several variables to the host
- `conn_fld`
This screen is reached if there has been a connection error during submission. if submission was performed using the “online” mode the program will try to submit in the “offline” mode
- `offline_rev`
It sets up the “offline” mode for the reversal submission.
- `subm_err`
This screen is reached if there was a connection error during the online submission and the offline pool is full.
- `rev_bad`
Displays a message to the user that the reversal can not be performed

Receipt Duplicate block

This application block checks if the receipt duplicate can be printed and sets up the variables that will be used by the Print Receipt block.

Receipt Duplicate block consists of the following screens:

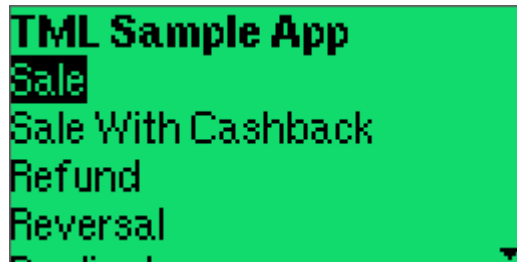
- `duplicate`
checks if the duplicate is available (`dupl_available` variable).
- `duplicate_na`
accessed if the receipt duplicate is not available.
- `duplicate_av`
sets up the variables for duplicate receipt printing and transfers the control to the **Print Receipt** block

The **Print Receipt** block will take care of the actual printing (see page 85).

SALE, SALE WITH CASHBACK or REFUND transaction

A new magnetic card or ICCMV transaction of the appropriate type is initiated if the user selects *Sale*, *Sale With Cashback* or *Refund* links during the execution of the **Transaction Preparation** block ([Figure 8 - 7 below](#)).

Figure 8 - 7: Terminal display during the execution of the main_menu screen of the Transaction Preparation block



The magnetic and ICCMV card transaction of the BTMLPA is an extension of the MAGCARD Application Example (see [page 28](#)) and ICCMV Application Example (see [page 42](#)) with some added functionality.

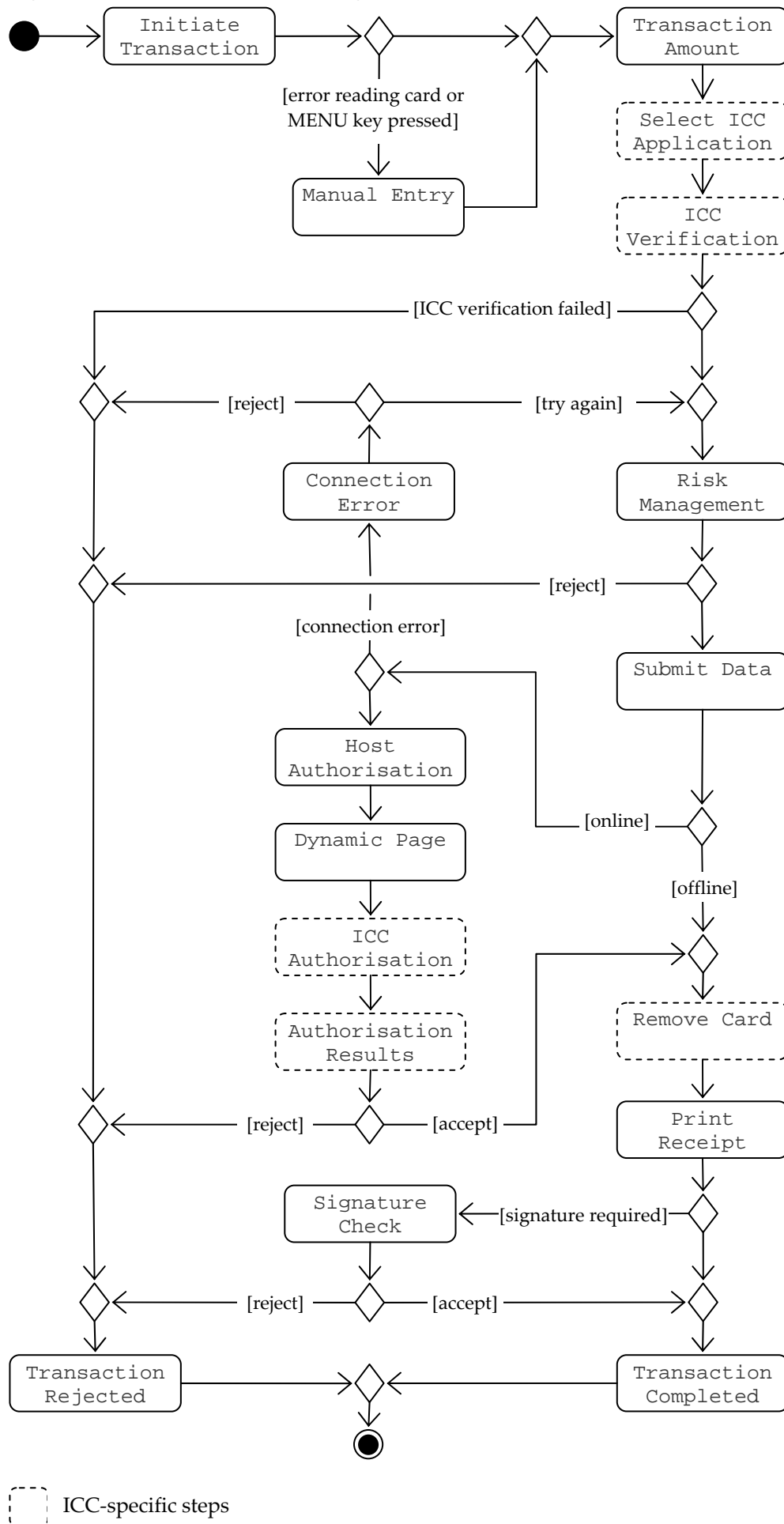
The added functionality is:

- several transaction types
- manual entry of the magnetic card data
- improved receipt printing
- signature checking

The transaction logic is shown on the [Figure 8 - 8 on page 70](#).

Each application block is a collection of TML screens. The sections that follow describe the function of each block.

Figure 8 - 8: BTMLPA transaction logic



Initiate Transaction block

Initiate Transaction block is the same for the magnetic card and ICCEMV transaction. Once the transaction has been prepared (see page 66), the transaction is initiated when the user swipes or inserts a card.

The purpose of this application block is to read the card data.

This is done by the following commands:

```
<card parser="mag" parser_params="read_data" />
<card parser="icc_emv" parser_params="init_app" />
```

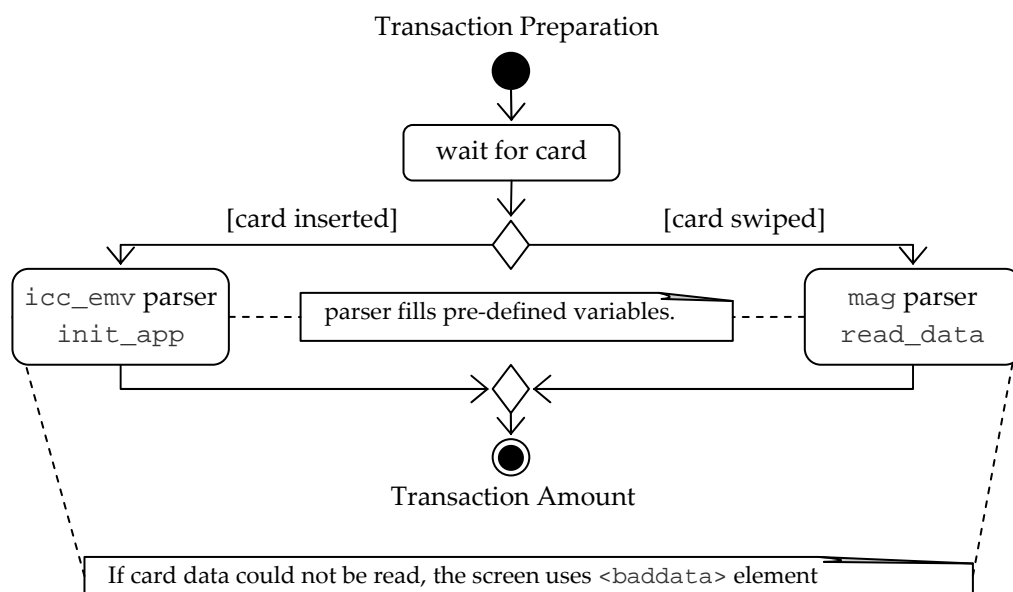
Both parsers are called from the same screen - read_card (Figure 8 - 9 below).

BTMLPA implements a basic fallback functionality – the next attribute of the <baddata> element is used to access the **Manual Entry** application block in case of the read error. **Manual Entry** can also be selected by the user by pressing the **Menu** button.

Figure 8 - 9: #read_card screen TML code

```
<screen id="read_card" class="c_center" menu="#man_entry"
next="#read_amount">
  <tform>
    <baddata class="c_center" max="3" next="#to_man_entry">
      <table border="2" height="100%" width="100%">
        <tr><td align="center" valign="middle">
          <getvar name="err.baddata_reason" />
        </td></tr>
      </table>
    </baddata>
    <card parser="mag" parser_params="read_data" />
    <card parser="icc_emv" parser_params="init_app" />
    <prompt>
      <table height="100%" width="100%">
        <tr><td align="center" valign="middle">
          Please, swipe/insert card<br />
          Press MENU for keying
        </td></tr>
      </table>
    </prompt>
  </tform>
</screen>
```

Figure 8 - 10: Initiate Transaction process



If the card is swiped, it is read by the mag parser. If ICC card is inserted, icc_emv parser is used (Figure 8 - 10 above).

The parser fills pre-defined variables with the card data. One of these variables is `card.input_type`. It is set to:

- 1 if a magnetic card transaction was initiated, or
- 2 if ICCEMV transaction was started.

Note: *TML Application Development Guidelines* provides detailed description of the card parser operation.

Only one TML screen is used by this application block:

- `read_card`

Manual Entry block

This application block is used if the card data can not be read from the card or the user wishes to enter the data manually.

The following card data can be entered manually (using the terminal keypad):

- the card number
`card.pan` pre-refined variable
- the card expiry date
`card.expiry_date` pre-refined variable

All the above information is usually embossed on the card so the merchant can easily read it.

To indicate that the card details have been entered manually, the pre-defined variable `card.input_type` should be set to 3.

The data is entered using the `<input>` elements of the `<form>` element.

The following screens are used:

- `to_man_entry`
informs the user that the card is not readable
- `man_entry`
sets up some variables
- `card_nmb`
accepts the user card details input

Transaction Amount block

The purpose of this application block is to set the values for the following pre-defined variables:

- `payment.amount`
transaction amount, used for the Sale, Sale With Cashback and Refund transactions
- `payment.amount_other`
cash-back amount, used only for the Sale With Cashback transactions

BTMLPA uses the user input to set these variables. The data is entered using the `<input>` elements of the `<form>` element.

In addition, the application-specific variable `transid` is set to the value of the pre-defined variable `oebr.unique_id`.

The following screens are used:

- `read_amount`
- `read_sale`
- `read_scb`
- `read_refund`

For magnetic card transaction (`card.input_type="1"`) and for manual entry transactions (`card.input_type="3"`) the next step is **Risk Management** (page 76).

For ICCEMV transaction, the application must perform **ICC Verification** (page [Error! Bookmark not defined.](#)).

Select ICC Application

The purpose of this application block is to choose the ICC application that will be used for cardholder verification.

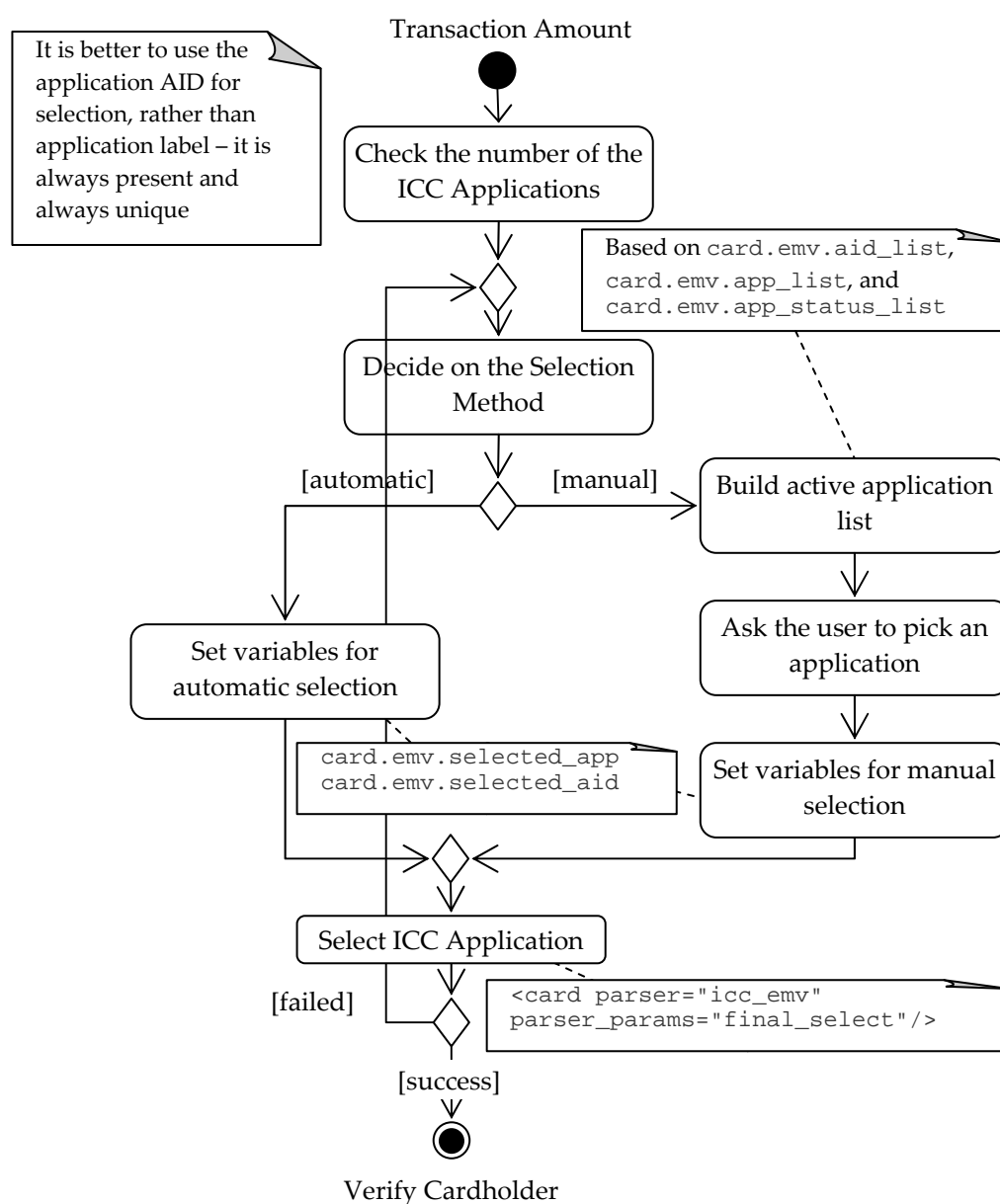
The terminal will only accept the AIDs that are present in the iccmv configuration file set for that terminal in the Incendo Online Gateway.

When an ICC card was inserted into the terminal (see **Initiate Transaction** block on page 71), the card parser filled the following list variables with the ICC data:

- `card.emv.aid_list`
list of all AIDs for all applications present on the card
- `card.emv.app_list`
corresponding application labels
- `card.emv.app_status_list`
contains the application status, "active" or "inactive", for each of the applications

If there is only one ICC application available on the card, it can be selected automatically. If there are several applications, the best practice is to display a list of active applications for user selection.

Figure 8 - 11 : ICC Application selection process



In order to select the application, you need to first set the `card.emv.selected_app` and `card.emv.selected_aid` variables. For automatic selection, both of these must be empty. For manual selection, one of these variables must be filled with a valid value for an active application, and the other must be empty.

The valid values for application AID are taken from the `card.emv.aid_list` variable. Application labels are contained in the `card.emv.app_list`. `card.emv.app_status_list` should be checked to see if particular application is active or not.

Once the required variables are filled, the application selection is done by calling the `final_select` function of the `icc_emv` card parser:

```
<card parser="icc_emv" parser_params="final_select"/>
```

A diagram of this process is shown on [Figure 8 - 11 on page 73](#).

Please consult the `tmlapp_icc.tml` file to see the implementation of this process.

ICC Verification

Once an ICC application has been selected, it is used to verify the cardholder.

To get a recommendation for a cardholder verification method, the program calls the `get_cvm` function of the `icc_emv` card parser:

```
<card parser="icc_emv" parser_params="get_cvm"/>
```

This will set the `card.parser.cvm` variable to one of four possible values:

- `"pin_online"`
online pin verification is required
- `"pin"`
offline pin verification
- `"no_cv"`
no cardholder verification needed
- `" "`
cardholder verification is complete

Based on this recommendation, the program executes the appropriate verification method.

The outcome (cardholder verification result) is put into the `card.parser.cvr` variable. This variable can be filled by the TML application or be updated by the card parser. It can have the following values:

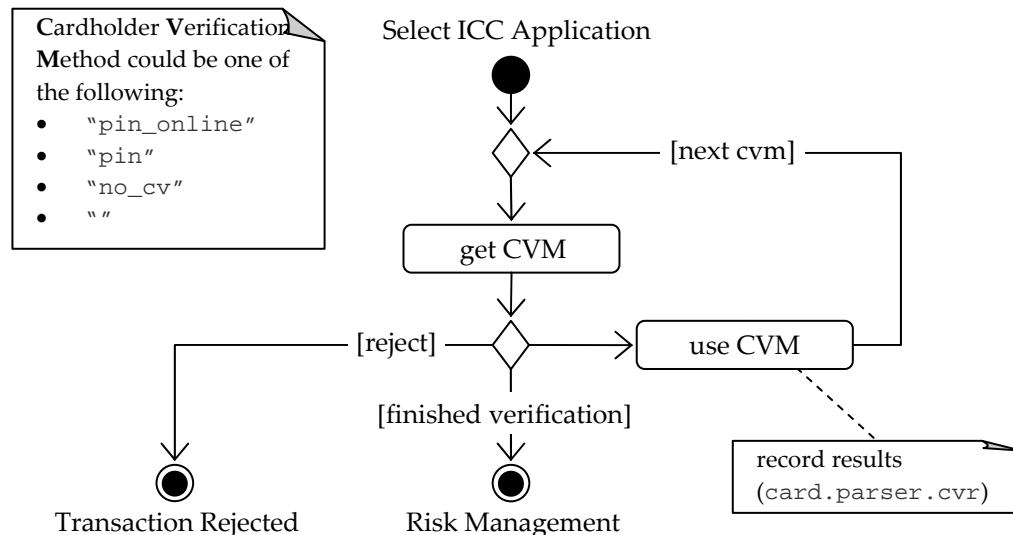
- `"bypassed"` – the method is not supported by the TML application
- `"ok"` – verification done by this method was a success
- `"ok_msg"` – same as `"ok"`, but additionally `"PIN OK"` message should be displayed
- `"failed"` – verification was unsuccessful
- `"pin_tries"` – wrong PIN was entered several times, and the number of tries exceeds PIN try limit

If the transaction should be rejected, the verification method sets the `card.parser.verdict` variable to `"reject"`.

Once a verification method has been conducted, unless `card.parser.verdict` is set to `"reject"`, the application should call the `get_cvm` function of the `icc_emv` parser again.

The verification is complete when `card.parser.cvm` variable is set to `" "`, and the application may proceed to the **Risk Management** block (see page 76).

The basic outline of the verification process is shown on [Figure 8 - 12 on page 75](#).

Figure 8 - 12: Verify Cardholder process

The sections below explain each of the verification methods.

Please consult the `tmlapp_icc.tml` file to see the implementation of application block.

Online Pin verification

If `card.parser.cvm` is set to the "pin_online", card pin PIN should be verified online by the acquirer.

Currently, BTMLPA does not implement this verification method. The `card.parser.cvr` is set to "bypassed" and another CVM is requested from the card parser.

Offline Pin verification

If `card.parser.cvm` is set to the "pin", card pin PIN should be verified offline.

To enter the pin, `pinentry` element is used:

```
<pinentry type="icc" prompt="Enter PIN" />
```

This way the PIN is entered using the underlying security features of the terminals and the operating system.

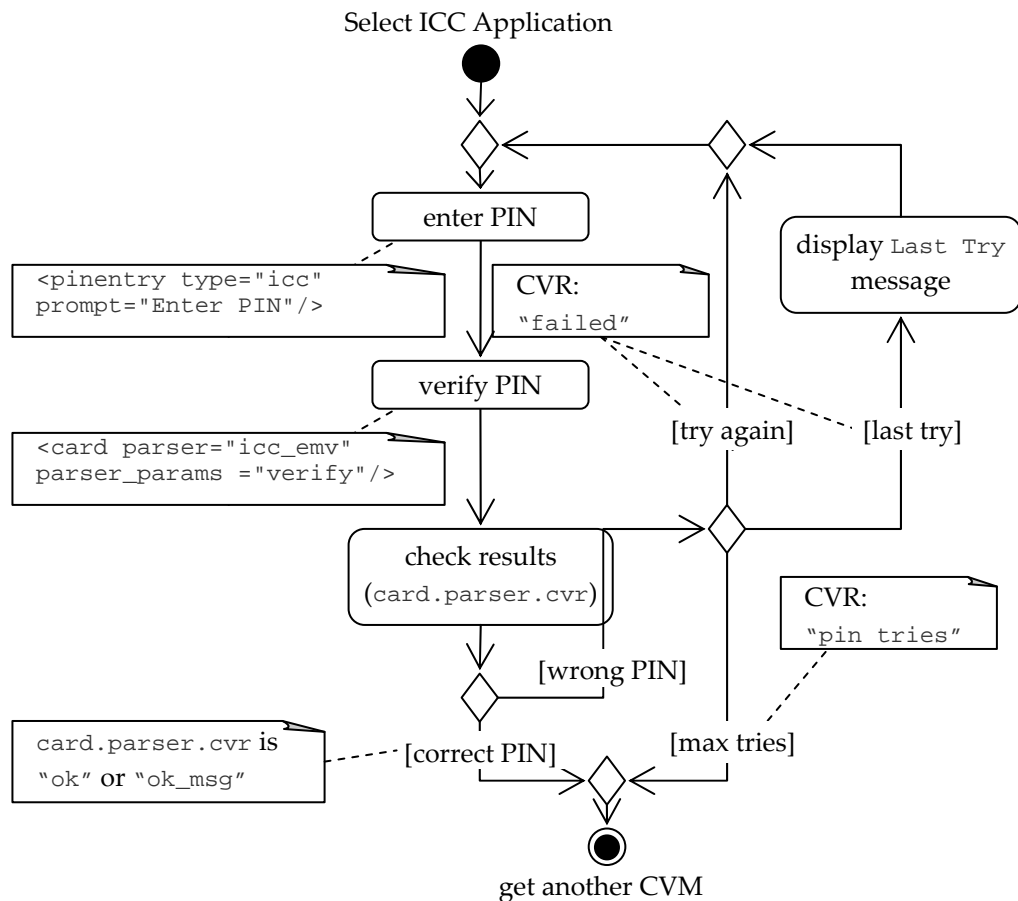
To verify the PIN, the `verify` command of the `icc_emv` parser is used:

```
<card parser="icc_emv" parser_params="verify" />
```

It sets the `card.parser.cvr` variable to one of the four values:

- "ok" – correct PIN. The program should call `get_cvm` function of the `icc_emv` parser again. The parser is likely to be satisfied with the verification and not ask for any more verification methods (see ["No more verification methods" on page 76](#)).
- "ok_msg" – same as "ok", but additionally "PIN OK" message should be displayed.
- "failed" – wrong PIN was entered, but the maximum amount of tries has not been exceeded yet, so another PIN entry attempt should be made. The program should also check the value of the `card.emv.last_attempt` variable. If this variable is set to 1, this means that only one more try remains. The customer should be notified of this before entering the PIN again.
- "pin_tries" – the number of wrong PIN entries has exceeded the maximum number of tries.

The pin entry should continue until a correct PIN is entered, or maximum number of tries is reached. Once the correct PIN has been entered, or the maximum number of PIN entry attempts has been exceeded, the program should call `get_cvm` function of the `icc_emv` parser.

Figure 8 - 13: Offline PIN verification

No Cardholder Verification

If `card.parser.cvm` is set to `"no_cvm"` no cardholder verification is required. However, the card parser may still ask for a signature check.

No more verification methods

If `card.parser.cvm` contains an empty string (`" "`), it signifies that no more cardholder verification methods are available. This result means that all CVM checks have been made and the application can progress to the **Risk Management** stage.

Signature Check

In addition to the verification methods described above, the card parser may ask for a signature check to verify the customer.

If the signature check is required, the `get_cvm` function of the `icc_emv` card parser will set the `card.emv.signature` variable to 1. Otherwise, it will be set to 0.

The signature will be checked after the customer receipt copy is printed.

Risk Management block

The purpose of the **Risk Management** block is to decide whether the transaction should be processed online, offline or rejected. It is done differently for magnetic card and ICCMV transaction.

Risk management for magnetic card transaction is performed by the `mag` parser using the `<card parser="mag" parser_params="risk_mgmt" />` command.

Risk management for ICCMV transaction is performed by the `icc_emv` parser using the `<card parser="icc_emv" parser_params="risk_mgmt" />` command.

However, the outcome of the process is the same. The parser analyses the amount and transaction type and previous submission attempts for the transaction and responds by assigning a value to the predefined variables `card.parser.verdict` and `card.parser.reject_reason` (if necessary).

The parser sets the pre-defined variable `card.parser.verdict` to one of the following values:

- "offline"
immediate online host authorisation is not necessary and the transaction can be performed offline
- "online"
online host authorisation is required
- "reject"
transaction is rejected. The pre-defined variable `card.parser.reject_reason` will contain the reason for the rejection

If the evaluation is successful, the application proceeds to **Submit Data** block (see page 77), if not – to the **Transaction Rejected** block (see page 86).

For magnetic card and manual entry transaction risk management the following screens are used:

- `pre_mg_risk`
- `mg_risk_mgmt`

These screens are in the `tmlapp.tml` file.

ICCEMV risk management is implemented using the `icc_rsk_mgmt` screen. It is contained in the `tmlapp_icc.tml` file.

Submit Data block

The purpose of the **Submit Data** program block is to submit to the host the variables necessary for the transaction processing. There are slight differences in which variables the host requires to process a magnetic card or ICCEMV transaction.

Submit Data block uses the following screens for a magnetic card and manual entry transaction:

- `mg_submit`, part of the `tmlapp.tml` file (Figure 8 - 15 on page 78)

For an ICCEMV transaction:

- `icc_submit`, part of the `tmlapp_icc.tml` file (Figure 8 - 16 on page 79)

The submission can be done either offline or online, depending on the outcome of the **Risk Management** process (see page 76).

Risk management verdict is contained in the `card.parser.verdict` variable. It can be one of the following:

- "offline"
- "online"
- "reject"

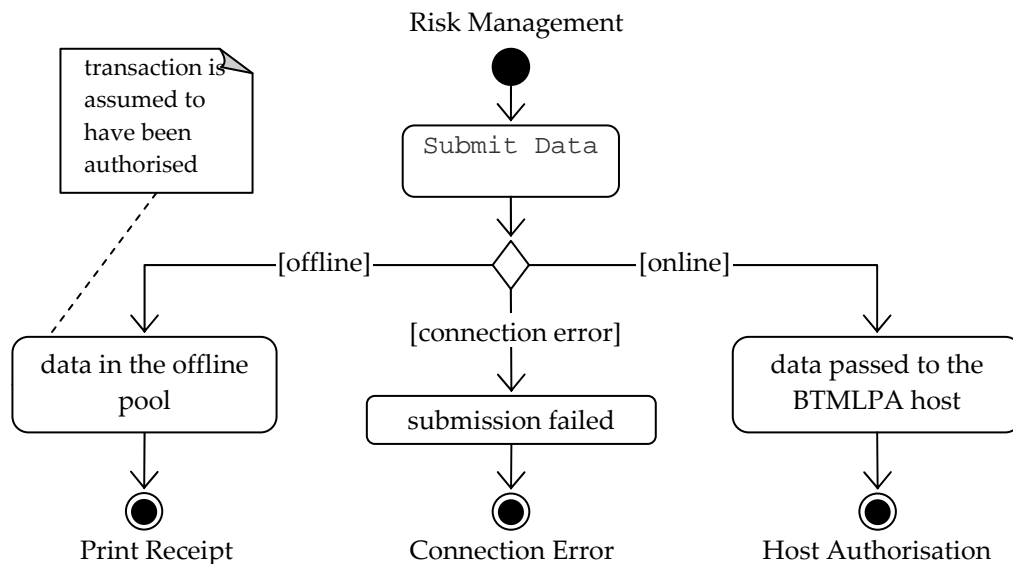
If the risk management verdict is "online", host authorisation is required. Therefore, the submission must be done online and the next step of the application will be the **Host Authorisation** block (see page 54).

If the risk management verdict is "offline", immediate authorisation is not necessary. The submission will be put into the offline pool and the host will process the transaction later. The next step will be **Print Receipt** (see page 85).

If the risk management verdict was "reject", the transaction was rejected, and there is no need to submit data to host.

The application must also be able to process connection errors. This is done by the **Connection Error** block (see page 79).

Figure 8 - 14 on page 78 shows possible outcomes of the transaction data submission process.

Figure 8 - 14: Submit Data outcomes

Data submission to the host is done using the `<submit>` TML command. It generates a form that will be sent to the host module component that is responsible for accepting terminal requests. This component is specified by the `tgt` attribute of the `<submit>` tag. The page that the application will go to in case of a connection error is specified by the `econn` attribute.

Both magnetic card and ICCEMV submissions have the same `tgt` attribute: `tgt="/btmlpa/auth"`. It refers to the `TxnAuthServlet`, which processes the transaction on the host side. For more information on the servlet, see [Host Authorisation block on page 80](#).

Note: for more information on the `<submit>` command, see *TML Application Development Guidelines*.

Incendo Online Microbrowser variable that is responsible for setting the submit mode is `oebr.submit_mode`. It is set to "online" if the submission must be done online, and "offline" if otherwise. The easiest way to do that is to set it to the value of the risk management verdict:

```
<setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict" />
```

Figure 8 - 15: Submitting data to host for magnetic card transaction - #mg_submit screen

```
<screen id="mg_submit" next="#receipt" cancel="#trans_abrt">
  <setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict" />
  <setvar name="receipt_id" lo="1" />
  <setvar name="payment.auth_code" lo="" />
  <setvar name="sbmt_url" lo="#mg_submit" />
  <submit tgt="/btmlpa/auth" econn="#conn_failed">
    <getvar name="oebr.submit_mode" />
    <getvar name="payment.trans_type" />
    <getvar name="card.pan" />
    <getvar name="card.issue_number" />
    <getvar name="card.expiry_date" />
    <getvar name="card.cardholder_name" />
    <getvar name="card.effective_date" />
    <getvar name="transid" />
    <getvar name="payment.amount" />
    <getvar name="payment.amount_other" />
    <getvar name="currency_code" />
    <getvar name="card.input_type" />
    <getvar name="card.mag.iso2_track" />
    <getvar name="payment.txn_date" />
  </submit>
</screen>
```

```

        <getvar name="oebr.time_zone" />
    </submit>
</screen>

```

Figure 8 - 16: Submitting data to host for ICC EMV transaction - #icc_submit screen of the tmlapp_icc.tml

```

<screen id="icc_submit" cancel="tmlapp.tml#trans_abrt">
    <setvar name="oebr.submit_mode"
lo="tmlvar:card.parser.verdict"/>
    <setvar name="receipt_id" lo="1"/>
    <setvar name="payment.auth_code" lo="" />
    <next uri="tmlapp.tml#receipt">
        <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="tmlapp.tml#trans_abrt"/>
    </next>
    <submit tgt="/btmlpa/auth" econn="#icc_conn_fld">
        <getvar name="oebr.submit_mode"/>
        <getvar name="payment.trans_type"/>
        <getvar name="card.pan"/>
        <getvar name="card.issue_number"/>
        <getvar name="card.expiry_date"/>
        <getvar name="card.cardholder_name"/>
        <getvar name="card.effective_date"/>
        <getvar name="transid"/>
        <getvar name="payment.amount"/>
        <getvar name="payment.amount_other"/>
        <getvar name="currency_code"/>
        <getvar name="card.input_type"/>
        <getvar name="card.mag.iso2_track"/>
    <!-- ICC data -->
        <getvar name="card.emv.aid"/>
        <getvar name="card.emv.aip"/>
        <getvar name="card.emv.auc"/>
        <getvar name="card.emv.atc"/>
        <getvar name="card.emv.aac"/>
        <getvar name="card.emv.tc"/>
        <getvar name="card.emv.argc"/>
        <getvar name="card.emv.iad"/>
        <getvar name="card.emv.tvr"/>
        <getvar name="card.emv.unumber"/>
    <!-- end of the ICC data -->
        <getvar name="payment.txn_date"/>
        <getvar name="oebr.time_zone"/>
    </submit>
</screen>

```

Connection Error block

The purpose of this application block is to decide what to do in case there was an error during submission.

If an error occurred during the online submission (`oebr.submit_mode = "online"`) it means that there was connection error. The application can either try to re-submit the data, do offline transaction or reject the transaction. **Risk Management** application block (see page 76) should be used again to reach that decision.

If an error occurred during the offline submission (`oebr.submit_mode = "offline"`) it means that the offline pool is full and can not accept any more messages. The application should reject the transaction.

This application block uses the following screens:

- `conn_failed`
is used for magnetic card and manual entry transaction. It decides whether

Risk Management should be performed again or the transaction should be aborted

- `icc_conn_fld` (of the `tmlapp_icc.tml` file) does the same for the ICCEMV transaction
- `print_pf` informs the user that the offline transaction pool is full. Application proceeds to the **Transaction Rejected** block (see page 86).

Host Authorisation block

The purpose of the **Host Authorisation** program block is to conduct online authorisation. The terminal application host-side module should do the following:

1. parse the TML data sent by the **Submit Data** block (see page 77)
2. generate an appropriate authorisation request and send it to the card acquirer
3. process the response from the acquirer
4. send an appropriate dynamic TML page to the terminal, depending on the acquirer's verdict.

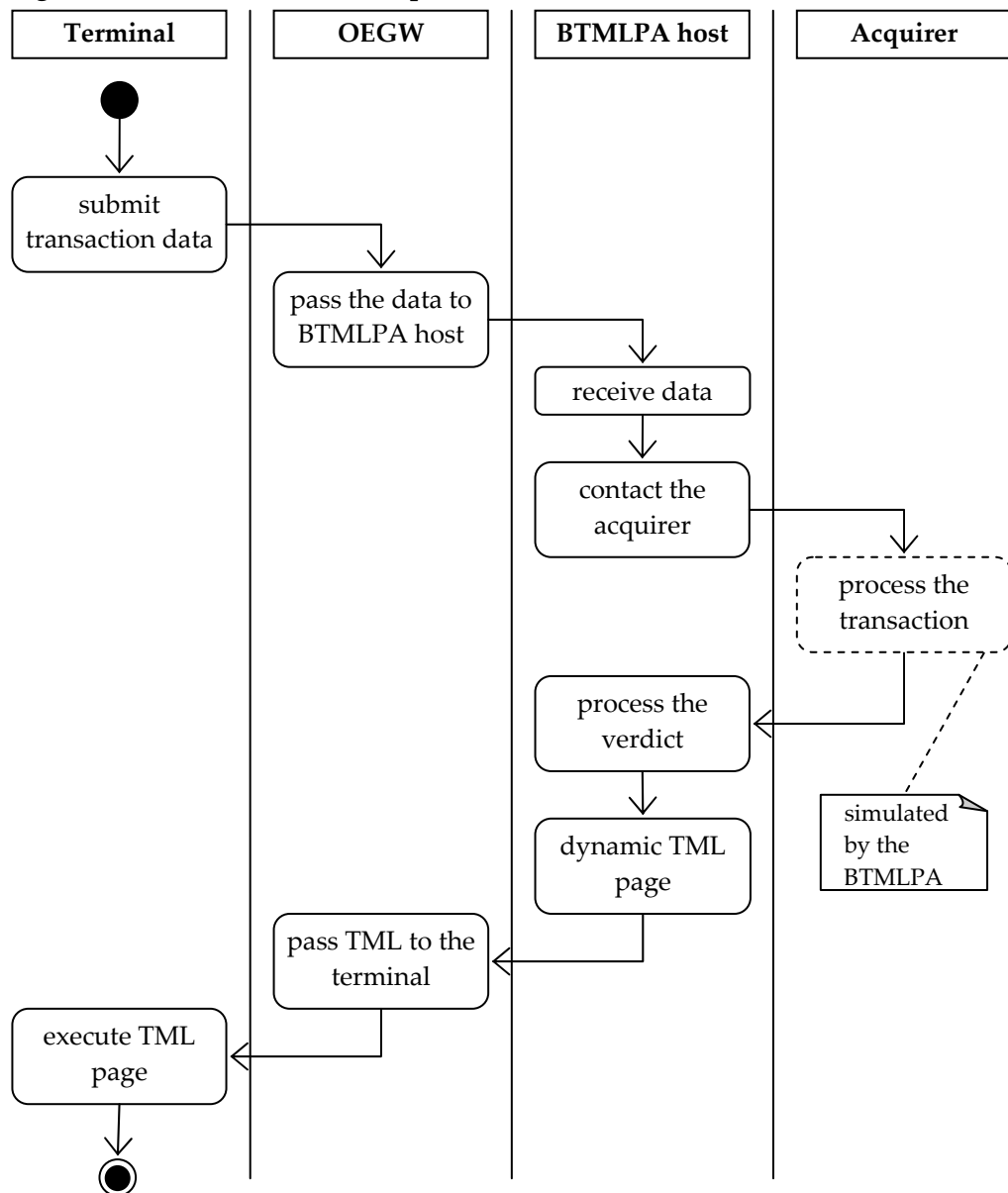
In BTMLPA Application Example, two Java servlets are used to emulate this process:

- `TxnAuthData` that works with the data structure that was sent by the **Submit Data** block, and
- `TxnAuthServlet` that processes the information
- `AuthResultData` defines the data structure used for the authorisation results

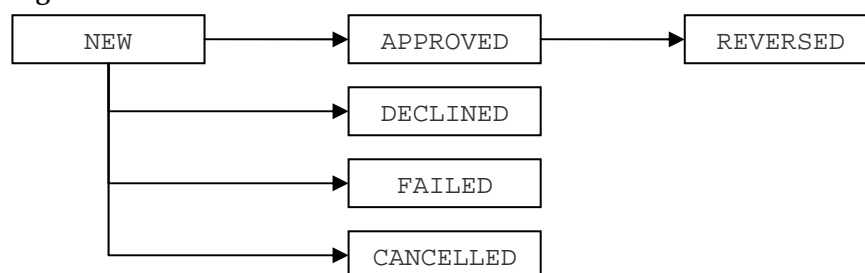
`TxnAuthServlet` emulates the interaction with the acquirer, and does the following:

- parses the TML post request.
- checks the card expiry date against the current date.
- checks the transaction amount against the defined top limit.
- selects the dynamic page that will be sent to the terminal.
- updates the BTMLPA database with transaction information and transaction status

[Figure 8 - 17 on page 81](#) shows the host authorisation procedure in a graphic form.

Figure 8 - 17: Host Authorisation process

The transaction/request processing states and the possible transitions between the states are shown on the following diagram.

Figure 8 - 18: BTMLPA transaction states

As soon as an authorisation request is received by the host module, it is assigned the state **NEW**. The transaction data, accompanied with the date and time, is stored in the application database. Then, the data is passed to the payment engine.

The payment engine converts the transaction data into the format appropriate for the acquirer host (for example, APACS 30, APACS 40, APACS 60, ISO 8583, or some other). Then the data are sent to the acquirer host for authorisation decision.

As a result of transaction data analysis, the acquirer can send back the following replies:

- **APPROVED**
The transaction has been approved. The acquirer also sends an authorisation code which is stored in the application database.
- **DECLINED**
The transaction has been declined. The acquirer also sends a message explaining the decline reason.

Within a short period of time, defined by the acquirer, the merchant has an ability to void the last of the approved transaction. In this case, the terminal part of BTMLPA sends a request with the transaction ID to the host-side module (see). If the acquirer approves the transaction voiding, the host module changes the transaction state from **APPROVED** to **REVERSED**. If otherwise, the transaction state does not change, and the terminal user is informed about the impossibility of transaction reversal.

If, when processing an authorisation request, the host module fails to connect to the acquirer host, the transaction state is changed to **FAILED**. The transaction in this case is not processed any further, and an error message is sent back to the terminal.

If, in a rare occasion, the terminal request (not a transaction) is cancelled by the merchant before the data was sent to the acquirer, the transaction state is changed to **CANCELLED**.

To inform the terminal user about the result of authorisation request processing, the host module generates a dynamic TML page using an appropriate template (see “[Dynamic Page block](#)” below). The page is then sent to the terminal as a reply.

Dynamic Page block

As the result of the terminal request, the host supplies a TML page to the terminal. This page must contain a valid TML code and can be generated in many ways.

The BTMLPA host-side module generates a dynamic TML page in one of two ways:

- selects a pre-made TML page (such as `reversed.tml`), or
- uses the FreeMarkerⁱ Java Template Engine Library to create a TML page on the fly

This TML page will instruct the terminal on what should be its next step.

The outcome will depend on the results of the **Host Authentication**.

The pre-made TML pages have a `.tml` extension, while the templates for FreeMarker use the `.dtml` extension.

Apart from the TML elements, the FreeMarker templates also have logical elements (`<#if [condition]>`, `<#else>`, `</#if>`). These elements tell the host module which parts of the template should be selected (and which should not) when constructing the screen and in which circumstances.

Additionally, it contains `${javaVariable}` sequences.

Prior to sending information to the terminal, the host module will remove the `<#if [condition]>`, `<#else>`, `</#if>` tags and replace the sequences `${javaVariable}` with the actual data.

The outcome of this process will be a valid TML page that the host will transmit to the terminal.

`icc_auth_rslt.dtml` template is used for ICCMV transactions

`mag_auth_rslt.dtml` template is used for magnetic card and manual entry transactions.

For example, if the BTMLPA host approved an ICCMV transaction, the template `icc_auth_rslt.dtml` will be parsed and the following `icc_auth_rslt.tml` page

ⁱ FreeMarker is an Open Source class library for Java programmers. It is a generic tool used to generate text output. For more information, consult <http://freemarker.sourceforge.net/>

will be generated (Figure 8 - 19 below). The sequences `${ javaVariable }` will also be replaced with the actual data.

Figure 8 - 19: one of the possible `icc_auth_rslt.tml` pages that may be generated from the `icc_auth_rslt.dtml` template

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">

  <screen id="auth_reply" class="c_center" timeout="1">
    <next uri="#scnd_auth"/>
    <display>
      <table height="100%" width="100%"><tr><td align="center"
valign="middle">
        Transaction Approved<br/>
        Auth Code: ${authResult.authorisationCode}
      </td></tr>
    </table>
    </display>
  </screen>

  <screen id="scnd_auth">
    <setvar name="payment.emv.arpc"
lo="${authResult.icc_arpc}" />
    <setvar name="payment.emv.issuer_script1"
lo="${authResult.issuer_script}" />
    <setvar name="payment.emv.issuer_auth"
lo="${authResult.issuer_auth}" />
    <setvar name="payment.txn_result" lo="1" />
    <setvar name="payment.auth_code"
lo="${authResult.authorisationCode}" />
    <next uri="${contextPath}/tmlapp.tml#receipt">
      <variant lo="tmlvar:card.parser.verdict" op="equal"
ro="reject" uri="${contextPath}/tmlapp.tml#submit_rev"/>
    </next>
    <tform>
      <baddata class="c_center"
next="/btmlpa/tmlapp.tml#submit_rev">
        <table border="2" height="100%" width="100%">
          <tr><td align="center" valign="middle">
            <getvar name="err.baddata_reason"/>
          </td></tr>
        </table>
      </baddata>
      <card parser="icc_emv" parser_params="auth"/>
    </tform>
  </screen>
</tml>
```

A different page (but still with the name `icc_auth_rslt.tml`) will be generated if the transaction is rejected.

Then, the terminal will process that page.

So, the next application step is determined by the code of the dynamic TML page sent to the terminal by the host.

The application will proceed to the **ICC Authorisation** block if the transaction is of the ICCEMV type, regardless of whether the host authorised the transaction or not. The ICC will make the final decision, taking into account the recommendation by the host.

If it is a magnetic card transaction, the application will proceed to the **Print Receipt** block (see page 85) if the transaction has been authorised, and to the **Transaction Rejected** block otherwise (see page 86).

ICC Authorisation block

This step is performed only if ICCEMV transaction is being processed online. An offline transaction does not require this step – the transaction is deemed to be authorised if **ICC Verification** (page 74) and **Risk Management** (page 76) has been completed successfully.

As mentioned previously, as the result of the **Host Authorisation** process (see page 80) the host sends a dynamic TML page to the terminal.

ICC authorisation is performed from within the dynamic page using the following command:

```
<card parser="icc_emv" parser_params="auth"/>
```

The card parser uses the following variables to decide whether to authorise the transaction:

- `payment.auth_code`
- `payment.auth_resp_code`
- `payment.txn_result`
- `payment.trans_type`
- `payment.emv.issuer_auth`
- `payment.emv.issuer_script1`
- `payment.emv.issuer_script2`

Therefore, the dynamic TML page should fill some of these variables before **ICC Authorisation** is conducted.

Authorisation Results block

Once the `icc_emv` parser processes the "auth" command, it sets the following variables:

- `card.emv.aac`
This variable is filled if the transaction was declined
- `card.emv.tc`
This variable is filled if the transaction was accepted
- `payment.emv.issuer_script_results`
Contains the result returned by the `icc_emv` parser after executing an issuer's script

In addition, if the authorisation has failed the variable `card.parser.verdict` is set to "reject". If this is the case, BTMLPA reverses the transaction.

Otherwise, the transaction was accepted and the application proceeds to the **Print Receipt** block (see page 85).

Remove Card block

This application block is used by the **Print Receipt** and the **Transaction Rejected** blocks to remove the ICC card from the reader.

This block consists of only one screen, `remove_card` (Figure 8 - 20 below).

The user is asked to remove the card, and the application waits for the card to be removed.

This is done using the `wait_remove_card` command of the `icc_emv` parser:

```
<card parser="icc_emv" parser_params="wait_remove_card"/>
```

Once the card removed, the application proceeds back to the **Print Receipt** (see page 85) or to the **Transaction Rejected** block (see page 86), depending on the value of the `screen_after_call` variable.

Figure 8 - 20: #remove_card screen code

```
<screen id="remove_card" class="c_center"
next="tmlvar:screen_after_call">
  <tform>
    <card parser="icc_emv" parser_params="wait_remove_card"/>
    <prompt>
```

```

<table height="100%" width="100%">
  <tr><td valign="middle" align="center">
    Please, Remove Card
  </td></tr>
</table>
</prompt>
</tform>
</screen>

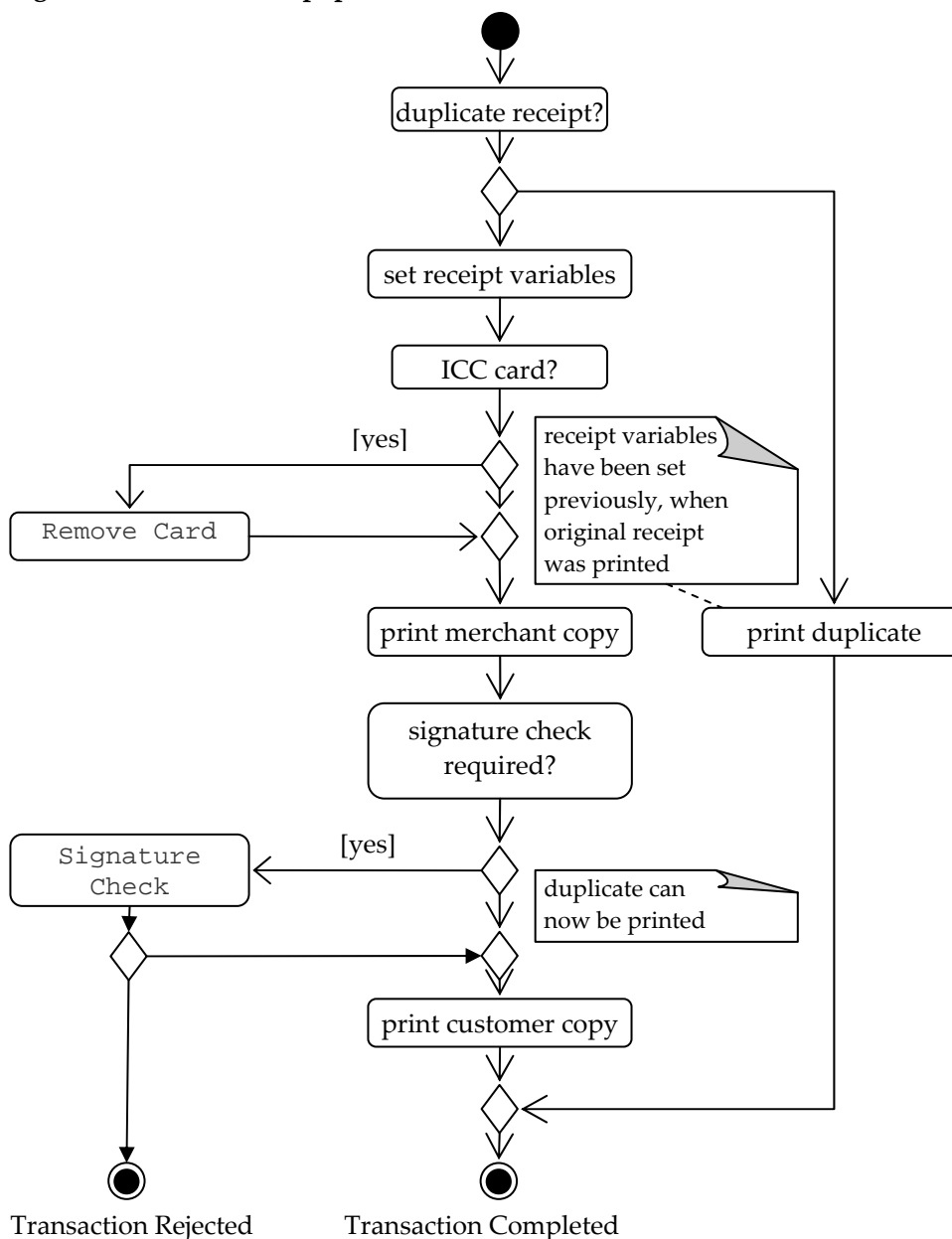
```

Print Receipt block

The receipt printing functionality is implemented using `btmlpa_receipt.tml`.

The application logic is shown in Figure 8 - 21 below.

Figure 8 - 21: Print Receipt process



Signature Check block

It is assumed that for magnetic card and manual entry transactions, signature check is always required.

For ICCMV transactions, during the previous application steps, the `card.emv.signature` variable would have been set to 1 if signature check is necessary; otherwise it would have been set to 0.

Once the `MERCHANT COPY` receipt is printed, the merchant should ask the customer to sign the receipt and then verify the signature against the one at the back of the card.

The merchant then selects whether the signatures match.

If they do not, the transaction is rejected and should be reversed to update BTMLPA database.

If the signature is fine, the application proceeds back to the **Print Receipt** block (see page 85) to print a `CUSTOMER COPY` receipt.

Signature Check block consists of the `mg_sig_val` screen (Figure 8 - 22 below).

Figure 8 - 22: #mg_sig_val screen code

```
<screen id="mg_sig_val" class="c_center">
  <display>
    <table height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Is Signature OK?<br/>
        <a href="#compl_txn">YES</a><br/>
        <a href="#sub_rev_prep">NO</a>
      </td></tr>
    </table>
  </display>
</screen>
```

Transaction Completed block

The application informs the user that the transaction has been completed successfully.

The transaction is finished; it returns to the **Transaction Preparation** block (see page 66) to start a new transaction.

Transaction Completed block consists of the `compl_txn` screen.

Transaction Rejected block

This application block is used if transaction is rejected or cancelled. The user is informed of the transaction outcome. If the ICC card is still in the card reader, the user will be asked to remove the card (see “[Remove Card block](#)” on page 84).

The application returns to the **Transaction Preparation** block (see page 66) to start a new transaction.

The following TML screens are used:

- `reject_trans`
- `trans_abrt`
- `void_trans`
- `close_trans`

assert screen

This is the screen for troubleshooting the application.

To use it, when choosing from multiple variants, set `#assert` up as the target URI of the `<next>` element (Figure 8 - 23 below).

Figure 8 - 23: using #assert screen

```
<next uri="tmlapp.tml#assert">
  <variant lo="tmlvar:card.emv.last_attempt" op="equal" ro="0"
uri="#enterpin"/>
  <variant lo="tmlvar:card.emv.last_attempt" op="equal" ro="1"
uri="#enterpinlast"/>
</next>
```

If there are other variants that you have not thought of, application will proceed to the `assert` screen.

Therefore, when `assert` screen is used by your application, it indicates that there are problems with the implementation of your application logic.

The screen displays the “Assertion Error” and the name of the screen where the error occurred for a short period of time. Then, the application switches to the **Transaction Preparation** block (see page 66) to start a new transaction.

Figure 8 - 24: #assert screen code

```
<screen id="assert" timeout="3" class="c_center" next="up.tml">
  <display>
    <table border="2" height="100%" width="100%">
      <tr><td align="center" valign="middle">
        Assertion Error<br/>
        on screen "<getvar name="oebr.prev_screen"/>"
      </td></tr>
    </table>
  </display>
</screen>
```
