TML Application Development Guidelines

V3.0

Service Business Unit

Tuesday, 4 August 2009

Ingenico - 190-192 avenue Charles de Gaulle - 92200 Neuilly-sur-Seine
Tél. 33(0)1 46 25 82 00 - Fax 33 (0)1 47 72 56 95

Contents

About this document

1

This document is designed to serve as a comprehensive source of information on the Terminal Markup Language (TML) and its use for developing content for Ingenico payment devices operating in the Incendo Online environment.

Target audience

The document is addressed to software application developers using or intending to use TML. It is assumed that the reader is familiar with and has experience in using XML, HTML/XHTML, and/or WML[a].

Typographical conventions

The following conventions are used in this document:

Notation	Description
`Courier`	Text that appears on the terminal screen, programme code, file and directory names, TML tags and attributes, Uniform Resource Locators (URLs), names of variables and their values, and so on.
?	The specified element can be included zero or one time inside the described parent element, for example, `<base>` ?. This and the following conventions are used in the "Related elements" section of a TML element's description.
+	The specified element must be included at least once inside the described parent element, for example, `<dt>` +.
*	The specified element can be included any number of times inside the described parent element, for example, `<link>` *.
|	Separates alternative items, for example, `<call_func>` | `<display>` | `<print>` | `<submit>` | `<tform>`.

The element descriptions have the following standard content:

- **Description**
  Provides a short description of the element.
- **Related elements**
  Lists the child and parent elements of the current element.
- **Attributes**
  Explains the attributes of the element.
- **Example**
  Provides a simple example illustrating the element usage.

Related documents

This guide is intended to serve as a comprehensive reference manual relating to the technologies presented.

[a] XML – **Ex**tensible **M**arkup **L**anguage XHTML – **Ex**tensible **H**yper**t**ext **M**arkup **L**anguage HTML – **H**yper**t**ext **M**arkup **L**anguage WML – **W**ireless **M**arkup **L**anguage

Overview of TML

2

This chapter provides a brief introduction to Terminal Markup Language.

TML and markup languages

TML is an XML language which is based on a specific XML schema and is used for developing TML applications. TML, for the terminal MicroBrowser, is just like HTML for a web browser.

TML is used to define the content to be rendered on the terminal screen or printed with the terminal printer as well as the data to be posted by the terminal to the Application Server.

TML is significantly based on XHTML. Some portions of TML code, especially those which describe the data to be displayed or printed can be written in XHTML. However, some XHTML elements and concepts are not permitted and some new elements and attributes have been introduced in TML.

TML concepts common for small screen hand-held devices are derived from WAP WML. For example, TML *pages* and *screens* are similar to WML *decks* and *cards* but in TML these concepts have been generalised to reflect the Ingenico terminal and card payment specifics.

TML and payment cards

TML has been designed with payment in mind. It provides enhanced capabilities for working with major types of payment cards which are currently on the market. It supports the cards with magnetic stripe and chip cards, also known as ICC EMV (**I**ntegrated **C**ircuit **C**ards) or smart cards.

Using TML you can easily design an application which supports transaction flows for both card types. TML features specific elements `<card>` and `<pinentry>` that provide interaction with terminal card readers and a PIN pad. Your application can read the data from the card, perform risk management and PIN verification and authorise transactions using these two elements.

TML and CSS

TML supports Cascading Style Sheets (CSS). CSS describes how documents are represented on the terminal screen with the MicroBrowser. Through the use of CSS, TML application developers control the way how the TML pages look without adding new markup. TML CSS is a subset of CSS, appropriate for small terminal screens; it omits the features that are not useful in that context.

TML page with a reference to an external style sheet separates the content of the document or application from the visualisation. The style sheet is downloaded once and cached by the MicroBrowser for use with subsequent screens, which speeds up the overall page rendering. Moreover, terminal has an embedded stylesheet which is used by MicroBrowser by default when an external stylesheet is not provided for the page; see "Embedded terminal software" on page 39.

Many aspects of the TML page appearance, such as positioning, fonts, text attributes, borders, margin, alignment and flow can be defined in the style sheet.

Design guidelines

3

Before designing an application

When designing an application for Ingenico terminals sketch the application logic defining the screens as well as the supposed transaction flow model which will satisfy your service needs.

Consider the guidelines provided in the following sections for a hassle-free and straight-forward application design.

Defining TML screens

If you are used to working with HTML and WML, you will find some aspects of TML familiar, while the others will appear to be very different. TML is defined as an XML schema, and, therefore, requires more careful implementation than a piece of HTML code.

The most significant difference between TML and HTML is that a TML page (or a file with the `.tml` extension) is divided into *screens*. Each screen is processed as a separate entity by the terminal MicroBrowser. This is done to make things easier and avoid traffic overhead between the terminal and Application Server.

TML screen represents a portion of TML data that can be either rendered on the terminal screen or printed on the terminal printer or submitted to the Application Server at one time.

This concept of screens requires you to explicitly define navigation instructions for every TML screen, so the users will be able to easily access the required information and make their input appropriately.

Every screen is identified by URI which is used as a screen name. The URI is required to handle navigation between the screens within the page.

The following basic types of screens are introduced with TML, see "Screens and navigation" on page 17 for details:

- **display screen**
  Always rendered on the terminal display. It is scrollable and can contain links, graphics and terminal forms for accepting user input.
- **print screen**
  Is similar to the display screen but its contents are printed using an embedded terminal printer rather than rendered on the terminal display.
- **terminal form screen**
  Is used for accepting user card data or PIN from a card reader or terminal PIN pad.
- **submit screen**
  Is used for submitting user input to the Application server.
- **functional screen**
  Is used for defining calls to the MicroBrowser routines

In TML 1.0 you cannot apply style rules directly to the elements in your page or screen markup. Instead, you can link a CSS file which contains all necessary style rules to your TML page. A style rule can be associated with a class of elements or with an individual element referred by id.

Organizing TML code

A well-organized TML code is more readable and is easier to modify or maintain. Below are several techniques that can be used to organize the code.

Split the code into several TML pages

Instead of having a single TML page that contains all of your application, separate your code into several pages. Each page should implement a particular aspect of the program's functionality, such as receipt printing, presentation layer etc. Use variables to pass data between various TML pages.

This makes the code more readable and easier to maintain.

Additionally, when you modify your program, the terminal will need to upload only the changed TML page - a relatively small amount of data.

Also, you may be able to re-use your pages for other applications, with only minimal modifications.

Comment your TML code.

A well-commented code is much more readable and easier to maintain.

Comments are removed by the OEGW when transferring TML from the server side to the terminal, so extensive comments do not affect application performance.

The comments are placed between `<!--` and `-->` tags (see Figure 3 - 1 below).

Figure 3 - 1: TML comment

```
<!-- this comment will be removed when TML code is sent to the
terminal -->
```

Keep business logic separate from the presentation layer

Separating the business logic and the presentation layer makes it much easier to change the look and feel of your application, or to port it to the terminals with different screen sizes or color capabilities.

Use CSS as much as possible

You can use styles with most TML elements, and define those styles in the CSS file attached to your TML page.

This makes it a lot easier to make changes to your presentation layer. Instead of changing each screen individually, you only need to change data in one location.

When you use styles with descriptive names, such as `error`, `message` or `bold`, your code is more readable.

Additionally, styles provide your TML applications with a consistent look.

Figure 3 - 2 below shows some styles that will be used for other TML examples.

Figure 3 - 2: CSS entries

```
table.message{text-align: center; vertical-align: middle;
height: 100%; width: 100%;}
span.message{ font-size: medium;}
```

Share presentation screens between various parts of your application

It is a good idea to keep your presentation screens in one location, and share them between various parts of your application. This way, if you decide to modify the look of your display or print screens, you only need to make changes in one spot.

This is done by using variables that contain the data to display or print, and a variable that is set to the URI of the next screen.

For instance, consider the TML code in the example in Figure 3 - 3 on page 10. A display screen is shared between two other screens, using two `string` type variables – `message_text` and `next_screen`. First, `Processing Transaction` message is displayed, then `Transaction Successful`.

Figure 3 - 3: Example of sharing a presentation screen

```
<screen id="process" next="#display_msg" >
  <setvar name="message_text" lo="Processing Transaction"/>
  <setvar name="next_screen" lo="#success"/>
</screen>

<screen id="success" next="#display_msg" >
  <setvar name="message_text" lo="Transaction Successful"/>
  <setvar name="next_screen" lo="#main_menu"/>
</screen>

<screen id="display_msg" next="tmlvar:next_screen">
  <display>
  <!-- table is used to set the text alignment -->
    <table class="message">
      <tr><td>
      <!-- span is used to set the text properties -->
        <span class="message"><getvar
name="message_text"/></span>
      </td></tr>
    </table>
  </display>
</screen>
```

This technique works well for simple messages, and it makes TML code more compact and more readable.

For text with complex formatting, dedicated display and print screens should be used.

Use volatile variables

TML allows you to specify how variables should be stored. This is set by the `volatile` property of the `<vardcl>` element.

- **volatile** variables (`volatile="yes"`) are stored in the terminal RAM and can be quickly accessed and modified. This is a default setting.
- **non-volatile** variables (`volatile="no"`) are saved in the terminal flash memory. Non-volatile variables will not be destroyed, if for instance, terminal power is disconnected or the terminal is rebooted. But, every operation that involves non-volatile variables is much slower than the same operation with the variables, stored in the terminal RAM.

To improve the speed of your application, always try to use the volatile variables.

Note: for more information on the declaration and the use of variables see "Variables and constants" on page 21

User interface design

The introduction of TML screens, variables and CSS creates a variety of new user interface constructions. Though TML contains fewer elements than WML and XHTML, the visual rendering of elements can also be modified in different ways with CSS. TML offers more possibilities for service providers to make their services more attractive while, at the same time, adding a layer of complexity and therefore more usability challenges.

Follow the navigation model

You should provide an easy-to-use user interface built around a consistent, easy-to-learn navigation model. This is much more important than using many of the display features inherited by TML from XHTML. Offer concise, precise, and quick information. Avoid wrapping the core service content up in superfluous pages or splash screens; however, you may wish to display a company logo or other branding highlight to create familiarity. Entering data is significantly more challenging and

time consuming with a terminal, so construct your TML application with a minimum need for user input. Consider whether it is possible to ask the user to choose from form screens rather than typing in a selection.

Design the navigation hierarchy

The navigation model is the way the user walks through the TML screens that shape a service, interacting with the application via links, menus, and data input. When designing a navigation model, consider the following:

- Keep the navigation model consistent throughout the service
- Avoid creating a service that is too "deep." It is often difficult for users to comprehend a service with more than four or five sequential screens
- Use a possibility to navigate back to the previous page of the service so that users can easily return to the accomplished step. However, it is not recommended to use this feature excessively
- Specify which screen to switch to if a user presses the Stop button or the input was incorrect.

Provide informative feedback for user actions

Provide proper feedback for user actions and for error and problem situations. Reducing the number of steps in navigation should not increase the user's feeling of insecurity, e.g. confirmation pages for user actions (especially related to payment) should be provided, even if it requires pressing Enter one more time. If the confirmation page is missing, the user may feel that they need to check if the action has taken place — this leads to even more clicks. Users should feel that they are controlling the system. If problems arise, users should be informed what to do next. Errors can be prevented by explaining the format of the expected input and by marking mandatory fields.

Perform a usability test

It is always a good idea to perform a usability test for new applications. People who have not been involved in the design or development of an application tend to notice potential usability problems that are often not obvious to those who know the design pretty well.

Incendo Online Terminal Simulator is a great help for application developers since there is no need to build costly hardware and software environments to test the application. For information on how to install and run Incendo Online Terminal Simulator, refer to "Installing and running Incendo Online Terminal Simulator" in *Incendo Online Desktop Installation Guidelines*.

Presentation layer design

When designing the presentation layer for your application, keep in mind that your application may be used on different terminal models with differing display capabilities.

Design for small-sized terminal screens

Though TML offers great opportunities for developing attractive user-friendly applications for small terminal screens, consider the following:

- Make sure there is visible content when the user enters the page.
- Use consistent styling for all TML pages in a service. Consistency increases learnability – especially for users who will return and use the service repeatedly.
- Try to avoid vertical scrolling wherever it is possible. If the text you want to display on the screen do not fit, place the most significant information on the top of the screen.

- Since horizontal scrolling is not allowed for terminals, always design the screens that are not wider or longer than the display of the targeted device.
- Use alignment properties (left, right, and center) with elements to increase readability, but do not use more than two or three on a single page as this affects the user's ability to grasp the application structure.
- Avoid overusing text decoration properties such as bold, as it affects readability on small displays.
- Avoid using long, complex words when shorter, more precise alternatives are available.

Use tables carefully

MicroBrowser supports the use of tables and nested tables on the TML page. Developers should be careful when defining cell widths. Also, table borders should not be too thick, since in a device of limited display size, the border width can easily consume enough pixels to make the actual content area too small to be recognised.

When using nested tables ensure that the total table width is the same as the sum of the widths of the individual columns plus borders and cell spacing.

Avoid using very deeply nested tables in order to improve the MicroBrowser performance.

Optimizing connection with the host

An optimized connection with the host will reduce your transaction time. You do that by reducing the terminal-host communication and, if modem connection is used, by managing the modem connection manually.

Reduce the terminal-host communication

To save time spent on establishing the terminal-host connection and on transferring data between the terminal and the host, you should reduce the terminal-host communication.

The strategies that can be employed to achieve that are described below.

Enable caching of pages

Place all of the resources that are not likely to be changed often into the terminal cache, to speed up terminal-host interaction.

MicroBrowser places all static TML pages in the cache by default. To allow caching of dynamic pages, use the `cache` attribute of the `<tml>` element.

On the host side of your application, the `no-cache` HTTP header directive can be set by the Application server to define that the pages are not to be cached.

Cache uses the “least recently used” algorithm which considers that items used least are removed from cache first.

It is a good idea to cache images and external CSS for all TML pages.

Reduce the number of times you communicate with the host

Each time you communicate with the Incendo Online Gateway, it takes some time to establish a connection before you can start transferring data. This is particularly relevant for modem connection.

Try to group the information you need to transfer to the host. The fewer times you connect to the Incendo Online Gateway, the less time you spend establishing the connection.

Keep your presentation layer in the static TML pages

It takes time to load images and code and to parse TML pages. Static TML pages have already being loaded and parsed – using them is quicker. You can use the techniques described in 0 on page 9 to separate the presentation layer from the business logic.

Reduce the amount of data you send to the host and receive from it

To minimize the connection time between the host and the terminal, only transfer the minimal amount of data necessary. Ideally, within the dynamic pages you receive from the host, you should only set some variables and then transfer control to the static TML pages.

The terminal will spend less time parsing the TML and transferring data. Additionally, memory management is better when we operate mostly with static (cached) resources.

This is particularly relevant if you use a slow connection, such as modem or GPRS.

Use image libraries

Images that are not likely to be frequently changed should be grouped into image libraries. The advantage is that an image library is that it is loaded and cached as a group. This means that the load time is quicker and memory is managed better than if the images were loaded individually.

However, if you want to modify one of the images within an image library, the whole image library must be recreated and reloaded. Therefore, images that will be often changed should be outside of image libraries.

See "Image libraries: increasing you application's performance" on page 36 for more information.

Manage modem connection manually

You may have to manually manage your modem connection to reduce the connection time, by using pre-dial, persistent connection and manual disconnect

Use pre-dial functionality when you work with the modem.

Modem connection and handshaking takes time. If you start the pre-dial at the appropriate time in your TML application, modem connection will be established while other parts of your application are being executed.

This will minimize the time the user will have to wait before being able to submit data to the host.

To start the modem pre-dial set the `oebr.connection.state` variable to `connected`:

```
<setvar name="oebr.connection.state" lo="connected" />
```

Use persistent connection and manually disconnect when transaction is finished

If your application communicates with the host several times during one transaction, it is a good idea to use the persistent connection. A non-persistent connection may require several modem sessions, and each session will go through the connection and handshake.

Persistent connection is established once, and will be maintained until manually disconnected.

So, you can use pre-dial (see above) to establish your connection before submitting data to the host, conduct your transaction, and manually disconnect once you finish communicating with the host.

You can enable persistent connection through the terminal configuration menu, or by setting the `oebr.connection.endstate` to `up`. Setting it to `down` will disable the persistent connection.

You manually disconnect the modem by setting the `oebr.connection.state` variable to `down`.

Authoring in TML

4

This chapter provides an overview of the TML language and its most important elements, illustrated with code examples. See "TML elements reference" on page 52 for the detailed syntax description of a particular element or attribute.

TML pages and screens

TML markup is organised into a collection of *screens* wrapped into *pages*. A screen specifies a single unit of interaction with a user, terminal peripheral or the host side of your application. For example, a screen can contain text (displayed or printed on the printer), a menu or input form which requires user input or a list of variables to submit to the Application Server which does not. Logically, a user navigates through a series of TML screens, reviews their contents, enters information requested, makes choices and moves on to another screen. Besides the screens visible to a user, you usually need to design auxiliary screens for assigning variables, interacting with terminal card parsers and Application Server.

Static and dynamic pages

A TML application consists of the root page and, possibly, a number of auxiliary TML pages referenced from the root. A TML page is a well-formed XML document which typically includes the XML declaration, page header and a sequence of linked TML screens. The page contents is enclosed into `<tml>` and `</tml>` tags.

Static TML pages contain fixed TML markup and are similar to those used in XHTML. Typically, you develop mostly static pages.

Dynamic pages contain dynamic markup which can be generated by MicroBrowser or the host part of your application. Pages generated by MicroBrowser usually contain transaction data to be sent to the Application Server for analysis, e.g. variable values required for the transaction processing. Pages generated at the host side might contain the result of data analysis, e.g. the result of online transaction authorisation, as in the example below:

Figure 4 - 1: Dynamic TML page

```
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
  <screen id="auth_ok" next="/iccemv/iccemv.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>
```

Dynamic pages take precedence over static pages.

URIs and TML

Like XHTML, TML uses a URI (Uniform Resource Identifier) to refer to the location of a TML page or a screen within a page. A URL in TML has the same form as that in XHTML; that is `http://domain_name/path/file_name#offset`

TML uses HTTP protocol for URIs since the files and pages are requested from the Application Server over HTTP.

The offset of a URI tells the MicroBrowser which screen to switch to on the page. The screens are identified by their `id` attribute. To switch to a particular screen within a page, you specify the screen name (id) as the offset of the URI.

You can use absolute and relative addresses in TML. Absolute addresses include the protocol, domain name and the full path to the target file located on the Application Server, for example:

```
http://localhost:8080/tml/examples/btmlpa/tmlapp.tml#man_entry
```

You can shorten the URLs by using relative addresses:

- If the target is located on the different page you can use the path relative to the root TML page starting with / (slash) symbol, e.g.
  `<next uri="/btmlpa/tmlapp_icc.tml#icc_get_cvm">`.
  See also "Specifying the root URI for a TML page" on page 17.
- If the target is a screen located on the same page you can reference it by the screen id starting with # (hash) symbol, e.g.
  `<next uri="#read_amount">`

If the particular screen id is not specified in the URL, the MicroBrowser displays the first screen on the TML page.

If the target TML page has not been specified explicitly, it is assumed that the requested file is `index.tml`.

Important: The length of TML file names including the file name extension (`.tml`) must not exceed 13 characters. The same refers to the names of image and CSS files. File names are case-sensitive.

URIs and data scopes

URIs in TML in most cases implicitly define the scopes of objects and data such as:

- variables (see "Variables and constants" on page 21)
- logs (see "Logs" on page 28)
- offline posts[a].

The scope is understood as a page or a collection of TML pages where an object or certain portion of data can be referenced and manipulated.

The scope is defined by the part of the URI preceding the file name (that is, `domain_name/path`) of the page on which an object or data is 'created' (a variable or a log is declared or the submit screen that lead to creation of an offline post is located). So, for example, object/data created on the same page or on different pages with the same `domain_name/path` have the same scope.

All pages' URIs may be represented as an inverted (upside-down) tree with a root at the top and the branches and leaves – where more specific ('longer') URLs are located – closer to the bottom.

For variables and logs, the scope starts from the page[b] on which those variables and logs are declared and goes down all the way along the corresponding branches to the leaves of the URI tree. Thus, the variables and logs can be referenced on pages located in the same directory and on pages with more specific URIs but can not on the ones with more general URIs.

With the offline posts, the situation is the opposite. Their scope also starts at the URI of the page where a post was created. However in contrast to variables and logs, it goes *up* the branch of URIs to the tree's root. Consequently, offline posts can be manipulated on the page where it was 'created' and on all pages with more general URIs. However, it will be impossible to access these posts from pages lower down in the hierarchy, that is, with more specific URIs.

At the top of the page hierarchy is the Embedded TML Application (see "Embedded terminal software" on page 39) referenced as `emb://embedded.tml` since the symbolic part of its URL `emb://` is considered as a root of the URL tree. Variables

[a] If the data (say, payment transaction data) intended for Application Server (see "Submit screen" on page 18) can not be sent right away, they are temporarily stored in the terminal memory as an offline post.

[b] To be more precise, the scope starts from the *directory* in which the page is located.

and logs declared within this application have global scope and the application can work with the offline posts created by all other applications.

Special URI values

There are some URIs that have special functionality. These are:

- `back` - a link to the previous TML screen
- `menu` - a link to the URI defined by the `menu` attribute of the `<screen>` element
- `cancel` - a link to the URI defined by the `cancel` attribute of the `<screen>` element. Additionally, if this URI is used within a form screen, the values that have been entered into the input fields are cleared - input is cancelled.
- `exit` - exits the MicroBrowser

For example

```
<a href=”back” >Return to previous TML screen</a>
```

will create a link to the previous TML screen.

Application header

The header includes the following:

- XML declaration
- `<tml>` root element
- `<head>` element

XML declaration

You can use XML declaration at the beginning of each TML page, for example, to explicitly specify encoding as a value of an optional `encoding` attribute:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

If you omit the `encoding` attribute in XML declaration, the ISO-8859-1 encoding will be assumed by default.

The use of XML declaration is not mandatory and can be omitted.

<tml> root element

Since TML is a schema-based XML subset, DOCTYPE element is omitted. Following the XML declaration (if present), is the root `<tml>`, which must specify the `xmlns` namespace attribute. The value of this attribute is a unique URI pointing to a TML page that defines the namespace for all elements and attributes valid within your page. Specifying a wrong `xmlns` attribute results in a document that can be well formed but not strictly schema conforming.

```
<tml xmlns="http://www.ingenico.co.uk/tml">
```

<head> element

The `<head>` element defines the document header with preliminary information about your page, including:

- List of the documents referenced on the page (e.g. an external style sheet or other TML documents). Use `<link>` to specify URLs of the referenced documents.
- Optionally, two screens to handle presses of the **Menu** and **Cancel** keys. If the user presses either of those keys, the application will switch to the appropriate screen. Use `<defaults>` to define URLs of these screens.

```
<head>
  <link href="style.css" rev="stylesheet"/>
  <defaults menu="/index.tml" cancel="#init_prompt"/>
</head>
```

Specifying external style sheets and images

TML allows specifying optional external style sheets by defining one or more `<link>` elements.

CSS is the only way to provide styles to the documents; XHTML `<style>` elements are not allowed in TML.

Note: It is strictly recommended to reference all the images within the `<head>` element.

It is important since the referenced images will be loaded and cached in the terminal before they are actually used.

Specifying the root URI for a TML page

The `<base>` includes the `href` attribute which defines the root for all relative links on your page. If the element is omitted, it is considered that the root is an URI of the current page.

Specifying Menu and Cancel screens

Ingenico terminals are equipped with the **Menu** and **Cancel** keys[a]. To handle presses of these keys in your application, you can define which screen the application should go to when either of the keys is pressed.

Using `<defaults>`, you can specify the URIs of the screens to switch to at the whole page scope. To redefine the Menu and/or Cancel screens at a particular screen's level, you can use the `menu` and/or `cancel` attributes of the `<screen>` element.

Screens and navigation

A screen is a portion of information that can be processed by the terminal MicroBrowser at one time. MicroBrowser can render the screen on the display, print it contents with the embedded terminal printer, sent it to the Application Server or use it for terminal configuration purposes.

The screens are flow-based and can contain both inline and block-level elements, see "Inline, Block, NoForm and Flow" on page 50. The name of the screen is specified by its `id` attribute.

TML introduces the following elements that define the screen content:

- `<display>` for defining display screens
  The content is rendered on the terminal display allowing the user to browse it.
- `<print>` for defining print screens
  The content is printed with the embedded terminal printer.
- `<tform>` for defining TML forms
  The content represents a TML form for accepting user input from payment cards or terminal PIN pad.
- `<submit>` for defining submit screens
  The content represents the data to be submitted to the Application Server. Usually, the data includes the values of the variables assigned during processing TML forms.
- `<call_func>` for defining calls to the MicroBrowser C functions

Display and Print screens

The elements defining these screens are similar to block-level and inline elements used in XHTML. Moreover, the contents of these screens can often be a pure XHTML.

Note: The contents of `<form>` cannot be printed using `<print>`.

[a] For more information on terminal keypad keys, their labels, colours, and so on, refer to *Basics of Working with Incendo Online-enabled Terminals*.

TML Forms

TML forms are used to handle the card data collected by TML-specific <card> and <pinentry> elements.

TML forms control the overall transaction flow for supported card types. Every transaction step is wrapped into <tform>. See "TML and payment cards" on page 41 for more details.

The data gathered on the screen (usually a list of TML variables) is always submitted using a special submit screen which might not necessarily follow the screen containing <tform>.

Note, that XHTML <form> is used within display screens in the same way as in XHTML.

Submit screen

The data entered by a user from keyboard or terminal PIN pad using <input>, <textarea> or <pinentry>, or read from a card using <card>, as well as any other data can be submitted to the Application Server for processing and possible reply. For most transactions you will need to support this functionality in your application. The data is submitted as a list of variable/value pairs. This list of variables is wrapped into <submit>.

Aside from the variables, you should provide some auxiliary data required to complete the submission, such as the destination server URI and type of submission using the element attributes.

The data can be submitted either online or offline. Data submitted offline are stored as offline posts. The type of submission is defined by the predefined variable oebr.submit_mode. If offline transactions are impossible (e.g. when the number of postponed requests is exceeded), the data is posted online.

Online submissions are posted to the Application Server immediately. Then, by default, MicroBrowser switches to the screen specified by the URI in the <next> element or the next attribute of the <screen> element. The host part of TML application can, however, override this behaviour and specify a different screen to switch to in a dynamic TML page sent to the terminal in reply.

<econn> (or econn attribute) can be used to specify the screen to switch to, if a connection error occurs during submission of data to the Application Server.

Functional screen

Use functional screens for calling terminal operating system C functions.

A function call is defined using the <call_func> element.

Functions cannot accept parameters, so you need to specify only the name of the called function (using the name attribute).

You should also provide the URI of the screen to switch to, if the call was successful (using <next>) and another URI if it was not (using <error>).

See "<call_func>" on page 54 for the list of available functions.

Switching between the screens

MicroBrowser switches to the next screen when one of the following events takes place:

Figure 4 - 2: Switching between TML screens

Event	Destination URI
User activated a hyperlink on the display screen.	URI specified by the href attribute of the corresponding <a> element

Event	Destination URI
• The terminal has completed printing the content of the print screen. • The time specified by the timeout attribute has elapsed (and there are no hyperlinks within the screen). • The C function called by the function screen has been successfully processed. • The data specified in the submit screen have been successfully sent to the Application Server and the Server 'accepted' the data. • When a display screen containing no hyperlinks is active, a user presses a key, and there is no particular URL assigned to that key (by means of the menu, cancel, or key attribute(s)). • User selected the Submit or Reset button in the form.	• If there is no `<next>` element: URI specified by the screen's next attribute • If the `<next>` element contains no `<variant>` elements or none of such elements contains a true logical statement: URI specified by the `uri` attribute of the `<next>` element • If there is a `<variant>` element containing true logical statement: URI specified by the `uri` attribute of the first of such `<variant>` elements
The C function called by the function screen has returned an error.	URI specified by the `uri` attribute of the `<error>` element or the `uri` attribute of the first of its child `<variant>` elements containing true logical statement – if there is such element
The data specified in the submit screen have been sent to the Application Server but the Server did not 'accept' the data.	URI specified in the server's response
Connection with the Application Server during data submission was lost.	URI specified in `<econn>` or one of its child `<variant>` elements.
A logical expression specified in a `<variant>` element is found to be true.	URI specified by the `uri` attribute of the corresponding `<variant>` element. The `uri` attribute of the `<variant>`'s parent element (`<next>`, `<error>`, or `<econn>`) in this case is ignored
User pressed the **Menu**[e] key and the `menu` attribute is defined for the screen or page.	URI specified by the `menu` attribute
User pressed the **Cancel** key and the `cancel` attribute is defined for the screen or page.	URI specified by the `cancel` attribute
User pressed a key and the active screen contains a `<variant>` element assigning a URI to this particular key.	URI specified by the `uri` attribute of the corresponding `<variant>` element
User has reached the maximum allowed number of tries for entering the data.	URI specified by the next attribute of `<baddata>`

[e] On *Ingenico 8550* terminals, this is the **F2** key.

A destination URI can be represented either by a variable or constant or can be chosen as a result of evaluating a series of logical expressions. This is defined by the data type of the attribute which specifies the URI.

For example, a URI specified in `href` attribute can only be a constant; however, within `menu`, `cancel` and `next` attributes the URI can also be a reference to a variable.

For `<next>`, `<error>` and `<econn>` elements you can define a number of conditional statements using child `<variant>` elements. As an alternative to defining logical conditions, you can use the `<variant>` elements for assigning terminal keypad keys to screens, or, to be more exact, their URIs. In this case, if a user presses a key specified in the `<variant>` element, Incendo Online MicroBrowser jumps onto a screen assigned to that key. For more information, see "<next>", "<error>", "<econn>", and "<variant>" on pages 81, 66, 65, and 111 respectively.

Displaying text

When defining text content for small-sized terminal displays, consider the following guidelines:

- Use simple, short and precise words.
- Avoid inserting empty lines between text sections on a TML screen which might cause the vertical page scrolling. A user can be misled assuming that he or she watches the whole screen.
- Avoid using different text styles and font sizes on the same TML screen. Two or three styles and sizes are usually more than enough.
- Select the fonts carefully. You cannot be aware that a terminal has a specific font style or font size. Terminals, with a limited available memory, have a limited collection of fonts installed (often just a single font).
- Avoid using uppercase characters excessively. It might decrease the text readability.

Tables

You can design virtually any layout using only TML `<p>`, `<div>`, and `<span>` elements. However, using tables you can design a screen layout in a tabular manner, which can look more efficient.

Aligning the text is easier when the screen content is placed within a table, especially when you need to design a print screen, e.g. for printing a receipt.

Tables are made up of columns and rows and can be as simple as one containing a single row and a column and as complex as one that utilises headers and footers. Tables in TML are similar to those in XHTML. You can use the following TML elements for defining the structure elements of a table:

Element	Description
`<table>`	The element makes up the entire table structure.
`<colgroup>`	The element assigns columns in the table to column groups.
`<col>`	The element assigns attribute values to the individual columns within `<colgroup>` element.
`<thead>`	The element defines the header section of the table.
`<tfoot>`	The element defines the footer section of the table.
`<tbody>`	The element defines the table body section.
`<tr>`	The element makes up the row structure of the table.
`<th>`	The table header element makes up the cells of the table. Always used along with `<tr>` element.
`<td>`	The table details element makes up the cells of the table. Always used along with `<tr>` element.

When designing the tables consider the following guidelines:

- Do not use deeply nested tables.
- Do not define tables wider than the target device display.
- Do not define table cells higher than the target device display.

Images

Images are valuable for TML application design. When inserting images consider the following guidelines:

- Use `.bmp` format for the images you plan to use in your TML application. This format is supported by TML MicroBrowser.
- Do not use the images exceeding the size of the terminal display to avoid scrolling. Remember, also, that the size of the images that can be printed with the terminal printer is limited[f].
- Explicitly specify the height and width for the images using `height` and `width` attributes to allow the browser to reserve necessary memory space.
- Use the `alt` attribute to specify alternative text, which is displayed if the image could not be loaded.

Note, that in TML the `<img>` element *does not* support the `align`, `border`, `hspace`, `vspace`, and `usemap` XHTML attributes.

Important: The length of image file names including file name extensions must not exceed 13 characters. File names are case-sensitive.

Variables and constants

Variables can be predefined and user-defined and can have different scopes.

Types of variables

The type of variable is set when the variable is declared (see "<vardcl>" on page 109).

TML supports variables of the following types: `date`, `string`, `integer` and `opaque`.

Additionally, there is a derived string list type. It is a string variable that contains a list of items, separated by the `;` character.

String

String variables contain a sequence of characters. The default value of a string is `""`.

Integer

Integer variables contain numeric values. They can be either signed or unsigned. The integers are signed by the default.

The minimum value of a signed integer is `-2147483648`. The maximum value of a signed integer is `2147483647`.

The minimum value of an unsigned integer is `0`. The maximum value of a signed integer is `4294967295`.

The default value of an integer is `0`.

Note: we recommend using only signed integers for consistent formatting

Date

Date variables contain date-time information. The default value of a date is `1970/01/01 00:00:00` (with `YYYY/MM/DD hh:mm:ss` format)

[f] The width of the image that you want to be printed must not exceed 384 pixels, while its maximum height is limited to 128 pixels.

Note: there is a special value of `0000/00/00 00:00:00` for date variables. If all fields of a variable are explicitly set to zero, this variable acts as an empty string for `<getvar>` and `<input>` elements.

Opaque

Opaque variables contain binary data. They can be used for storing binary cryptographic data exchanged by the terminal and the payment card, for storing images etc. The default value of an opaque variable is `""`.

Predefined variables

Pre-defined variables are declared by the embedded TML pages of the MicroBrowser. Therefore, these variables can be used by the other TML pages without declaring them first.

Predefined variables are used to store:

- terminal network connection settings
- terminal and Incendo Online Gateway authentication data
- payment card and transaction data
- information about the terminal and that related to system log
- bar code scanner-related data

as well as other data defining:

- how the terminal should interact with Incendo Online Gateway requesting updates of cached TML resources and configuration data for card parsers (see "TML and payment cards" on page 41)
- what information needs to be sent to other (third-party) applications running in the terminal
- what sounds and in which cases the terminal should produce, and
- other aspects of terminal behaviour.

Some of the predefined variables are read-only. Values of the others, if appropriate, can be modified in your applications (see also "Scopes of variables" below and "Permissions" on page 51).

For the complete list of predefined variables and their description, refer to "Pre-defined TML Variables" on page 117.

Predefined variables are declared in a TML application called Embedded Terminal Application which, along with the default CSS and default logo image, permanently resides in the terminal memory; see "Embedded terminal software" on page 39.

User-defined variables

You can create your own variables. Each variable is declared using the `<vardcl>` element. The `<vardcl>` elements are placed after the `<head>` element but before the `<logdcl>` elements (if any) and the first `<screen>` element of your TML page. If your application contains more than one page, the page where the variable is declared defines the variable scope or "visibility", see "Scopes of variables" below.

Note: The pages containing non-declared variables are rejected during TML parsing.

Scopes of variables

The scope of a variable is a page or a collection of pages where a variable exists and its value can be referenced[g]. The variable cannot be referenced outside of that scope.

You define the scope *implicitly* by declaring the variable in the `<vardcl>` element. The part of URI preceding the file name of the page where the variable is declared (that is, `http://domain_name/path/`) defines the variable's scope. In other words,

[g] All that is said in this section about the scopes, access permissions and redefinition of variables also fully applies to TML logs, see "Logs" on page 28.

the scope of a variable is a directory containing the page on which the variable is declared along with all subdirectories of that directory. So, for example, a variable declared on the page `http://localhost:8080/tml/page_1.tml` can be referenced from all pages whose URIs start with `http://localhost:8080/tml/`, that is:

- `http://localhost:8080/tml/page_2.tml`
- `http://localhost:8080/tml/custom-apps/page_1.tml`

and so on. On the other hand, a variable declared on the page `http://localhost:8080/tml/custom-apps/page_1.tml` will not be visible from `http://localhost:8080/tml/page_1.tml` since it will have a narrower scope limited to `http://localhost:8080/tml/custom-apps/` and more specific URIs.

Predefined variables have a "global" scope since they are declared on the embedded TML page referenced as `emb://embedded.tml`; the `emb://` is considered as a root for all custom TML pages.

Controlling the scope of and access to a variable

You can limit the scope of a variable and restrict access to it from pages other than the one on which the variable is declared. For this purposes the `perms` attribute of the `<vardcl>` element is used. By means of this attribute you can, for example,

- limit the variable's scope just to the page on which this variable is declared
- specify that the variable can be accessed only from pages located in the same directory as the one on which the variable is declared
- specify that on some or all pages other than the one on which it is declared the variable's value can not be modified.

In addition to that, the `perms` attribute can be used to specify whether or not the variable can be redefined within a narrower scope.

By default – if the `perms` attribute is not present in the in the `<vardcl>` element – a variable can be referenced and its value can be modified on any page within the variable's scope. The variable can also be redefined in any of the narrower scopes. For information on how to use the `perms` attribute, see "Permissions" on page 51.

Redefining a variable in a narrower scope

If the `perms` attribute allows doing so, a variable can be redefined in a narrower scope.

The variable is redefined exactly in the same way as it is declared – by means of the `<vardcl>` element. After the variable is redefined in a narrower scope, it becomes totally independent (within this narrower scope) of the variable with the same name declared in a broader scope.

To illustrate how redefinition works, let us assume that the variable `my-var` is declared on the pages `http://localhost:8080/tml/page_1.tml` and `http://localhost:8080/tml/custom-apps/page_1.tml`. This situation is treated as if two different variables were declared. Though both variables in this case can be referenced by the same name (that is, `my-var`), one of them exists in `http://localhost:8080/tml/custom-apps/` (and all its subdirectories), while the other – in `http://localhost:8080/tml/` (and all its subdirectories) with the exception of `http://localhost:8080/tml/custom-apps/` (and all its subdirectories).

Volatile and non-volatile variables

There is a memory management mechanism for TML variables which defines a variable life cycle. When declaring a TML variable you specify (using the `volatile` attribute of the `<vardcl>` element) where MicroBrowser should store its value – in the terminal flash memory or in the terminal RAM. See also the `<vardcl>` element's description for details.

- `volatile="yes"`
  Volatile variables are stored in the terminal RAM which is fast but flushed every time the terminal is switched off. If you need to preserve the variable value between terminal reboots, you should declare the variable as non-volatile.
- `volatile="no"`
  Non-volatile variable are stored in the terminal flash memory which is rather slow but preserves the stored data even if the terminal is switched off or rebooted.

By default – if the `volatile` attribute is not present in the declaration of a variable – the variable is considered to be volatile.

Note: if a non-volatile (`volatile="no"`) variable is declared from within a non-cached page, it may still be destroyed if the terminal cache is cleared. To avoid this situation, declare your persistent (non-volatile) variables from within the TML pages where the `cash` attribute of the `<tml>` tag is set to `"allow"`.

Assigning values to variables

While processing TML pages, MicroBrowser can change the values of TML variables.

It happens when one of the following is true:

- MicroBrowser processes a TML form with the data entered by the user.
- The assignment is explicitly defined by the `<setvar>` element. You need to consider the variable scope to avoid possible errors.

Using variables

Insert variable references whenever you need to:

- render the variable value on the terminal display or print it using the terminal printer
- post the variable value to the Application Server
- use the variable value as an operand in `<setvar>` expressions that might change the value of another variable
- use the variable value in logical expressions that might switch the TML application flow control, e.g. in `<variant>`.

Use the `<getvar>` element for inserting values of variables into display, print and submit screens.

For logical expressions, the constants and variables can be specified as expression operands using `lo` and `ro` attributes. Always use prefix `tmlvar:` to specify that the attribute value is a reference to a variable, e.g.:

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

If you omit the prefix, MicroBrowser will interpret the specified value as a constant.

Formatting and de-formatting

Formatting is transformation of values (for example, values of variables) according to a specified pattern. Formatting patterns are data type-specific and, depending on the data type, may, for example, look like this: `c*`, `0*`, `YYYY/MM/DD`, etc.

Formatting changes data presentation and, as a rule, is used to control how values are displayed or printed. Formatting may result in adding, removing, replacing and hiding of various data fragments.

To illustrate, let us assume that the amount of a payment transaction is stored as a number of pence in a variable of the `integer` type. Let this amount be equal to `1299`. Depending on the circumstances, you may want to show (display or print) this amount in a different way, for example, as `GBP: 12.99`. To achieve this, you specify a formatting pattern which would change the presentation of the variable's value in the desired way. In the case being discussed the required pattern would look like this:

`GBP: 0*.00`. This pattern would specify that the text `GBP:` followed by a blank space should be added before the value and the dot (`.`) should be inserted in front of the last two digits.

It should be noted that formatting does not include changing fonts, colours and so on. That is, formatting does not change just the appearance of data, though, in a sense, it does. It rather changes the data presentation somehow.

De-formatting is a reverse transformation with regard to formatting. It is used when certain text information needs to be converted to a value of the integer or the date type.

De-formatting as well as formatting also implies the use of a formatting pattern. In the case of de-formatting, however, the pattern is used in the 'reverse direction': it tells how to extract the value of interest from a string of characters.

Going back to the example given earlier, the string of text `GBP: 12.99` de-formatted with the pattern `GBP: 0*.00` would give `1299`. (In this case the text `GBP:` and the dot in front of the last two digits would be removed.)

For more information on formatting patterns and their usage, see "Formatter" on page 47.

Casting TML variables

When designing expressions which use variable references, make sure that the right and left operands are of the same data type. Otherwise, prior to calculating the expression, MicroBrowser attempts to *cast* (change) the value of the expression to the data type specified in `name` attribute of `<setvar>` element. For example, the integer `8105` can be converted to the string `"8105"`.

Casting assumes that the data of one type is transformed to another type according to some predefined rules. Usually, the value that is convorted into another data type must correspond to a valid formatter patterns for that data type

Note that some data type combinations cannot be cast at all or there can be some limitations defined for a particular data type casting. Below is the casting table for TML data types.

From/To	Integer	String	Opaque	Date
Integer		An *Integer* value is converted to its decimal *String* representation using integer formatting. By default `-0*` formatter is used e.g. `8105` –> `"8105"`. It may be modified using the `format` attribute.	Not supported.	An *Integer* value when converted to *Date* is interpreted as a number of seconds since `1970/01/01 00:00:00`. Unsigned integer value is used.

From/To	Integer	String	Opaque	Date
String	A *String* value can be converted to *Integer* only if it corresponds to a valid formatting pattern for the integer variable. `-0*` formatter is used by the default, eg. `"8105"` –> `8105`. It may be modified using `format` attribute. If the result of the cast is a value that is outside of the integer range, it is set to the maximum or the minimum integer value.		Not supported.	A *String* value can be converted to *Date* only if it corresponds to a valid formatting pattern for the date variable. `YYYY/MM/DD`, formatter is used by default, eg. `2008/08/28`. It may be modified using the `format` attribute.
Opaque	Not supported	An *Opaque* value is converted to a *String* representation as a sequence of ASCII characters. These characters will correspond to the opaque formatter setting e.g. `"64 AC BD 0E"` if `hex` format is used. The formatter may be changed using the `format` attribute. Default formatter is `base64`.		Not supported
Date	A *Date* is converted to the *Integer* value equal to the corresponding number of seconds since `1970/01/01 00:00:00`. If the date is less than this value, the resulting integer value will be `0`.	A *Date* is converted to the *String* according to the date formatter pattern. `YYYY/MM/DD`, formatter is used by default, but it may be modified using `format` attribute.	Not supported.	

Comparison of variables

Variables values are compared when evaluating logical expressions. In TML only two values of the same type can be compared. One of the values should be a variable; the other can be either a variable or a constant.

Before the comparison proceeds, the value of the right operand of the expression is cast to the type of the left operand. If one of the operands is a constant, its value is cast to the type of the operand defined by a variable.

When comparing integers and dates, the biggest and the most recent values win.

Note: The result of comparison of two dates may depend on the formatters specified for corresponding variables. See "Remarks" and "Example" in "<variant>" on page 111. Also note that all data related to the time zone (if any) is ignored.

The strings are compared in the following way:

1. First, the lengths are compared; the string which is longer, that is, contains more characters is considered "bigger".
2. If the lengths are the same, the strings are compared character by character from left to right: first, the leftmost characters are compared. If they are the same, the second characters are compared, and so on until different characters in the same position are found. Then the rule for separate characters evaluation is applied: the character with a higher ASCII code is considered "bigger." ASCII codes increase in the following row: digits, uppercase letters, lowercase letters, so that
   `"0" < "1" < ... < "A" < "B" < ... < "a" < "b" < ... < "z".`

For example, the following statements about string values are true:

- `"huge" < "small"` (the second of the strings is longer)
- `"Smith" < "smith"` (the strings have the same length but the ASCII code of the first character in the second string is higher)
- `"smile" < "smith"` (the strings have the same number of characters; the first three characters in both strings are the same, but the ASCII code of the fourth character in the second string is higher)

Opaque values are compared in the way similar to how strings are compared. The longer the value, that is, the more bytes its binary representation has, the bigger it is.

Values with the same length are compared byte by byte starting from the left until the first different byte in the same position is found. The opaque value is considered bigger if the value of such byte is higher.

Branching of the TML code

TML program logic is implemented using <variant> elements, which can be used to branch the TML code. This allows a TML program to react to user input, such as key press, or to events that change values of TML variables.

Note: see page 111 for a description of the <variant> element.

A very common strategy is to assign a specific value to a particular TML variable depending on a chosen condition. Later in the program flow, this variable is evaluated within the `<variant>` elements and the application proceeds to a particular TML screen, depending on the value of the variable.

For example, suppose you assign a value `"first"` to `var1` if you want your application to proceed to the `#first_page` screen, `"second"` if you want the next screen to be the `#second_page`, and `"third"` if you want the application to go to the `#third_page`. You could use the code in Figure 4 - 3 below to implement this logic, but it would not be the best practice.

Figure 4 - 3: incorrect branching by enum values

```
<next uri="#third_page">
    <variant uri="#first_page"  lo="first"  ro="tmlvar:var1"
op="equal" />
    <variant uri="#second_page" lo="second" ro="tmlvar:var1"
op="equal"/>
</next>
```

It is much better to use the following TML code within your page:

Figure 4 - 4: correct branching by enum values

```
<next uri="#error_page">
    <variant uri="#first_page"  lo="first"  ro="tmlvar:var1"
op="equal" />
    <variant uri="#second_page" lo="second" ro="tmlvar:var1"
op="equal"/>
    <variant uri="#third_page"  lo="third"  ro="tmlvar:var1"
op="equal" />
</next>
```

This way an error page will catch assertion errors in your TML code. This error page could look something like Figure 4 - 5 below.

Figure 4 - 5: sample error page

```
<!-- When the screen is reached it is most likely a programming
error -->
<screen id="error_page" timeout="3" next="index.tml">
  <display>
    Assertion Error<br/>
    on screen "<getvar name="oebr.prev_screen"/>"
  </display>
</screen>
```

This will make tracking errors in your code a lot easier. All possible values of `var1` are listed explicitly. If a value that we have not considered is assigned to `var1` (or we just incorrectly write one of values in our TML code) then `#error_page` will be displayed and we can solve the error.

If we use "incorrect branching" (as in Figure 4 - 3 on page 27) we just go the `#third_page` and it will be hard to find this error.

This convention could also be used for branching by several variables or by non-enum variables, if we describe all possible ways of branching.

Logs

Logs in TML are used to monitor values of different sets of TML variables.

To create a log, you declare it by means of the `<logdcl>` element (on page 79).

When declaring a log, you can:

- Specify the set of TML variables whose values you want to monitor
- Define the log access permissions
- Link the log with the external CSS file that defines the appearance of the log when it is displayed or printed
- Specify various layouts – rendering patterns – for log records each defining what log information is to be displayed or printed and how this information should look when it is displayed or printed

You can do the following with a log:

- Add records, see "<logrec>" on page 81
- Show the log on the terminal display or print it, see "<log>" on page 78
- Submit the log to the Application Server, see "<submit_log>" on page 99
- Clear the log, see description of the `clear_log` function in "<call_func>" on page 54

System log

One of the examples of a TML log is the system log, which is implemented in Embedded TML Application (see "Embedded terminal software" on page 39). This log is used to monitor exceptional situations (system errors) detected by Incendo Online MicroBrowser.

The system log can be accessed from the Embedded Application only.

The predefined TML variables related to the system log are listed and described in "Variables related to working with TML logs" on page 132.

Accepting user input

Use TML forms for accepting user input. The `<card>` and `<pinentry>` elements appear in terminal form screens enabling various different interactions with a card, see "TML and payment cards" on page 41; while `<input>` and `<textarea>` elements appear in display screens (within the `<form>` element) to accept the data entered by the user manually (using the terminal keypad or touch screen), see "Manual input" below and "Form processing" on page 30.

Pin entry

Input of the secure pin should always be done by using the `<pinentry>` element. This element is used within the terminal form screens (see <tform> on page 104). The application switches to the secure mode and uses the security features of the terminal to accept user input and encrypt it according to the specified encryption algorithm. This ensures that the pin is never stored in an unencrypted form, and therefore can not be compromised. See <pinentry> on page 82 for more details.

Manual input

The elements `<input>` and `<textarea>` are the main building blocks of a TML form (`<form>`). Using these elements, you can define controls for data input. The controls are the data fields and/or options that allow the user to enter information into the form. For each control, the entered data is assigned to the specified variable.

You can also specify some restrictions and/or additional checks for the entered data, for example, for matching maximum/minimum value, verifying the PIN associated with a card, and so on.

TML supports the following controls:

- Fields and areas for entering text (`<input type="text" .../>` and `<textarea .../>`).
- Fields for entering text where the text is masked (`<input type="password" .../>`).
- Fields for entering (non-negative) integer numbers (`<input type="number" .../>`).
- Fields for entering the dates (`<input type="date" .../>`).
- Check boxes (`<input type="checkbox" .../>`), switching elements that represent the options which can be either selected or not selected.
- Radio buttons (`<input type="radio" .../>`), elements that represent one of the possible options. Radio buttons are usually grouped so that only one radio button within the group can be selected.
- List boxes (`<input type="list" .../>`), elements representing lists of options a user can choose from.
- Buttons for submitting and resetting the form (`<input type="submit" .../>` and `<input type="reset" .../>`).

Form processing

When a form defined by the <form> element is displayed on the terminal screen, MicroBrowser switches into the form processing mode that defines specific navigation rules within the form.

When you design navigation in a form that contains <input> and/or <textarea> elements consider the following:

- A user navigates within the form until he or she presses the **OK**, **Menu** or **Cancel** keypad key. Pressing **Menu** or **Cancel** makes MicroBrowser jump onto a screen defined by the menu or cancel attribute. If the **OK** key is pressed, the next screen in most cases is defined by the next element or attribute.
- Pressing **Up** and **Down** keys will move the focus onto the previous or next <input> or <textarea> element. An element in focus is marked visually somehow: in most cases such element has a thickened border.
- Depending on the element type, pressing **OK** will:
  - submit the form and make the application go onto the next screen, if the <input> element of the type submit is in focus[h].
  - reset the form variables and make the application switch onto the next screen, if the <input> element of the type reset is in focus.
  - lead the user onto the next screen, if the <textarea> element or the <input> element of the date, number, password, or text type is in focus and there are no hyperlinks on the current screen. If there is at least one hyperlink on the screen, pressing **OK** is ignored and no action is performed.

Note: The **Reset** button (defined as <input type="reset" .../>) is guaranteed to demonstrate its normal behaviour, that is, to reset the form variables only in the case when all variables are non-volatile. If there are variables of both types – volatile and non-volatile – within the form, the **Reset** button may in some cases work as the **Submit** button. So, avoid using <input type="reset" .../> in such cases.

Handling incorrect user input

Using <baddata> element (see description on page 53), you can design a simple error message screen that will be displayed if a user enters incorrect data in the form field. The message appears for a short while; then the terminal switches back to the form. It is useful to include the error description in displayed error messages using the err.baddata_reason variable.

The screen can include any TML formatting elements and images.

Note, that every time you apply restrictions to the user input, you should provide the <baddata> element as well. Otherwise, MicroBrowser ignores the input data restrictions.

The following TML elements can include the <baddata> element: <tform>, <textarea>, and <input>.

[h] In relation to a form and the corresponding input element, the term *submit* is, to a great extent, used metaphorically. In fact, selection of the **Submit** button does not lead to sending the form to a server – as, for example, in HTML. The purpose of the button in TML is just to save new values of variables associated with <input> and <textarea> elements present within the form. To actually submit something to the server the <submit> element is used.
Also note that since new values of variables are automatically saved each time MicroBrowser switches from one screen onto another, the presence of the **Submit** button within a form is not necessary at all.

Error handling

In addition to the `err.baddata_reason` (see "Handling incorrect user input" on page 30), the following predefined TML variables are intended for implementing various error handling mechanisms in your applications:

- `err.code.high`
- `err.code.low`
- `err.description`

The first two variables are used to store numeric error codes: the `err.code.high` contains the code of the last error; in the `err.code.low` the code of the previous (the next to last) error is stored. Error codes are listed and described in Appendix A on page 142.

The `err.description` contains a brief textual description of the last error.

The following provides the information on where the values of these variables come from.

Incendo Online MicroBrowser has a multilevel hierarchical modular structure with the main module at the top and the modules performing more specific functions closer to the bottom.

Some of the functions requested by a user from the MicroBrowser (via a TML application) are performed by the main module itself. Others are passed by the main module to a lower level module for processing.

The lower level module in its turn either generates a result or passes the request to a module of the underlying level and so on.

Thus, the request may travel from level to level starting from the main module (the top of the hierarchy) until it reaches the module responsible for delivering the required functionality. When at last the result is generated, it is returned to the main module up along the processing chain.

Now, if at a certain level an error occurs this is what happens:

1. The module in which an error occurred informs the higher-level module (one that has called the lower level module) about the error and writes the textual error description into the variable `err.description`.
2. The higher-level module, in turn, returns the error code to the model which is one level up in the hierarchy (the caller) and so on.
3. When the error is returned to the main module of Incendo Online MicroBrowser, it places the code stored in the `err.code.high` into the `err.code.low` and then stores the code of the error that just occurred in the `err.code.high`.

This process generates several error messages with the same time stamp for each error, one for each module layer. These messages are stored in the System Log of the terminal. Together, these error messages are known as the *error stack*.

Note: see *Configuration and Initialization of Incendo Online-enabled Terminals* for information on how to access System Log of the terminal.

Appendix A on page 142 contains the complete listing of the error code numbers, generated by the Incendo Online MicroBrowser.

Managing time information: standard local time, daylight saving time and GMT

In your applications, you may need to deal with the following aspects of timekeeping:

- **Terminal local time,** that is, the time at the location where the terminal is installed and its possible change during daylight saving time period[i]. Payment transactions data in most cases should be accompanied with a certain, reasonable, timestamp: a card holder, definitely, expects to see the local time printed on a receipt.
- **Time synchronisation** between the terminals, Incendo Online Gateway and the application server(s) that may be installed at the locations where the local time is different.
  Some objects/data in the system have limited period of validity (e.g. terminal initialisation credentials, authentication certificates, etc.) and the terminals, Incendo Online Gateway and the application servers should be able to 'speak the common language' when dealing with various kinds of expirations.
  The base for this common language could be the Greenwich Mean Time (GMT) or Coordinated Universal Time (UTC)[j].

By default, the date variables are assumed to be in local time. However, you can use the `GMT` formatter with `setvar` or `getvar` elements, to set or display the date variables in the GMT time:

```
<setvar name="date1" lo="2009/01/01 00:00:00 GMT"
format="YYYY/MM/DD 00:00:00 GMT" />
```

There are two predefined TML variables intended for working with time information: `terminal.datetime` and `oebr.time_zone`.

The first of these variables (`terminal.datetime`) stores the current *local* terminal date and time. This variable should only be set once, when the terminal is installed at the point of sale where it's going to be used.

The variable `oebr.time_zone` is intended for storing the offset in hours of the *standard local time* from GMT[k] (this is what is called a time zone). Though the time zone itself, in fact, does not change during the year, the variable can be used to adjust the difference between GMT (which also never changes) and the local time during the daylight saving time period.

In the current implementation of Embedded Terminal Application (see "Embedded terminal software" on page 39) there is a screen where the terminal user can set the value of this variable. If properly set, you can use the value of this variable to calculate GMT.

If the terminal is installed at the location where the daylight saving (summer) time is not used, values of the two variables can be set once and forever. In this case the value of the `terminal.datetime` variable will always correspond to the local time and the following relationship will always hold for GMT:

`GMT = terminal.datetime – oebr.time_zone`.

At the locations where the daylight saving (summer) time is used the following situations are possible (during the summer time period) depending on whether neither, one of or both variables are reset:

- Neither of the variables is reset:
  local time is to be calculated as `terminal.datatime + 1 hour`, GMT is to be calculated as `terminal.datetime – oebr.time_zone`

---

[i] In most countries there is a convention to adjust the clocks one hour forward sometime near the start of spring and adjust the clocks backward in autumn. Thus, it is necessary to distinguish between the *standard local time* and *daylight saving time* (or *summer time*).
On the other hand, there are countries where the daylight saving time is not used, for example, Hong Cong, China, most countries in Africa, and so on.

[j] Here GMT and UTC are used as synonyms though, strictly speaking, there is a slight difference between GMT and UTC.

[k] This may be an integer number in the range from –12 to +13.

- The variable `terminal.datetime` is properly adjusted to the summer time, while the value of the `oebr.time_zone` is not changed:
  the value read from the `terminal.datetime` corresponds to the local time; GMT is to be calculated as `terminal.datetime – oebr.time_zone – 1 hour`
- The value of the `terminal.datetime` is not changed, the value of the `oebr.time_zone` is incremented by one:
  the local time is to be calculated as `terminal.datetime + 1 hour`, GMT is to be calculated as `terminal.datetime – oebr.time_zone + 1 hour`
- The variable `terminal.datetime` is properly adjusted to the summer time, the value of the `oebr.time_zone` is incremented by one:
  the value read from the `terminal.datetime` corresponds to the local time; GMT is to be calculated as `terminal.datetime – oebr.time_zone`

You should somehow handle all those and similar[l] situations in your application – either requiring a terminal user to perform appropriate actions or not.

Processing GMA events

Using TML, you can subscribe Incendo Online MicroBrowser to receiving the information about certain events from GMA[m] and then use this information in your TML application to make the user's interaction with the terminal more efficient and comfortable.

Events that you are capable of processing in your TML application are related to:

- pressing of certain keys on the terminal keypad
- swiping of a magnetic card though the card reader
- insertion of a smart card into the terminal

The following predefined TML variables are intended for working with 'GMA events:'

- `gma.event.subscribed`
- `gma.event.occured`
- `gma.event.key.pressed`

gma.event.subscribed

This variable specifies which of the events 'reported' by GMA you want to process in your TML application.

The value of this variable can be an empty string (`""`) or a list composed of the `"anykey"`, `"mag"`, and `"icc"`. In the latter case the items in the list are separated with a semicolon (`;`), for example, `"mag;icc"`.

`"anykey"` corresponds to pressing of certain keypad keys (normally – all keys with the exception of alphanumeric keys and the **menu** key); `"mag"` corresponds to swiping of a magnetic card through the card reader; `"icc"` corresponds to insertion of a smart card into the terminal.

It's worth mentioning that any of the events whose names are listed in the value of the variable `gma.event.subscribed` will result in starting Incendo Online MicroBrowser – if it isn't running.

---

[l] Say, when the transition back to the standard local time is performed.

[m] GMA – **G**lobal **M**aster **A**pplication – is an Idle Terminal Application acting as terminal applications dispatcher.
GMA listens for the terminal events (e.g. keypad key presses, etc.) and once an event occurs passes the information about the event to other terminal applications, for example, Incendo Online MicroBrowser.
GMA normally starts when the terminal is powered up or rebooted.

In that case as usual, the MicroBrowser – right after its start – will switch to the first of the screens within the TML page whose URL is defined by the current setting for the variable `oebr.start_page`. (For more information on this variable, see its description in "Incendo Online MicroBrowser-related variables" on page 125.)

`gma.event.subscribed=""` corresponds to the case when you don't want Incendo Online MicroBrowser to be informed of any GMA events and, consequently, you are not going to use the information about the 'GMA events' in your application.

gma.event.occured

This variable stores the name of the event that took place. It can have one of the following values: `"menu"`, `"anykey"`, `"mag"`, or `"icc"` where the `"menu"` corresponds to the **menu** key press and the rest of the names have the same meaning as in the case of the variable `gma.event.subscribed`.

As a by the way note, the set of possible values for this variable is, effectively, limited by the list of event names currently stored in the variable `gma.event.subscribed`. This, of course, doesn't refer to the event with the name `"menu"`.

gma.event.key.pressed

The value of this variable tells which of the keypad keys was pressed (if Incendo Online MicroBrowser is subscribed to key presses and one of the corresponding keys in fact was just pressed).

The variable can take one of the values from the following set: `"enter"`, `"stop"`, `"00"`, `"cancel"`, `"sys"`, `"lfeed"`, `"f1"`, `"f2"`, `"f3"`, `"f4"`, `"f5"`, `"f6"`, `"f7"`, `"f8"`, and `"f9"`. All these values correspond to the keypad key names adopted in TML.

Declaring GMA event variables

The variables related to processing of the GMA events are declared in the Embedded Terminal Application (see "Embedded terminal software" on page 39) in the following way:

```
<vardcl name="gma.event.subscribed" value="" perms="rwxrw"
volatile= "no"/>
<vardcl name="gma.event.occured" value="menu" perms="rwxr-"/>
<vardcl name="gma.event.key.pressed" value="" perms="rwxr-"/>
```

As you can see, the variable `gma.event.occured` is initially set to `"menu"`; an empty string is assigned to two other variables.

Example of GMA events being processed in a TML application

To illustrate how GMA events can be processed in a TML application, let us consider one of the possible implementations of the starting TML page:

Figure 4 - 6: GMA event processing using TML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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">
    <next uri="#initialscr1">
      <variant uri="/magcard/magcard.tml"
lo="tmlvar:gma.event.occured" ro="mag" op="equal"/>
      <variant uri="/iccemv/iccemv.tml"
lo="tmlvar:gma.event.occured" ro="icc" op="equal"/>
    </next>
  </screen>
```

```
  <screen id="initialscr1" cancel="#initialscr1">
    <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="#gma_subscr">Subscribe to GMA events</a><br/>
      <a href="#gma_unsubscr">Unsubscribe to GMA
events</a><br/>
      <a href="emb://embedded.tml">Terminal Config</a><br/>
      <a href="exit">Exit OEBR</a>
    </display>
  </screen>

  <screen id="gma_subscr" next="#initialscr1">
    <setvar name="gma.event.subscribed" lo="anykey;mag;icc" />
  </screen>

  <screen id="gma_unsubscr" next="#initialscr1">
    <setvar name="gma.event.subscribed" lo="" />
  </screen>
</tml>
```

Initially, when GMA is active (and Incendo Online MicroBrowser is not running yet), the keypad key presses (the `"anykey"` event), swiping of a magnetic card (the `"mag"` event) and insertion of a smart card (the `"icc"` event) won't lead to the start of Incendo Online MicroBrowser (`gma.event.subscribed=""`).

Then, if the terminal user starts the MicroBrowser in 'traditional' way (by selecting the corresponding function in GMA menu, see, for example, "Starting Incendo Online MicroBrowser" in *Technical Note N1, "Incendo Online terminal software installation"*), the MicroBrowser switches to the first screen (`initialscr`) within the starting page; see the code example.

Since the `gma.event.occured` at that time is equal to `"menu"` (this value is specified for the variable in its declaration) and, consequently, is not equal to `"mag"` or `"icc"`, the logical conditions present in both `<variant>` elements evaluate as false and the URL of the next screen is defined by the `<next>` element (`<next uri="#initialscr1">`). So Incendo Online MicroBrowser displays the screen initialscr1 containing the heading **OE Examples** and seven hyperlinks.

If the user selects the link **Subscribe to GMA events**, Incendo Online MicroBrowser switches to the screen `gma_subscr` where the value of the variable `gma.event.subscribed` is set to `"anykey;mag;icc"`. As a result – from that moment on – Incendo Online MicroBrowser starts to accept the information about the key presses, swiping of a magnetic card and insertion of a smart card coming from GMA.

After processing the screen `gma_subscr`, Incendo Online MicroBrowser returns onto the screen `initialscr1` (via the screen `initialscr`: the value of `gma.event.occured` is still the `"menu"`).

Let's now assume that the terminal user selects the link **Exit OEBR**. Incendo Online MicroBrowser is stopped and the control of the terminal is passed to GMA.

Now if the user swipes the magnetic card, what happens is:

1. Initiated by the event, Incendo Online MicroBrowser starts.
2. The value of the variable `gma.event.occured` is set to `"mag"`.
3. Incendo Online MicroBrowser switches to the first screen within the starting TML page (`initialscr`).

4. Since the logical condition in the first of the `<variant>` elements now evaluates as true, Incendo Online MicroBrowser goes to the page whose URL is specified within this `<variant>` element (`/magcard/magcard.tml`).

Image libraries: increasing you application's performance

To speed up downloading of your application by the terminals, you can put all (or some) of the image files referenced in your application into an image library file and then use this library file instead of the image files[n].

To start using an image library file in your application – instead of 'ordinary' image files, you should:

1. Create an image library file containing the required image files (see "Creating an image library" below).
2. Deploy the library file on a web server (see "Deploying an image library on a web server" on page 37).
3. Register the image library file name extension on the web server (see "Registering the file name extension on a web server" on page 38).
4. In the code of your TML application, modify the references to the image files which have now become part of the image library (see "Modifying references to image files" on page 38).

After completing this procedure you can remove the initial image files from the server: your TML application does not contain references to them any more.

Note: Whenever you need to make a change to the image library, for example, to add a new image file to it or replace an image file with another one, etc., you'll have to generate a new image library using the full set of image files that you need. After that you should replace the old image library file with the new one on the server.

Depending on the circumstances, you may also need to update the references to image files in the TML code.

Creating an image library

To create an image library file, use the command line utility `imagelib.cmd`[o] provided as part of Incendo Online Gateway software. This utility is located in the subfolder bin of Incendo Online Gateway folder.

The command-line syntax for running the utility is:

```
imagelib.cmd [output file] [source]
```

where the `[output file]` and `[source]` are the parameters specifying the resulting image library file and the source image files to be placed into the library respectively.

The `[output file]` is a required parameter specifying the location, name and the file name extension of the image library to be created, for example, `C:\imagelibs\my_lib.iml`. You can specify just the file name and extension (e.g. `my_lib.iml`). In that case the image library file with the specified name and extension will be created in the current directory.

You can use any name and extension; the only requirement is that you register the extension you've chosen on the web server (see "Registering the file name extension on a web server" on page 38).

[n] Downloading one image library file is much faster than downloading numerous image files.

[o] Linux users should use the file `imagelib.sh` located in the same directory as the file `imagelib.cmd` directory.

The [source] is an optional parameter. If you omit it, all image files (that is, ones having the following extensions: .bmp, .gif, .jpeg, .jpg, .tiff, .tif) located in the current directory will be added to the image library.

In place of the [source] you can use:

- An absolute or a relative path to a directory where the image files are located, for example, C:\images, images, and so on.
  In this case all the image files located in the specified directory will be placed into the image library.
- A blank-space-separated list of paths to the image files, for example:
  ok.gif yes.gif cancel.gif images\logo.bmp
  in this case the listed files will be placed into the library.
- File selection pattern containing wild cards such as ? and *, for example, images\???.gif.
  In this case all the files whose paths satisfy the specified selection condition will be placed into the library. (If the pattern images\???.gif were used, all .gif files with the names containing three characters and located in the subdirectory images of the current directory would be added to the image library.)

This is an example of how to create an image library in the easiest way:

1. Copy all the image files referenced in your TML application to the folder where the imagelib.cmd utility is located (that is, the bin subfolder of the Incendo Online Gateway folder).
2. Start Windows command interpreter **cmd.exe**:
   2.1 Click **Start** at the bottom left corner of the screen, and then click **Run**.
   The **Run** window is displayed.
   2.2 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]: e.g. C:) is not the one on which the file imagelib.cmd 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 make the directory where the file imagelib.cmd is located the current directory, type cd [path to Incendo Online Gateway]\bin (for example, cd C:\oegw\bin), and then press **Enter**.
   The path to the directory you have just specifies is shown on the current line.
5. To create an image library file in the current directory, type
   imagelib.cmd [output_file_name].[extension]
   for example, imagelib.cmd my_lib.iml
   The imagelib.cmd utility starts creating the image library. If all is well, the last two messages output by the utility into the **cmd.exe** window will look something like this:

```
20/08/07 16:28:57 INFO [main] [ImgLib Generator] 25 file(s)
processed.
20/08/07 16:28:57 INFO [main] [ImgLib Generator] done.
```

Note: You are advised that you run the imagelib.cmd from its 'native' folder, that is, the subfolder bin of the Incendo Online Gateway folder. You may experience problems if you copy the imagelib.cmd to a different directory and run it from there.

Deploying an image library on a web server

To deploy an image library on a web server, simply copy the image library file to corresponding application folder located on the server.

If you are using the `images` folder to store the image files used in your application, probably the best idea is to copy the image library file to that folder.

Registering the file name extension on a web server

In this section instructions are provided for registering the image library file name extension on Apache Tomcat web server. For a web server other than Apache Tomcat the corresponding procedure is similar. However if you are using a different server, consult the server documentation to accomplish the task.

To register the image library file name extension on Apache Tomcat web server:

1. Find the Apache Tomcat's configuration file `web.xml`. This file is located in `[path to Tomcat]\webapps\ROOT\WEB-INF` folder (for example, `C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\ROOT\WEB-INF`).
2. Open the file in a text editor.
3. Insert the following lines at the end of the file, before the closing `</web-app>` tag:

```
<mime-mapping>
<extension>iml</extension>
<mime-type>image/iml</mime-type>
</mime-mapping>
```

Note: It is assumed here that the file name extension you are using for your image library files is the `.iml`. If this is not the case, you should specify the actual file name extension in place of the `iml` in the piece of code given above.

4. Save and close the file.
5. Restart Apache Tomcat.

Modifying references to image files

Image files contained in an image library are referenced very similarly to how 'ordinary' image files are referenced: the `src` attribute of the `<img>` element (see "<img>" on page 69) is used to specify the image file's URL.

The main difference is that you should now specify the URL of the image library file and then after the hash sign (#) the name and extension of the image file contained in the image library.

Assuming that you placed the image library file `my_lib.iml` to the directory where the 'ordinary' image files were located (say, the `images` folder on your web server), the reference to the image file `<img src="images/ok.gif" alt=""/>` should be replaced with `<img src="images/my_lib.iml#ok.gif" alt=""/>`.

After updating all the references in the TML code, you may want to remove the initial ('ordinary') image files from the server since now they are not referenced in your application.

Controlling multilingual input support

By setting the value of the predefined TML variable `oebr.languages` you can define the 'default' language for the terminal and specify the list of supported languages (alphabets).

The value of this variable can be a list composed of the `"english"`, `"french"`, and `"spanish"` where the list items are separated with a semicolon (`;`), for example, `"english;french"`. There can be only one element in this list, e.g. `"spanish"`.

The first item in the list defines the 'default' language or alphabet used when inputting the characters (into input fields) by means of the terminal keypad keys. It also defines the default virtual keyboard layout – one used when the data input screen is activated – for terminal models equipped with a touch screen.

The order of the items defines that in which the supported languages are switched on the virtual (touch screen) keyboard. The set of the items defines the set of supported languages (alphabets) on the virtual keyboard.

The default variable's value (one specified in the variable declaration) is `"english;french;spanish"`.

For more information on supported languages (alphabets) and the virtual keyboard layouts, see *Basics of Working with Incendo Online-enabled Terminals*.

Embedded terminal software

The Incendo Online distribution includes the default or "embedded" TML application which is used for

- declaration of the predefined TML variables
- interactive definition of the terminal network connection settings
- working with the system log, and
- initialisation of the terminal.

To reference this application, the URI `emb://embedded.tml` is used.

Generally, you cannot modify the contents of the embedded TML application. However you can customise the terminal's behaviour by making changes to the file `customer.tml` which also comes with Incendo Online (in the binary format). For more information, refer to *Configuration and Initialisation of Incendo Online-enabled Terminals*.

The embedded TML application references the embedded stylesheet (the `embedded.css` file) which controls the appearance of TML screens is cases when an external stylesheet is not defined.

TML and payment cards

5

This chapter introduces TML concepts related to card payment and working with terminal peripherals. It includes overview of transaction flows for the card types supported by Incendo Online and instructs the TML application developers on how to add card payment possibilities to their services.

Payment cards

Incendo Online currently supports two basic types of payment cards: the magnetic stripe cards and ICC EMV (**I**ntegrated **C**ircuit **C**ards).

Magnetic stripe cards provide rather limited capabilities for payment. They are able to store only simple data about the cardholder and card itself, required for card identification, and support a limited number of payment operations. ICC EMVs represent a new generation of payment cards, where the card is indeed a small embedded computer with multiple service applications on board. ICC EMVs offer enhanced payment possibilities and diversity of services being significantly more secure.

Accessing card data

Ingenico terminals are equipped with *card readers* – the hardware devices that are capable of read/write data access for a certain card type.

Card readers are controlled by *card parsers*, pluggable software components provided with Incendo Online. These modules are run within the terminal and can be controlled via terminal configuration. Card parsers allow MicroBrowser to interact with terminal card readers.

Each parser supports the card-specific *transaction flow*. Transaction flow defines how the parties involved into payment transaction (including the merchant, user, user's card, MicroBrowser, parser and acquirer's host) should communicate to each other in order to complete the transaction. The most important steps of a transaction are card and cardholder identification, risk management and action analysis.

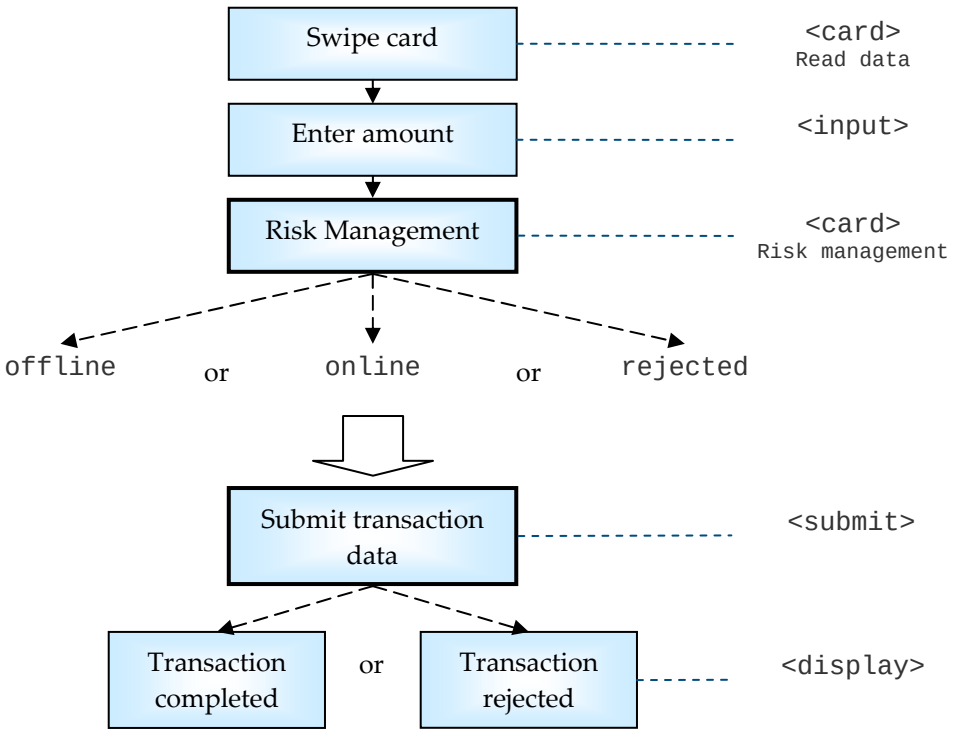
Transaction flow for magnetic stripe cards is standardised by ISO, for smart cards – by EMV. When designing your application you should strictly follow the algorithms suggested for card processing. These algorithms are described in details in the following sections.

Transaction flow for magnetic stripe cards

A typical magnetic stripe transaction includes two sequential steps (see "Figure 5 - 1" on page 42):

1. **Reading card data**
   The magnetic card parser reads data from the card and assigns the read data to the variables. Then, the user is asked for the transaction amount required for risk management, see "read_data" on page 56.
2. **Risk management**
   MicroBrowser analyses the card data, and solely decides (based on the current terminal risk management policy) whether to process the transaction offline or connect to the acquirer's host for online authorisation or reject the transaction. The policy defines a threshold or *floor limit* which represents a maximum amount of money which can be processed by the terminal offline, without connection to the acquirer's host, see "risk_mgmt" on page 57. The policy is set by the card issuer.

Figure 5 - 1: Magnetic Card Transation Flow



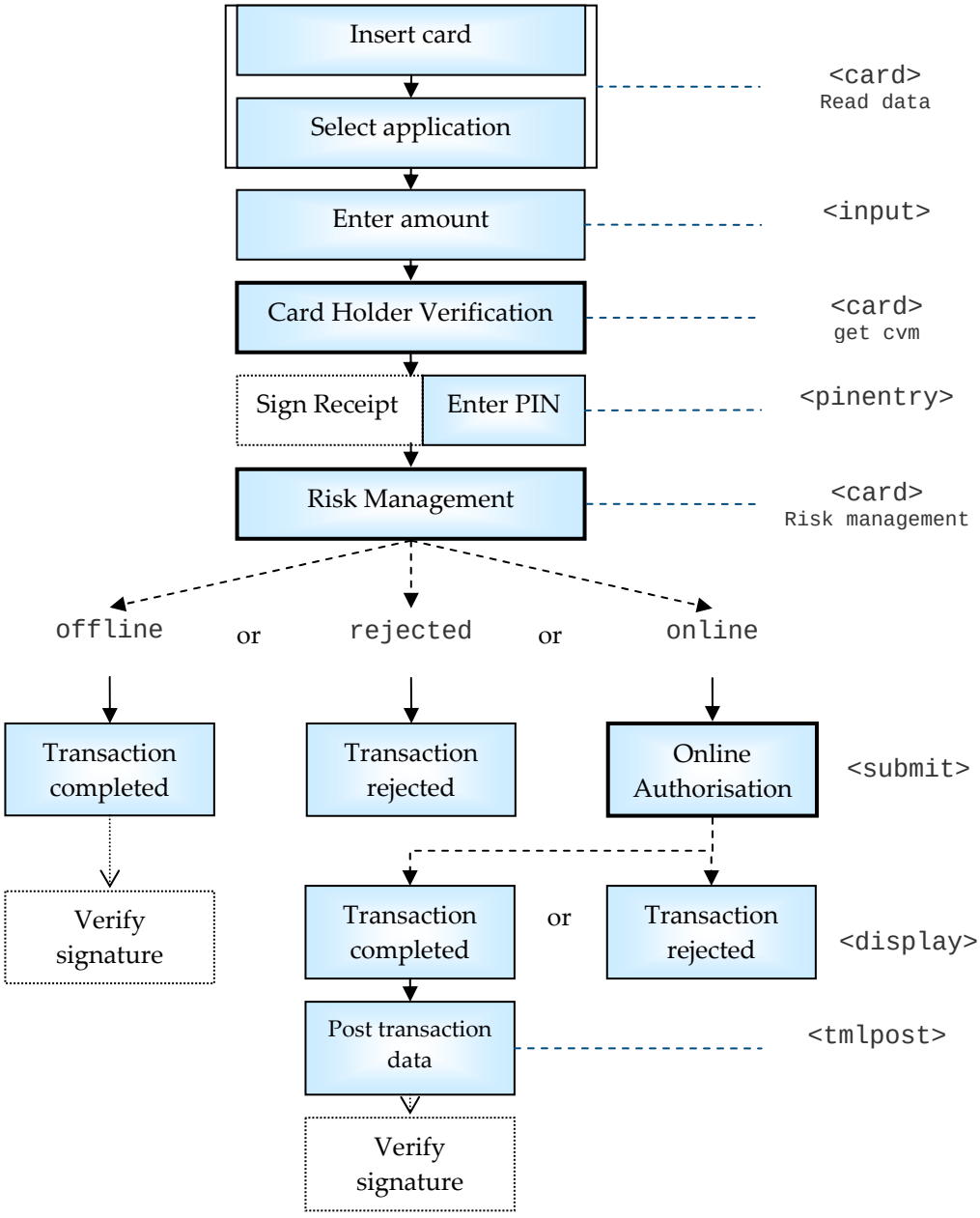
Transaction flow for ICC EMV

For smart cards the transaction flow is more complicated and includes the following steps (see Figure 5 - 2 on page 43):

1. **Card initialisation**
   ICC EMV card parser automatically selects the ICC application and checks whether the read data is genuine using Static Data Authentication (SDA) or Dynamic Data Authentication (DDA), see "init_app" on page 58. Note, that in Incendo Online 1.0 the user cannot select the application from the list.
2. **Cardholder verification**
   MicroBrowser determines the Cardholder Verification Method (CVM) which is chosen according to the data retrieved from the card. Incendo Online supports online and offline PIN verification methods. To enable signature-based customer verification, the variable `card.emv.signature` should be set to a non-zero value. Note, that the signature can be verified only after the transaction is completed, since you need to remove the card from the terminal in order to compare the signature on the receipt and the signature on the card. See "get_cvm" on page 58, "verify" on page 59 and "wait_remove_card" on page 61 for more details.
   Note, that the card can support several CVMs, therefore your application should always choose the most appropriate one.
3. **Risk management**
   The terminal MicroBrowser and the card mutually decide whether to process the transaction online, offline or reject it. The verdict is based on the currently applied risk management policy and terminal and card action analysis results. The analysis involves evaluating the amount of money specified in `payment.amount` and `payment.amount_other` variables, so their values should be set by your application before completing risk management.
   If online authorisation is not required, the transaction is completed at this step, see "risk_mgmt" on page 57.
4. **Online transaction authorisation**
   MicroBrowser posts the read card data to the Application Server, accepts the reply, updates the card data and re-requests the Application Server for the

verdict. If the transaction approved, MicroBrowser closes the card, see "auth" on page 60. The server can reject the transaction and generate a dynamic screen with an error message. If the connection to the server is broken, MicroBrowser will request risk management again to ask whether the transaction can be completed offline.

Figure 5 - 2: Smart Card (ICC) Transaction Flow



Interacting with card parsers

Card parsers provide TML developers with a convenient interface which allows to complete a transaction in several simple steps. MicroBrowser interacts with the parser using `<card>`. Offline PINs are accepted using `<pinentry>` and verified by ICC using `auth` parser command.

You can instruct your application to wait for either a magnetic stripe card swiping or for an ICC insertion by listing initial calls for both `mag` and `icc_emv` parsers one-by-one:

Figure 5 - 3: Interacting with the card parsers

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

To determine which exactly card is processed, check the value of the variable `card.parser.type`.

Every transaction step corresponds to a specific operation with the card conforming to the transaction flow defined for a specific card type, e.g. reading card data, specifying the amount of money to be paid, completing risk management checks and so on, see "Accessing card data" on page 41.

A parser assigns the data read from the card to the predefined parser-specific variables listed in the section "Variables used by card parsers" on page 120. It can also change values of some other auxiliary variables related to the processed transaction. When required, the assigned variables values are sent to the Application Server for processing.

TML reference

This chapter provides a reference to all the elements used by the TML.

6

Data types inherited from XHTML

Below is the summary of TML data types applicable to the text-only content of TML elements and attributes. The data types listed are analogous (with some limitations) to those used in XHTML 4.0.

Charset

The attribute value specifies the character encoding of a linked resource. TML currently supports ISO-8859-1 (default) and UTF-8 encodings.

Name	XML Type	Usage
Charset	string	The type is used for values of the `charset` attribute of the `<link>` element.

Example:

```
<link href="tmlapp_icc.tml" charset="ISO-8859-1"/>
```

ContentType

The attribute value declares a content type (also known as media type or MIME type) of a linked or embedded resource. For TML resources, you should use `"text/tml"` MIME extension (default).

Name	XML Type	Usage
ContentType	String	The type is used for values of the `rev` attribute of the `<link>` element.

Example:

```
<link href="/magcard/magcard.tml" rev="text/tml"/>
```

Length

The attribute value defines a number of pixels or a percentage of the horizontal or vertical space. The value of `50%` means half the available space while `50` means 50 pixels.

Name	XML Type	Usage
Length	String	The type is used for values of the `height`, `width`, `cellpadding` and `cellspacing` attributes of table elements.

Example:

```
<table width="100%">...<table />
```

LinkTypes

The attribute value defines a space-separated list of link types. See the table below for the list of link types allowed in TML:

Name	XML Type	Usage
LinkTypes	NMTOKENS	The type is used for values of the `rel` and `rev` attributes of the `<link>` element. The following link types are supported: `stylesheet` – specifies an external stylesheet for the document.

Example:

```
<link href="style.css" rev="stylesheet"/>
```

MultiLength

The attribute value defines a number of pixels or a percentage of the horisontal or vertical space or a relative length expressed as `i*` where `i` is an integer. When allocating space for elements, MicroBrowser first processes pixel and percentage lengths, then divides the remaining space among all elements with a relative length. An element with a length of `3*` will be allocated with space three times bigger than an element with length `1*`. The value `*` is equivalent to `1*` and instructs MicroBrowser to "fill the remaining space."

Name	XML Type	Usage
MultiLength	string	The type is used for values of the `width` attribute of `<col>` and `<colgroup>` elements.

Example:

```
<colgroup width="0*" />
```

Number

The attribute value defines a number which contains at least one digit in the range of 0 to 9.

Name	XML Type	Usage
Number	nonNegative Integer	Attribute values of numerous elements which represent integer numbers greater than or equal to zero.

Example:

```
<input alt="Amount:" type="number" name="payment.amount"
size="10" format="^*0.00"/>
```

Pixels

The attribute value defines a number of pixels.

Name	XML Type	Usage
Pixels	nonNegative Integer	The type is used for values of the `border` attribute of the `<table>` element.

Example:

```
<table border="3">...<table/>
```

Text

The attribute value defines a human-readable string.

Name	XML Type	Usage
Text	string	The type is used for values of the `alt` attribute of `<img>`, `<input>` and `<textarea>` elements.

Example:

```
<input alt="Expiry Date:" type="date" name="card.expiry_date"/>
```

URI

The attribute value defines a Uniform Resource Identifier (URI).

Name	XML Type	Usage
URI	anyURI	The type is used for values of the `href` attribute of `<a>`, `<base>` and `<link>` elements and `src` attribute of the `<img>` element.

Example:

```
<a href="/magcard/magcard.tml">MAGCARD</a>
```

TML-specific data types

Below is the summary of data types specific to TML. The data types define the contents of various TML attributes and elements.

Formatter

Formatter is a data type defining patterns for formatting and de-formatting of various values. For general information, see "Formatting and de-formatting" on page 24.

Name	XML Type	Usage
Formatter	string	The type is used for values of the `format` attribute of the `<vardcl>`, `<setvar>`, `<getvar>`, `<input>`, `<textarea>`, `<strtemplate>`, and `<variant>` elements. It is also used as a value of the `ro` attribute in `<setvar>` element if the element's `op` attribute is set to `"format"` (`op="format"`).

Example:

```
<getvar name="terminal.datetime" format="DD/MM/YYYY hh:mm:ss"/>
```

Formatter patterns for each variable type are described further in the section.

The patterns are case-sensitive; for example, `"M"` is different from `"m"`. If the custom pattern contains white-space characters or characters enclosed in single quotation marks, the output string will also contain those characters. Characters that are not part of a format pattern or not format characters are reproduced "as is".

The backslash (`\`) is used as an escape character in the patterns. It suppresses (escapes) the special meaning of the immediately following character. For example, to "escape" the asterisk (`*`) in cases when it can have special meaning, you can use the sequence `\*`.

Numbers

Formatter patterns for numbers can contain any characters. The following characters when used in the pattern have special meanings:

Symbol	Description
`^`	If appears as the first symbol in the pattern, specifies that the pattern should be filled *from right to left* – when the terminal user inputs the data. In any other position within the pattern – to be displayed literally – this character should be escaped with the backslash: `\^`.
`0`	A decimal digit. If the value is such that the corresponding position in the output "is empty," this position is filled with zero.
`*`	An arbitrary number of digits. This character is not allowed in the first position. Also, it can not be used more than once within a pattern as a character with a special meaning. For example, the pattern `0**` is illegal, while the pattern `0*\*` is a valid one.
`-`	In the first position or in the second position when immediately following the `^` means that the number can be negative. Displayed literally in front of the value if the number is negative and is omitted for positive numbers. To be displayed literally in positions other than the first one, this character should be escaped with the backslash: `\-`.

The pattern used for numbers by default (in the absence of the `format` attribute) is: `-0*`

The table below shows some number formatting examples.

Formatter	Value	Transforms to
-0*	-12	-12
0*	-12	12
00:00	12	12:00
^*0.00	12	0.12
^GBP: *0.00	12	GBP: 0.12
^GBP: *0.00	12345	GBP: 123.45
0.0**	1	1.0*
0	123	1
0^hi!*0	1	1^hi!0
0^hi!*0	123	1^hi!23

Strings

Formatter patterns for strings can contain any characters. The following characters when used in the pattern have special meanings:

Symbol	Description
c	Any character that should be reproduced "as is".
n	Any digit. The entered digit is displayed "as is".
c#	Hidden character – one that should be reproduced as an asterisk (*).
n#	Hidden digit – one that should be reproduced as an asterisk (*).
number	If appears after c, n, c#, or n#, indicates the number of characters or digits, for example, c4 is the same as cccc.
*	If appears after c, n, c#, or n# indicates the arbitrary number of characters or digits. This character is not allowed in the first position. Also, it can not be used more than once within the pattern as a character with a special meaning. For example, the pattern c** is illegal, while the pattern c** is a valid one.

The pattern used for strings by default (in the absence of the format attribute) is c*.

For patterns containing c, n, c#, or n# followed by a number, the "empty" positions in the output are displayed as dashes (-).

The table below shows some formatting examples for strings:

Formatter	Value	Transforms to
c*	ABCD	ABCD
c#*	ABCD	****
c8	ABCD	ABCD----
c2c#2c4c*	ABCD	AB**----
^*0.00c**	ABCD	^*0.00ABCD*

Dates

Formatter patterns for dates can contain any characters. The following sequences of characters when used in the pattern have special meanings:

Sequence	Description
YYYY or yyyy	The four-digit representation of the year, for example, 1995, 2005, and so on.

Sequence	Description
YY or yy	The abbreviated, two-digit representation of the year in which two first digits are omitted. In this representation, the years from 00 to 69 are treated as belonging to the 21th century, while the ones from 70 to 99 – as belonging to the 20th century. For example, 55 and 72 would mean 2055 and 1972 respectively. If the year in this representation is less than 10 it is displayed with a leading zero.
MM	The two-digit representation of the month number: 01 for January, 02 for February, and so on. Months whose numbers are less than 10, have a leading zero in this representation.
DD or dd	The two-digit representation of the number of the day within a month. Numbers less than 10 are displayed with a leading zero.
hh	In the absence of am/pm, the hour in 24-hour clock. Single-digit hours are displayed with a leading zero.
mm	The minute. Single-digit minutes are displayed with a leading zero.
ss	The second. Single-digit seconds are displayed with a leading zero.
am/pm	Specifies that 12-hour clock should be used. This sequence can not precede the hh and can not be used in the absence of hh. Depending on an hour, either the am or pm will be shown in place of am/pm.
GMT	Indicates that the date belongs to the GMT timezone. If this sequence is not present, it is assumed that the variable is set to the local (terminal) timezone.

Note: In date patterns each of sequences listed above can be used only once. The following characters must be escaped by means of the backslash (\) for "as-is" reproduction: y, Y, m, M, d, D, h, H, s, S.

The pattern used for dates by default (in the absence of the format attribute) is YYYY/MM/DD.

To fill the 'empty' positions in the pattern the data from the 'default' date and time 1970/01/01 00:00:00 are used.

When GMT formatter is used to set a date variable, the letters GMT must also be present in the appropriate place in the lo attribute value of the setvar:

```
<setvar name="date1" lo="01/01/2009 00:00:00 GMT"
format="DD/MM/YYYY hh:mm:ss GMT" />
```

The table below shows some formatting examples for dates:

Formatter	Value	Transforms to
DD-MM-YY	2005/12/31	31-12-2005
DD-MM-YY hh:mm:ss	2005/12/31	31-12-2005 00:00:00
DD-MM-YYYY	2005/12/31 23:59:59	31-12-2005
hh:mm:ss	2005/12/31 23:59:59	23:59:59
hh:mm am/pm	2005/12/31 23:59:59	11:59 pm
To\da\y i\s DD/MM	2005/12/31 23:59:59	Today is 31/12

Opaque

While internally opaque variables contain binary values, to set or display opaque variables you have to use one of two possible formats: `hex` or `base64`.

Formatter	Description
`base64`	This is the default format. Base 64 format uses 64 characters to encode data, consisting of upper and lower-case Roman aphabet (`A-Z`, `a-z`), the numerals (`0-9`), and the symbols `+` and `/`. Symbol `=` is used as a special suffix.
`hex`	The variable is presented as a sequence of hexadecimal characters (`0-9`, `A-F`). Each two characters represent a byte and are separated by a space. For example: `A0 FF 11 EC DA 13` etc.

The pattern used for opaque variables by default is `base64`.

Inline, Block, NoForm and Flow

Define layout formatting elements. The `Inline` type represents a "text-level" information while `Block`, `NoForm`, `Flow` – "screen-level". Note, that text in TML should be enclosed in `<p>` or one of the other screen-level or inline elements (such as headers, lists, tables, etc.)

Name	Type	Usage
Inline	<a>, , <getvar>, <input>, <textarea>	Defines text layout.

Block	, <dl>, <h1>, <h2>, <h3>,
	<h4>, <h5>, <h6>,
	<hr>, <img>, , <p>, <pre>,
	<table>, , <form>
		Defines screen layout. The elements support only `id` and `class` attributes for CSS styling.

NoForm	, <dl>, <h1>, <h2>, <h3>,
	<h4>, <h5>, <h6>,
	<hr>, <img>, , <p>, <pre>,
	<table>,
	<baddata>, <prompt>
		Includes Block type without the `<form>` element and adds error handling elements.

Flow	<a>
	
	, , <getvar>
	<input>, <textarea>
	 , <dl>, <h1>, <h2>, <h3>,
	<h4>, <h5>, <h6>,
	<hr>, <img>, , <p>, <pre>,
	<table>, , <form>
	<dd>, <display>, , <print>,
	<td>, <th>
		Includes Block and Inline types, as well as screen, table and list elements.

InputType

The attribute value defines a type of the input expected from the user.

Name	XML Type	Usage
InputType	token	The type is used for values of the attribute `type` (`<input>` element). The values define the type of the input expected from the user. The following tokens are allowed: `"number"` – a number is expected, `"date"` – a date, `"text"` – any text, `"password"` – a password (input is masked), `"checkbox"` – a user should fill a checkbox, `"radio"` – a radio button, `"submit"` – Submit button, `"reset"` – Reset button. By default, a text input is expected.

Example:

```
<input alt="APN:" name="gprs.apn" value="tmlvar:gprs.apn"/>
```

NextScreen

Defines the elements which are used to identify which screen should be processed next.

Name	XML Type	Usage
NextScreen	`<econn>`, `<error>`, `<next>`	Defines the URI of the next screen. The URI can be specified either explicitly as a constant reference, or as a variable reference, or as a selection of one or more conditional choices specified using child `<variant>` elements. The variant conditions are processed in the order they are listed. If none of the conditions matches, the URI specified in the required `uri` attribute of the element is used.

Example:

```
<next uri="#mag_submit">
  <variant lo="tmlvar:card.parser.verdict" op="equal"
  ro="reject" uri="#reject_trans"/>
</next>
```

Permissions

The attribute value defines a pattern for applying read and write access permissions to a TML variable or a TML log. The pattern controls the use of the variable or the log throughout its entire scope, see "Scopes of variables" on page 22 for more details.

Name	XML Type	Usage
Permissions	string	The type is used for values of the perms attribute of the `<vardcl>` and `<logdcl>` elements. The value is a five-character string in the `rwxrw` format, where each character specifies a certain access right option. Any symbols, except `"-"` identify that the option is set. First two options define the access rights to the variable or the log from pages located in the same directory (the root-of-the-scope directory) as the one on which the variable or the log is declared. The first option grants **read** access permission, the second – **write** access permission. The third option (if set) does not allow to redefine the variable or the log in the narrower scope. The last two options define access rights to the variable or the log from pages located in subdirectories of the root-of-the-scope directory. The fourth option grants **read** access permission, the fifth – **write** access permission. Default pattern is `"rw-rw"` which grants "full access" to the variable or the log.

Example:

```
<vardcl name="payment.amount" type="integer" perms="rwxrw"/>
```

TRules

The attribute value defines a type of ruling applied to the table.

Name	XML Type	Usage
TRules	token	The type is used for values of the attribute `rules` (`<table>` element). The values define the table column and body row ruling. The following tokens are allowed: `"none"` – no rules appear between table cells, `"rows"` – the rules appear between rows, `"cols"` – between columns, `"all"` – between columns and rows. By default, if the border attribute is not specified or equal to zero, `"none"` is assumed, otherwise – `"all"`.

Example:

```
<table border="2" rules="rows">
...
<table />
```

Valref

The attribute value represents a string that may contain variable references.

Name	XML Type	Usage
Valref	string	The type is used for values of attributes which can contain variable or constant references. To reference a variable value the prefix `"tmlvar:"` should be specified before the variable name, see the example below.

Example:

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

TML elements reference

This section lists TML elements alphabetically. If you know the name of an element and want a complete description of it, look it up in this section.

<a>

Description

This element defines a source anchor for a hypertext link which points to a TML screen or page. The target is specified in `href` attribute. The value of this attribute can be represented by a constant or a variable reference, for example, `href="#connect"`, `href="tmlvar:oebr.start_page"`.

The anchors cannot be nested.

An anchor can contain an `<img>` element.

Attributes

Name	Type	Description
`href`	URI	Specifies a destination URI for the link. The value of the attribute can be represented by a constant or a variable reference. It can also be one of special URIs - `back`, `menu` or `cancel` (see on page 16).
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<p> <dt> <pre> <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <prompt>, <baddata>, <form>, <print>, <display>, <td>, <th>, <dd>, ,	<img>

Remarks

Note that the URI cannot reference graphic files. See "URIs and TML" on page 14 for general information about URI formats supported by TML.

For `href` attribute, instead of URI, you can also specify predefined string values, such as `back` and `exit`. The first switches to the previously processed screen or to the root screen if the previous screen cannot be defined. The second – commands the terminal to exit MicroBrowser. See "Special URI values" on page 16 for more information

Example

```
<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>
```

<baddata>

Description

Using `<baddata>` element, you can design a message that will be displayed if a user enters incorrect data in the form field.

The message appears for the specified time or until the user presses a button, then the terminal switches back to the form.

Attributes

Name	Type	Description
`timeout`	Number	Specifies the amount of time the message is displayed on the screen. If `timeout` is `0`, the user should press a key to return to the parent screen. Default value is `2`.
`max`	Number	Specifies the maximum amount of attempts the user is allowed for entering the data. If the user has exhausted the maximum amount of attempts but still failed to provide the correct data, MicroBrowser switches to the screen specified in the `next` attribute. Default value is `0`.
`next`	ValRef	Specifies an URI of the screen to which MicroBrowser switches if the user failed to enter the valid data. Default value is `tmlvar:oebr.prev_screen`. It can also be one of special URIs - `back`, `menu` or `cancel` (see on page 16).
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<input> <textarea> <tform>	<a> <dl> <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, <getvar>, , <p>, <pre>, , <table>,

Example

See the example for `<tform>` element on page 104.

<base>

Description

This element defines the base URI for all relative links on your page. The element must be placed between the opening and closing tags of the `<head>` element.

Attributes

Name	Type	Description
`href`	URI	Specifies a base URI for the page.

Related elements

Parent elements	Child element
<head>	none

Remarks

If `<base>` is omitted, it is considered that the base URI is the URI of the current page.

The `<link>` element(s) following the `<base>` tag are relative to the defined base URI.

The base can be defined as an absolute or as a relative URL, see "URIs and TML" on page 14.

Description

This tag is used to create a line break.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<a>, <baddata>, <display>, , <dd>, <dt>, <form>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, , <p>, <pre>, <print>, <prompt>, , <td>, <th>	none

<call_func>

Description

This element calls the specified terminal operating system C function. This is required, for example, for printing or cancelling offline posts, performing the terminal initialisation procedure, clearing a log, etc.

When calling a C function, you should specify two URIs: one for the screen to switch to if the function call was a success and the other – for the screen to go to if the function call failed.

The first of the URIs is specified by means of the `next` attribute (see "<screen>" on page 85) or the `<next>` element (see "<next>" on page 81), while the second – by means of the `<error>` element (see "<error>" on page 66).

If the function call fails, a brief description of the error that occurred is 'placed' into the predefined TML variable `err.description` (see "Error handling" on page 31 and "Error handling variables" on page 123).

Attributes

Name	Type	Description
`name`	string	Required attribute specifying the name of the function. See "Remarks" below for the list of the predefined function names.

Related elements

Parent elements	Child element
<screen>	none

Remarks

Some of the functions use predefined TML variables as input parameters though these variables are not explicitly referenced within the <call_func> element. Values of such variables should be properly set prior to calling the corresponding function (see the following table).

Please also note that the functions working with offline posts, cached TML pages and logs are applied only to the resources/data with the appropriate scopes; see "URIs and data scopes" on page 15. Scopes of cached TML pages are defined similarly to those of the offline posts.

Function name	Description
`cancel_offline_post`	Cancels an offline post. The ID of the post to be cancelled is defined by the value of the predefined variable `oebr.post_id`. Its value should be properly set prior tocalling this function.
`clear_http_cache`	Deletes cached TML pages from the terminal cache. The function does not affect cached offline transactions.
`clear_log`	Clears a log, that is, deletes all log records. The name of the log to be cleared is defined by the value of the predefined variable `oebr.log_id`. The value of this variable should be properly set prior to performing the function call.
`connect_to_server`	Forces the terminal to connect to Incendo Online Gateway and, depending on the current values of the corresponding TML variables: • submit to the Application Server all offline posts – if `oebr.connect.pool_off="yes"` • request the updated versions of all TML resources cached in the terminal memory (TML pages, images, etc.) – if `oebr.connect.sync_cache="yes"` • request the updates of configuration data for ICC EMV and magnetic card parsers – if `oebr.connect.sync_config="yes"` For more information, see descriptions of these variables in "Incendo Online MicroBrowser-related variables" on page 125.
`net_initialisation`	Activates terminal initialisation, which assumes requesting the binary terminal password from the Incendo Online Gateway.

Function name	Description
print_offline_posts	Prints information about all postponed HTTP POST requests using the embedded printer. When printed, each request is accompanied with a local unique identifier which is assigned by the terminal.
release_transport	Instructs the terminal to disconnect from the server.

Example

```
...
<logdcl name="my-log" ..
...
<screen id="clear-my-log" next="#call_ok">
  <setvar name="oebr.log_id" lo="my-log"/>
  <error uri="#call_error"/>
  <call_func name="clear_log"/>
</screen>
```

<card>

Description

The element allows TML application to interact and control terminal card parsers. The detailed information on card parsers is provided in "Payment cards" on page 41.

You should specify (using the parser attribute) which card parser the terminal should use in order to accept the user card data.

You define the type of a card operation using the parser_params attribute. It instructs the parser to perform a particular action or sequence of actions with the card.

The card parser assigns the read data to the card-related predefined variables, see "Variables used by card parsers" on page 120. The assigned values can be submitted to the Application Server for processing. You can also specify parser-specific commands using the parser_params attribute.

Attributes

Name	Type	Description
parser	string	Specifies the card parser. Possible values are "mag" and "icc_emv"; see "Accessing card data" on page 41 for more information about the parsers.
parser_params	string	Specifies the command passed to the card parser. The available commands are parser-specific: "mag": read_data, risk_mgmt "icc_emv": init_app, get_cvm, verify, risk_mgmt, auth, wait_remove_card These commands are discussed in "Remarks" below.

Related elements

Parent elements	Child element
<tform>	none

Remarks

Below is the description of available parser commands.

read_data

This function reads data from a magnetic stripe card. This is the initial step of a magnetic stripe card transaction. You can use this command on the initial screen of your application, which prompts the user to swipe a card.

```
...
<tform>
  <baddata class="c_center">
    <getvar name="err.baddata_reason"/>
  </baddata>

  <card parser="mag" parser_params="read_data"/>
  <prompt class="c_center">
    Please, Swipe Card
  </prompt>
</tform>
...
```

The command instructs the `mag` card parser to perform the following sequence of actions:

1. Wait until the card is swiped through the card reader.
2. Read ISO data tracks from the card.
3. Verify the card number and expiration date, and if any of these is considered to be invalid, specify the reason using the `err.baddata_reason` variable. See "Error handling variables" on page 123.
4. Parse the read data and assign it to the predefined TML variables, see "Card-related variables" on page 120 and "Variables used by "mag" parser" on page 121.

Note: Before issuing the command the variable `card.input_type` should be set to `1`, see the description of this variable in "Card-related variables" on page 120.

When the command is completed, your application should analyse the `mag` parser-specific variables and ask the user to enter the amount of money, which is usually a subject of a transaction.

The user input should be assigned to the predefined variables `payment.amount` and `payment.amount_other`. The entered amount is required for the next stage of the transaction processing – risk management, see "risk_mgmt" below.

Your application can also provide the parser with a recommendation on transaction processing mode by assigning the `offline` or `online` values to the variable `card.parser.verdict`.

risk_mgmt

This function instructs the `mag` parser to perform risk management – the next step of the transaction processing.

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

Risk management is a policy defined by the card issuer, which specifies the probability of carrying out the transaction online (rather than offline).

The parser analyses the following:

- transaction amount and type
- the card scheme and issuer
- the terminal internal statistics (that is, the current number of postponed transactions)
- application-proposed transaction processing mode specified in the `card.parser.verdict` variable

The parser decides whether to process the transaction online, offline or reject it by assigning values to the variables `card.parser.verdict` (`online` or `offline`) and `card.parser.reject_reason` respectively. The transaction can be rejected if MicroBrowser has failed to find risk management data for a particular card issuer or if the manually entered card details are incorrect.

Note: for online transactions, if connection to Incendo Online Gateway is lost, risk management should be repeated again. The parser should analyse the value of `err.baddata_reason` variable and, depending on the current risk management policy, can either recommend to use the offline mode of transaction processing, or reject the transaction.

init_app

Initiates communication with an ICC card. This is an initial step of the ICC EMV transaction processing:

```
...
<tform>
  <baddata>
    <getvar name="err.baddata_reason"/>
  </baddata>

  <card parser="icc_emv" parser_params="init_app"/>
  <prompt>
    Please, Insert Card
  </prompt>
</tform>
...
```

The command instructs the `icc_emv` parser to perform the following sequence of actions:

1. Wait until the card is inserted into the card reader.
2. Select an appropriate ICC application.
3. Read and authenticate the application data, see "Transaction flow for ICC EMV" on page 42.

When processing this command, the `icc_emv` parser fills the following TML variables:

- the ones listed in "Card-related variables" on page 120 with the exception of `card.issue_number` and `card.issuer_name`
- the ones listed in "Variables used by "icc_emv" parser" on page 120 with the exception of `card.emv.aac`, `card.emv.arqc`, `card.emv.atc`, `card.emv.cvmr`, `card.emv.last_attempt`, `card.emv.signature`, `card.emv.tc`, `card.emv.tvr`, `card.emv.unumber`
- `card.parser.type`, see "Card parsers-related variables" on page 121

Note: The `icc_emv` parser interacts with the card at all stages of transaction processing and, thus, the card should remain in the card reader. If the card is removed from the reader before the transaction is completed, the parser interrupts execution of the current command and sets the values of `card.parser.verdict` and `card.parser.reject_reason` to `"reject"` and `"The ICC is removed"` respectively.

get_cvm

This function retrieves the card verification method (CVM) which should be used for the transaction. Usually, a card contains a prioritised list of available card holder verification methods.

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

Note: The command can be used only after the user of the terminal, following the logic of your application, has selected the transaction type and entered the amount(s) of money. The amount(s) entered by the user should be assigned to the predefined variables `payment.amount` and `payment.amount_other`.

The `get_cvm` command allows to retrieve the available CVMs one by one.

When the command is executed for the first time, the parser sets the value of the `card.parser.cvm` variable to the first item retrieved from the list of CVMs. Each next execution of the command sets the value of this variable to the next CVM available in list. As a result, the variable `card.parser.cvm` can have one of the following values:

Value	Description
`"pin_online"`	PIN should be verified online.
`"pin"`	PIN should be verified offline.
`"no_cv"`	No card holder verification required.
`""` (empty string)	No more CVMs are available on the card.

If the card suggests the offline PIN verification method, your TML application should ask the user to enter the PIN.

To capture the PIN, use the `<pinentry>` element. You can specify that the PIN should be verified by ICC EMV using the `icc` attribute of `<pinentry>` element. Then, to verify the entered PIN, use the verify parser command (see the next section).

There may be cases when a PIN pad is not present or the user refuses to enter the PIN. In such cases your TML application should set the value of the `card.parser.cvr` variable (CVR – **C**ardholder **V**erification **R**esult) to `"no_pinpad"` or `"no_pin"` respectively.

If the CVM being used implies verification of the cardholder's signature, MicroBrowser will assign the non-zero value to the variable `card.emv.signature`, and after the receipt is printed and signed by the customer, you TML application should ask the user (the merchant) to verify the signature.

verify

This function verifies the entered PIN offline.

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

The current number of PIN entry attempts is stored in `card.emv.last_attempt` variable.

As a result of the command execution, the parser assigns one of the following values to the predefined variable `card.parser.cvr` (CVR – **C**ardholder **V**erification **R**esult):

Value	Description
`"ok"`	PIN is valid, card holder verification was successful.
`"pin_tries"`	The user has exceeded the maximum number of attempts allowed to enter the PIN. The application should try another CVM.
`"failed"`	The entered PIN is not valid. The user should try to enter the PIN one more time.

risk_mgmt

This function requests the parser to perform risk management and action analysis for the current ICC transaction.

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

The parser interacts with the card analysing the current terminal risk management policy, amount of money, card history and terminal and card action analysis results.

As a result, the parser suggests a verdict for the transaction which defines the mode of transaction authorisation (online or offline). The verdict is stored in the predefined variable `card.parser.verdict`; the possible values are `"online"`, `"offline"`, or `"reject"`.

Prior to performing the requested operations, the parser checks the value of the predefined variable `oebr.econn`. It is done for troubleshooting possible connection failures which might occur while performing the authorisation online. If the previous attempt to post the data to the Application Server has failed due to the connection error, the parser can either approve the offline risk management or reject the transaction (depending on the policy currently set for the terminal).

Typically, the output variable values are posted to the Application Server for analysis. The host part of the application processes the posted values and decides which of them should be sent to the acquirer and which not.

If online authorisation is required, the host part replies with a dynamically generated TML page which contains another parser command `auth`. Otherwise, the transaction is considered completed at this step.

auth

This function requests the parser to perform the transaction authorisation online – according to the results of preceding `risk_mgmt` command. The `auth` command usually appears on a dynamic TML screen generated by the host part of your application.

```
<tml xmlns="http://www.ingenico.co.uk/tml" cache="deny">
<!-- Example of a dynamic TML page -->
  <screen id="auth_ok" next="/iccemv/iccemv.tml#subm_tc_aac">
<!-- authorisation approved -->
    <setvar name="payment.txn_result" lo="1"/>
      <tform>
        <card parser="icc_emv" parser_params="auth"/>
      </tform>
  </screen>
</tml>

<!-- Screen within the static page that follows the dynamic
page -->
<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="oebr.transid"/>
  </submit>
</screen>
```

Online authorisation follows the algorithm described below:

1. In response to the submitted ICC variables (after `risk_mgmt` command) the host part of the application detects that the transaction goes online and responds with a dynamic TML page. Usually, the page (see the example above) represents a TML form, which calls the parser with the `auth` command and assigns values to the following variables: `payment.auth_code`, `payment.emv.issuer_auth`, `payment.emv.issuer_script1`, `payment.emv.issuer_script2`.
2. The MicroBrowser executes the page. The received data is written to the card, the card executes issuer's scripts and the parser requests the card to calculate Application Authentication Cryptogram (AAC) and Transaction Certificate (TC) for the updated card data.

3. The parser assigns results of the operation to either `card.emv.arqc` (online), `card.emv.aac` (rejected) or `card.emv.tc` (offline) variables. The status of the issuer's scripts execution is put into the variable `payment.emv.issuer_script_results`. It is recommended to submit all these variables to the server for processing.
4. The parser closes the card. Note, that for offline transactions the card is closed by `risk_mgmt` command.

wait_remove_card

This function instructs your application to wait until the user removes the card from the card reader. This command is usually accompanied with a short prompt for the user:

```
<tform>
  <card parser="icc_emv" parser_params="wait_remove_card"/>
    <prompt>
      Please, Remove Card
    </prompt>
</tform>
```

<col>

Description

Assigns attribute values to the individual columns within `<colgroup>`. You should not use this element if you have already specified `span` attribute for `<colgroup>` element.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`span`	Length	Defines how many columns are affected by the `<col>` element. If omitted, the element spans a single column.
`width`	Length	Specifies the width for each spanned column.
`align`	token	Sets the horisontal alignment of the cell contents. The possible values are `left`, `center` and `right`.
`valign`	token	Set the vertical alignment of the cell contents. The possible values are `bottom`, `middle` and `top`.

Related elements

Parent elements	Child element
<colgroup>, <table>	none

Example

```
<table width="90%" border="1" cellspacing="5">
  <colgroup>
    <col width="30%" />
    <col width="40%" />
  </colgroup>
  <col width="60%" />
    <tr>
      <th>First Column Header</th>
      <th>Second Column Header</th>
      <th>Third Column Header</th>
    </tr>
    <tr>
      <td>First Column First Row</td>
      <td>Second Column First Row</td>
      <td>Third Column First Row</td>
    </tr>
```

```
        <tr>
          <td>First Column Second Row</td>
          <td>Second Column Second Row</td>
          <td>Third Column Second Row</td>
        </tr>
    </table>
```

<colgroup>

Description

This element allows you to create a column-centric table as compared to the standard row-centric XHTML table. You can organise semantically related columns in one or more columns groups.

If the span attribute is not specified for a group, you can assign attributes to the individual columns using the `<col>` element.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`span`	Length	Sets the number of columns that are associated with each column group. If the columns are unequal, it is recommended to use the `<col>` element to create each column instead. If omitted, the element spans a single column.
`width`	Length	Specifies the width for each spanned column.
`align`	token	Sets the horisontal alignment of the cell contents. The possible values are `left`, `center` and `right`.
`valign`	token	Set the vertical alignment of the cell contents. The possible values are `bottom`, `middle` and `top`.

Related elements

Parent elements	Child element
<colgroup>, <table>	<col>

Example

See "Example" for the `<col>` element on page 61.

<dd>

Description

This element represents the definition in definition lists. You can type the text directly between the opening and closing tags.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<dl>	<a>, , , <getvar>, <input>, <textarea>, , <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <form>

Example

See the example for the `<dl>` element on page 64.

<defaults>

Description

Defines default values for menu and cancel attributes of `<screen>` elements within the page. The attributes are used for setting URIs of the screens to switch when the user presses the Menu or Stop button on the terminal.

Attributes

Name	Type	Description
`menu`	Valref	Specifies the URI to switch to if a user presses the **Menu** button on the terminal. The value should be a constant or variable reference.
`cancel`	Valref	Specifies the URI to switch to if a user presses the **Cancel** button on the terminal. The value should be a constant or variable reference.

Related elements

Parent elements	Child element
<head>	none

Remarks

You can always override these settings for a particular screen by defining `menu` and `cancel` attributes of the corresponding `<screen>` element.

If you omit the element or specify empty values for the element attributes, pressing **Menu** and **Cancel** buttons on the terminal will have no effect.

Example

See the example for `<head>` element on page 68.

<display>

Description

This element defines the limits of the display screen. You can type the text you want to appear on the screen directly between the opening and closing tags. The content placed between the element tags is rendered on the terminal display.

If the displayed content does not include links, it will appear on the screen until the user presses a button or the system timeout occurs.

Attributes

The element has no attributes.

Related elements

The element is flow-based.

Parent elements	Child element
<screen>	<a>, , , <dl>, <form>, <getvar>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, <input>, <log>, , <p>, <pre>, , <table>, <textarea>,

Example

```
<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>
```


Description

This element defines a section or division of a document that requires a special formatting style. The block of text is delimited by line breaks (analogous to `<br>`). You can type the text directly between the opening and closing tags.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

The element is flow-based.

Parent elements	Child element
<dd>, <display>, , <print>, <td>, <th>	<a>, , , <dl>, <form>, <getvar>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, <input>, , <p>, <pre>, , <table>, <textarea>,

Remarks

You can apply core attributes to the element contents. For example, you can use the `class` core attribute to apply the effects of CSS to the text block or you could assign an `id` to each block of code to reference it with a hyperlink.

If you want to apply attributes to an inline portion of a page, use `<span>` instead.

Example

```
<div class="c_left">
  <p><a href="#main">configure</a></p>
  <p><a href="/">ignore</a></p>
  <p><a href="exit">exit browser</a></p>
</div>
```

<dl>

Description

This element denotes a definition list. In a definition list, an introduced term or phrase is followed by a definition or explanation. The element is only used with the `<dt>` element, which defines the term, and the `<dd>` element, which describes the definition.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<baddata>, <form>, <prompt>, <dd>, <display>, , <print>, <td>, <th>	<dt> + <dd> +

Remarks

You can also use the `<ol>` element to create an ordered list and the `<ul>` element to create an unordered list.

Example

```
<dl>
  <dt>Term</dt>
  <dd>Definition</dd>
  <dt>Another term</dt>
  <dd>Another definition.</dd>
</dl>
```

<dt>

Description

This element represents the term in definition lists. You can type the text directly between the opening and closing tags.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<dl>	<a>, , , <dl>, <form>, <getvar>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, <input>, , <p>, <pre>, , <table>, <textarea>,

Example

See the example for the `<dl>` element above.

<econn>

Description

This element is a child of the `<submit>` element. Specifies the URI of the screen to switch to if the connection to the Application Server is lost. The element is of type InputType, so the URI can be specified as a constant or variable reference or can be selected from a number of choices using child `<variant>` elements.

Attributes

Name	Type	Description
`uri`	string	The mandatory attribute which specifies the URI of the error screen. It can be a constant or a variable reference. It can also be one of special URIs - `back`, `menu` or `cancel` (see on page 16).

Related elements

Parent elements	Child element
<submit>	<variant> *

Example

See the example for the <submit> element on page 98.

<error>

Description

A child element of the <head> (see "<head>" on page 68) and the <screen> (see "<screen>" on page 85) elements which specifies the URI of the screen to switch to if an error occurs.

When used within the <head>, the <error> element defines the URI of the 'default error screen' for a page: it is activated if an error occurs when processing a screen containing no <error> element.

The URI is defined by the element's uri attribute whose value may be represented by either a constant or a variable reference. URI may also be selected from a number of choices defined by the child <variant> elements (see "<variant>" on page 111).

Attributes

Name	Type	Description
uri	string	Required attribute specifying the URI of an 'error screen'.

Related elements

Parent elements	Child element
<head>, <screen>	<variant> *

Example

See the example for the <call_func> element on page 56.

<form>

Description

This element defines a layout of the TML form. You can type the text you want to appear on the screen directly between the opening and closing tags.

Note: The contents of the <form> element cannot be printed using <print> element.

Attributes

The element has no attributes.

Parent elements	Child element
<display>, <print>, <form>, <td>, <th>, <dd>, ,	, <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <input>, <textarea>

Example

```
<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>
```

<getvar>

Description

This element returns the value of the variable specified by the `name` attribute.

Attributes

Name	Type	Description
`name`	string	The required attribute that specifies the name of the referenced variable.
`format`	Formatter / Valref	Defines a pattern for the variable value. Can be a constant or a reference to a string-type variable that contains a valid formatter pattern. The valid pattern depends on the type of variable that is being set (see "Formatter" on page 47). If the `format` attribute is not used, or it is set to empty (""), the default formatter for the variable is used

Related elements

Parent elements	Child element
<layout>, <strtemplate>, <submit>, etc.	none

Example

```
<getvar name="payment.amount" format="^*0.00"/>
```

It is possible to use a variable reference for the value of the `format` attribute. It must be of the `string` type and has to contain a valid formatter pattern:

```
<screen id="formtr_test">
  <!-- string variable -->
  <setvar name="amount_formatter" lo="^EUR: *0.00" />
  <!-- integer variable -->
  <setvar name="amount" lo="12345" />

  <display>
    <getvar name="amount" format="tmlvar:amount_formatter"/>
  </display>

</screen>
```

This screen will show `EUR: 123.45` on the terminal display.

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

Description

TML supports six levels of headings from `<h1>` (the most important) to `<h6>` (the least important).

The heading elements define text headers for a document.

MicroBrowser is able to display the header text in one of six different sizes. `<h1>` usually is the largest size, `<h6>` – the smallest one. The heading styling is controlled by CSS.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<display>, <print>, <form>, <td>, <th>, <dd>, , , <prompt>, <baddata>	<a>, , <getvar>, <input>, , <textarea>

Remarks

Only inline elements can be used within the heading elements.

Example

```
<h1>OE Examples:</h1>
```

<head>

Description

A child element of the <tml> element (see "<tml>" on page 106) representing a TML page header and, as a rule, containing page-wide settings and defaults specified by the means of its child elements.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.

Related elements

Parent elements	Child element
<tml>	<base> ? <link> * <defaults> ?, <error> ?

Example

```
<head>
  <link href="emb://customer.tml" rev="text/tml"/>
  <link href="embedded.css" rev="stylesheet"/>
  <defaults menu="#main" cancel="#main"/>
</head>
```

<hr>

Description

This tag is used to render a horizontal rule (line). The element takes a single line. The appearance of the rule is controlled by the CSS.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<form>, <dd>, <display>, , <print>, <td>, <th>, <prompt>, <baddata>	none

Example

```
<hr/>
```


Description

This element is used to insert an image directly into the text flow. No additional line breaks or carriage returns are automatically inserted before or after the image. The image positioning can be controlled by means of CSS.

The image file URI is specified by means of the `src` attribute whose value can be represented by a constant or a variable reference, for example, `src="images/btmlpa.gif"`, `src="images/my_lib.iml#btmlpa.gif"`, `src="tmlvar:print_icon"`, and so on.

The element can be placed inside an `<a>` tag to provide a "clickable" image for a hyperlink.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`src`	URI	Required attribute specifying the URI of the image file. The value of the attribute can be represented by a constant or a variable reference. Image files contained in image libraries (see "Image libraries: increasing you application's performance" on page 36) are referenced like this: `[url of the image library file]#[image file name and extension]`, for example `images/my_lib.iml#btmlpa.gif` See also, "Modifying references to image files" on page 38.
`alt`	Text	Required attribute specifying a text message that will be displayed (in place of the image) if MicroBrowser cannot display a graphic image or picture.
`height`	Length	An attribute setting the height of the image in pixels or relative to the width of the screen.
`width`	Length	An attribute setting the width of the image in pixels or relative to the width of the screen.

Related elements

Parent elements	Child element
<a>, <baddata>, <form>, <prompt>, <dd>, <display>, , <print>, <td>, <th>	none

Example

```
<a href="#mag_rm">
  <img src="images/ok_up.gif" alt="ok_up"/>
</a>
```

<input>

Description

This tag is used within the `<form>` element to collect information (text, password, etc.) entered by the user.

`<input>` can be used to create input elements of different types. The type of an input element is defined by the value of the `type` attribute, see Table 1: The possible values of the type attribute and their meanings on page 71.

The element has a number of validity check attributes that allow to restrict user input.

These attributes are `equal`, `not_equal`, `max` and `min`. These can be either constants or variable references.

If the validity check fails, MicroBrowser switches to the URI specified in the child `<baddata>` element. If you omit `<baddata>`, validity check is skipped.

If the input data is valid, the value entered by the user is assigned to the variable specified in the mandatory `name` attribute.

When MicroBrowser encounters `<input>` element, it switches the terminal to the form processing mode, see "Form processing" on page 30.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`alt`	Text	Provides the alternative text for the terminals which are not capable of displaying images and forms.
`name`	string	For input elements of the `checkbox`, `date`, `number`, `password`, `radio`, and `text` types: specifies the name of the variable which receives the value entered by the user. For input elements of the `submit` and `reset` types: defines the label of (the text shown on) the corresponding button. By default (in the absence of this attribute), the button for submitting the form is labelled **OK**, while the button for resetting the form – **Reset**.
`type`	InputType	The attribute defines the type of an input element that should be rendered on the terminal display. See Table 1 on page 71 for the list of possible values and their explanations.
`readonly`	token	Displays the value that cannot be changed by the user.
`size`	Length	Sets the width of a single-line input box for input elements of the `date`, `number`, `password`, and `text` types (see Table 1 on page 71). The value defines how many characters can fit in the box. In addition to that, the value of this attribute defines the maximum length (measured in the number of characters) of the input value.
`equal`	Valref	This is a validity check attribute. It can be a constant or a variable reference. The data entered by the user is checked whether it is equal to the value specified by the attribute. If not, the flow control is passed to the child `<baddata>` element.
`not_equal`	Valref	This is a validity check attribute. It can be a constant or a variable reference. The data entered by the user is checked whether it is unequal to the value specified by the attribute. If not, the flow control is passed to the child `<baddata>` element.

Name	Type	Description
max	Valref	This is a validity check attribute. It can be a constant or a variable reference. The data entered by the user is checked whether it is less than the maximum value specified by the attribute. If not, the flow control is passed to the child <baddata>. For strings, the length of the entered string is checked.
min	Valref	This is a validity check attribute. It can be a constant or a variable reference. The data entered by the user is checked whether it is more than the minimum value specified by the attribute. If not, the flow control is passed to the child <baddata>. For strings, the length of the entered string is checked.
value	Valref	For input elements of the date, number, password, and text types: defines an initial value to be displayed in the input box. The value can be a constant or a variable reference. For input elements of the checkbox and radio types: specifies the value to be assigned to the variable defined by the name attribute when the element is selected.
format	Formatter	For input elements of the date, number, and text types: defines a formatting pattern for the input value.

Table 1: The possible values of the type attribute and their meanings

Value	Description
checkbox	A check box, a switching element representing an option which can be either selected or not. As any other input element, a check box has a TML variable associated with it, which is specified by the name attribute. The initial state of a check box on the screen is defined by the current value of the variable associated with the check box and the value of the value attribute. If those two are the same, the check box is initially selected; if otherwise, the check box is not selected. If a user changes the state of a check box, the value of the variable associated with it is updated; if otherwise – remains unchanged. If a user selects a check box, the variable is assigned the value defined by the value attribute. If a user deselects a checkbox, the variable is assigned an empty string ("") as its value.
date	a field (box) for entering dates. Even when not explicitly set, a date-type variable is set to the value of the Epoch (1970/01/01 00:00:00) by default, so some information will always be present in the input box. However, if you set all fields of the Date variable to 0 explicitly <setvar name="date1" lo="0000/00/00 00:00:00" format="YYYY/MM/DD 00:00:00" /> the variable can be used as an uninitialized (empty) variable for input fields. The 'date1' variable does not contain any values, so <getvar name="date1" format="YYYY/MM/DD 00:00:00"/> will return an empty string.

Value	Description
`number`	a field (box) for entering non-negative integer numbers. Note: The number of digits in the input number can not exceed 8. Thus, the maximum number that can be entered is `99999999`.
`password`	a field (box) for entering text where the text is masked
`radio`	a radio button – an element representing one of the possible options. A radio button usually belongs to a group in which there are at least two radio buttons. Only one radio button within the group can be selected. (The situation when none of the radio buttons in the group is selected or when there is only one radio button in the group is treated as normal.) All radio buttons in the group have the same value of the `name` attribute – the name of a TML variable associated with the group. When one of the radio buttons is selected, this variable is assigned a value defined by the `value` attribute of the radio button that has been selected. The initial state of a radio button on the screen is defined by the current value of the variable associated with the group and the value of the value attribute. If those two are the same, the radio button is initially selected; if otherwise, the radio button is not selected.
`reset`	a button for resetting the form variables, that is, for bringing them to the last saved state. Use this element *only* in the cases when *all variables* within the form are *non-volatile*. *Avoid* using the element if some or all variables within the form are *volatile*: in such cases the "resetting capabilities" of the corresponding button on the screen are *not guaranteed*.
`submit`	a button for "submitting the form," that is, for *saving* new values of the variables used within the form – if those were changed by the user. The use of this element within a form is *not necessary* since new values of the variables are saved *automatically* each time the application switches from one screen onto another. **Note:** To actually submit something to a server, the `<submit>` element is used.
`text`	a field (box) for entering text
`list`	See "<input type="list">" on page 73

Remarks

Any number of `<input>` elements can be placed anywhere between a pair of opening and closing `<form>` tags to create the desired appearance of the form.

Related elements

Parent elements	Child element
<form>	<baddata> ?

See also "<textarea>" on page 102.

Example

```
...
<vardcl name="oebr.run_on_reboot" value="yes" perms="rwx--"
volatile="no"/>
...
<screen id="oebr_prfrncs" ...>
  <display>
...
    <form>
...
```

```
        <input type="checkbox" name="oebr.run_on_reboot"
        value="yes"/>
         Run on Reboot<br/>
        <input type="submit"/>
      </form>
...
  </display>
</screen>
...
```

In the example above the value `yes` is assigned to the variable `oebr.run_on_reboot` in its declaration.

When the screen `oebr_prfrncs` is displayed for the first time, the check box is *selected* as the corresponding `<input>` element is associated with the variable `oebr.run_on_reboot` and the value of its `value` attribute (`yes`) is the same as the current value of the `oebr.run_on_reboot` variable.

If a user deselects the check box and then submits the form or switches onto a different screen, the empty string is assigned to the variable `oebr.run_on_reboot`.

<input type="list">

Description

The element is used within the `<form>` element (see "<form>" on page 66) to create a list box associated with a TML variable.

The name of the variable is defined by the element's `name` attribute.

The set of the list items – options a user can choose from is specified by the `value` attribute. The value of this attribute may be represented by a constant string where the list items are separated with semicolons (`;`). It may also be represented by a variable reference (`tmlvar:[variable name]`). In the latter case the semicolons within the variable value are treated as list items separators.

There may be list boxes where only one or a number of items can be selected. The multiple attribute that may be set either to `"yes"` or `"no"` defines whether or not multiple selections are possible. By default, the value `"no"` is assumed meaning that only one of the list items can be selected.

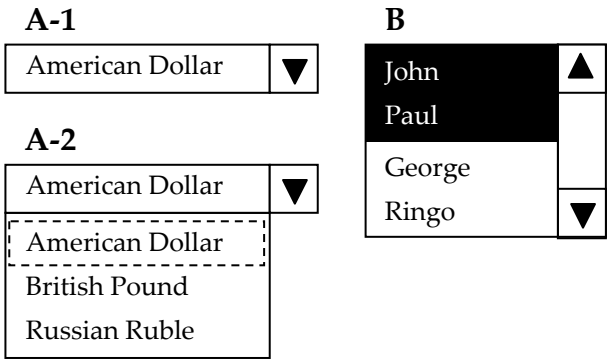
Once selection of an item is performed, the item is assigned to the variable associated with the element. When a number of items is selected, separate items in the variable's value are separated with semicolons.

You can use the element's `format` attribute to define the formatting pattern which should be applied to a string formed according to the current selection within the list box prior to assignment of a value to the variable (see "Formatter" on page 47).

Initially selected in the list (when the corresponding screen has just been shown and a user has not yet started to select/deselect the items) are the items whose textual representations are present in the value of the variable associated with the list box. For example, if the `my-var` variable is associated with a list box and its current value is `item-2`, for the list box with the items `item-1`, `item-2` and `item-3` (`<input type="list" name="my-var" value="item-1;item-2;item-3" .. />`), the item `item-2` will be initially selected.

The appearance of the list box is defined by the element's `rows`, `width` and `height` attributes. It may also be influenced by the value of the `multiple` attribute.

Figure 6 - 1: Examples of <input type="list">



The value of the `rows` attribute defines how many rows are to be shown. (A row corresponds to a list item.)

`rows="0"` and `rows="1"` would define a *drop-down* list box (See A-1 in Figure 6 - 1 on page 73) where only one row is initially shown. A user has an ability of opening such a list box (the list in this case 'drops down', A-2 in Figure 6 - 1 on page 73) to see all the list items and perform selection of the item.

If the value of the `rows` attribute is set to a value greater than one, a simple (open) list box is shown (See B in Figure 6 - 1 on page 73); the attribute's value in this case corresponds to the number of rows displayed. If the number of rows specified by the value of this attribute is less than the number of items in the list, a user has the ability of scrolling the list up and down. If the number of rows to be shown is greater than the number of items in the list, some of the rows will be empty.

If omitted, the `rows="1"` is assumed by default which corresponds to a drop-down list box.

If the value of the `multiple` attribute is set to `"yes"`, the value of the `rows` attribute will define the number of rows shown only in the case when this number is greater than the number of list items. Otherwise, the attribute is ignored and the number of rows shown will be equal to the number of items in the list.

The `width` attribute defines the (absolute or relative) width of the list box on the screen. It may be set as a number of pixels or as percentage of the width of the parent element (say, the width of a table's cell – if the list box is placed into that cell). The value of the attribute is ignored if the width specified by this attribute is less than that necessary to fully display the 'longest' of the list items.

The `height` attribute defines the (absolute or relative) height of a row in the list box. It may be set as a number of pixels or percentage of the *screen* height. If the row height specified by the value of this attribute is less than that necessary to fully display a list item, the value of the attribute is ignored.

If the attribute is omitted, the row height (default row height) corresponds to the height of a list item.

If the row height set by this attribute is greater than the default row height, the number of rows in the list will correspond to the number of the list items; the value of the `rows` attribute in this case is ignored. (The `rows="0"` and `rows="1"` – in the absence of the multiple attribute – will, however, specify a drop-down list box in which the row height will be defined by the value of the `height` attribute.)

Attributes

Name	Type	Description
`class`	NMTOKENS	See "class attribute" on page 115.
`alt`	Text	Provides the alternative text for the terminals which are not capable of displaying images and forms.
`name`	string	Required attribute specifying the name of a TML variable associated with the list box.

Name	Type	Description
`value`	Valref	Required attribute specifying the list of options where the items are separated with semicolons (`;`). The list may be represented by a constant string or a variable reference (`"tmlvar:[variable name]"`). A string in which the list items selected by a user are separated with semicolons is formatted according to a pattern defined by the `format` attribute (if present) and then assigned as a value to the variable associated with the list box.
`multiple`	token	An attribute defining whether or not more than one list item can be selected. Possible values are `"yes"` (more than one item can be selected) and `"no"` (only one item can be selected). Default value is `"no"`. `multiple="yes"` would specify a simple (open) list box, even though the value of the rows attribute may specify the opposite.
`format`	Formatter	An attribute whose value defines the formatting pattern that is applied to the string 'made of' the selected list items prior to assigning a value to the variable associated with the list box. If more than one of the list items is selected, the string in which these items are separated with semicolons is formed first and then the specified formatting pattern is applied to the resulting string.
`rows`	Number	An attribute whose value defines how many list-box rows are to be shown. If the `multiple` attribute is omitted or its value is set to `"no"`, the `rows="0"` or `rows="1"` would specify a drop-down list box.
`width`	Length	An attribute specifying an absolute or relative width of the list box. The attribute can be defined either as `width="n"` or `width="n%"`, where `n` is a positive integer number. The former would set the width of a list box to n pixels, while the latter would mean that the width of the list box should be equal to n% of the parent element's width (for example, screen, table cell's width, etc.). If the attribute is omitted, the width of the list box is defined by the width of the longest list item.

Name	Type	Description
height	Length	An attribute specifying an absolute or relative height of a list-box row. (A row corresponds to a list item). The attribute can be defined either as height="n" or height="n%", where n is a positive integer number. The former would set the height of a list box to n pixels, while the latter would mean that the width of the list box should be equal to n% of the *screen* height. If the attribute is omitted, the height of a row is defined by a list item height. If the attribute specifies the row height greater than that defined by a list item height, the number of rows shown for the list box will be equal to the number of the list items (even if the rows attribute defines a greater number).

Related elements

Parent elements	Child element
<form>	<baddata> ?

Remarks

When a screen containing a drop-down list box is displayed, what is initially shown in the drop-down list box is the current value of the variable associated with the corresponding <input type="list"> element. A user, obviously, expects this value to be in some reasonable relation with the set of available list items (options). Thus, it is recommended that you first assign one of the list items as a value to a variable and only then use a list box associated with this variable in your application.

Example

```
...
<vardcl name="currency" value="American Dollar"/>
<vardcl name="group" value="John;Paul"/>
...
<screen id="set_currency">
  <display>
    <form>
      Select currency:<br/>
      <input type="list" name="currency" value="American
      Dollar;British Pound;Russian Ruble"/>
    </form>
...
  </display>
</screen>
...
<screen id="set_group">
  <display>
    <form>
      Select group members:<br/>
      <input type="list" name="group"
      value="John;Paul;George;Ringo" multiple="yes"/>
    </form>
...
  </display>
</screen>
...
```

The set_currency screen contains a drop-down list box (the multiple="no" and rows= "1" are assumed by default) in which only one of the list items can be selected at a time.

When this screen is shown for the first time, the drop-down list box will contain the text `American Dollar`, which corresponds to the current value of the `currency` variable defined in the variable declaration.

If a user does not 'touch' the list box, the value of this variable will stay unchanged. If a user opens the list box and selects a different item, the selected item will be assigned as a value to the variable `currency` when the user finishes working with the list box.

The `set_group` screen contains a simple (open) list box in which more than one item can be selected (`multiple="yes"`).

When this screen is displayed for the first time, the items `John` and `Paul` will be shown as selected. If a user selects or deselects the items in this list box, a new list containing selected items will be assigned as a value to the variable group.

<layout>

Description

Used within the `<logdcl>` element (see description on page 79) to specify a log record rendering pattern and, thus, to define how log records should be shown on the terminal display or printed.

The element may contain any inline and block elements with the exception of the `<form>` element. However in practice, the element will normally contain text fragments, `<getvar>` elements (see "<getvar>" on page 67) and the tags defining the log records' appearance (such, for example, as `<p>`, `<br>`, `<h1>` and so on).

Variables referenced by `<getvar>` elements should be a subset of the 'log variables' – the ones enumerated by means of `<vardcl>` elements within the parent `<logdcl>` element.

Attributes

Name	Type	Description
`name`	NMTOKENS	Required attribute defining the layout name.

Related elements

Parent elements	Child element
<logdcl>	`<getvar>` as well as any other inline and block elements with the exception of the `<form>` element.

Example

```
<layout name="def_layout">
  <h1>
    <getvar name="terminal.datetime" format="MM/DD hh:mm:ss"/>
     - <getvar name="oebr.log_module"/>
     - <getvar name="oebr.log_severity"/>
  </h1>
  <p><getvar name="oebr.log_descr"/></p>
  <br/>
</layout>
```


Description

This tag is used to define an item in a list. You can type the text directly between the opening and closing tags.

The element is required for both the ordered list `<ol>` and the unordered list `<ul>`. In an unordered list, each item is preceded by a star, such as `*`. In an ordered list, each item is labelled with a number that increments with each following item.

Attributes

Name	Type	Description

id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
,	<a>, , , <getvar>, <input>, <textarea>, , <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <form>

Example

See the examples for <ul> element on page 109 and <ol> element on page 82.

<link>

Description

This element is used to establish a relationship between the current page and one or more other related pages.

The element is a child of the <head> element. MicroBrowser loads and caches the linked documents and files in advance, before rendering the page.

Attributes

Name	Type	Description
charset	Charset	The charset attribute is used to specify the character encoding used on the page that is the target of the link.
href	URI	This is a required attribute which contains a valid URI address for the linked document.
type	xs:token	Specifies the type of the medium the links apply to. Permitted values include: all, print and screen.
rel	xs:token	Represents a space-separated list of one or more values that specify the relationship from the source page to the target for a link. Permitted values include: next, prev, section, stylesheet.
rev	xs:token	Represents a space-separated list of one or more values that specify the relationship from the target page to the source for a link. The attribute can also be used to specify the content type of a linked resource. Typical values are stylesheet and text/tml.

Related elements

Parent elements	Child element
<head>	none

Example

See the example for the <head> element on page 68.

<log>

Description

Used within the <display> (see page 63) or the <print> element (see page 84) to display or print a log.

The required attributes name and layout define which of the logs is to be displayed or printed and which of the layouts (rendering patterns) is to be used. Consequently, the name attribute should reference the existing log's name (i.e. correspond to the

name attribute of the corresponding <logdcl> element, see page 79) and the layout attribute – the name of one of the layouts specified for that log (i.e. corresponds to the name attribute of one of the child <layout> elements (see page 77) of the corresponding <logdcl> element).

The optional type attribute specifies the order in which the log records are to appear. type="normal" would mean that the most recent records should be shown closer to the log's end (bottom). type="reverse" would specify the reversed order. By default, type="normal" is assumed.

Attributes

Name	Type	Description
name	NMTOKENS	Required attribute specifying the name of the log that should be displayed or printed. The attribute value should correspond to the name attribute of the corresponding <logdcl> element, see page 79.
layout	string	Required attribute specifying the name of the layout (rendering pattern) for the log records. The value of this attribute should correspond to the name attribute of one of the child <layout> elements (see page 77) of the corresponding <logdcl> element.
type	token	Optional attribute defining the order in which the log records are to be shown. Possible values are: "normal" and "reverse". type="normal" means that the most recent records are shown closer to the log's end, while type="reverse" specifies the reversed order. Default value is "normal".

Related elements

Parent elements	Child element
<display>, <print>	none

Example

```
...
<logdcl name="my-log" ...
...
  <layout name=="print-layout" ..
...
</logdcl>
...
<screen id="print-my-log" ...
  <print>
    <h1>My Log</h1>
    <hr/>
    <log name="my-log" layout="print-layout"/>
  </print>
</screen>
...
```

<logdcl>

Description

This element declares a log. (For general discussion of logs in TML, see "Logs" on page 28.)

The child <vardcl> elements define the list of variables whose values are included in each separate log record (that is, written to corresponding log file). The <logdcl> element can contain any number of <vardcl> elements or may contain none.

It should be specifically noted that the use and meaning of <vardcl> elements within the <logdcl> element are different from those of ordinary <vardcl> elements (ones described in "<vardcl>" on page 109). Here, they are used *not to declare* variables but to *specify* the ones whose values should be written to the log. Consequently, the <vardcl> elements within the <logdcl> element must reference (name) the variables that have already been declared.

The child <layout> elements define various possible patterns according to which separate log records may and should be rendered, that is, shown on the terminal display or printed (see "<layout>" on page 77). There can be any number of <layout> elements within the <logdcl> element or there may be none.

If present, the <logdcl> elements should appear at the beginning of a TML page after variable declarations (the <vardcl> elements) but before the <screen> elements.

Attributes

Name	Type	Description
name	NMTOKENS	Required attribute defining the name of a log.
css	string	Optional attribute specifying the URL of an external css file which contains definitions of styles according to which the log contents are to be formatted when shown on the terminal display or printed. For more information, see "TML CSS" on page 139.
perms	Permissions	An attribute defining the log access permissions. The meaning and use of this attribute is the same as that of the perms attribute of the <vardcl> element (see "Permissions" on page 51 and "<vardcl>" on page 109). The value rw-rw is assumed by default (that is, if the attribute is omitted).

Related elements

Parent elements	Child element
<tml>	Sequence: <vardcl>* <layout>*

Remarks

Please note that the <vardcl> elements within the <logdcl> just name the variables rather than declare them. The referenced variables have to be declared earlier.

The usage pattern for the <vardcl> element in this case is <vardcl name="[variable-name]"/>, see "Example" below.

Example

```
<logdcl name="system_log" css="emb://emb_log.css"
perms="rwx--">
  <vardcl name="terminal.datetime"/>
  <vardcl name="oebr.log_module"/>
  <vardcl name="oebr.log_descr"/>
  <vardcl name="oebr.log_severity"/>
  <layout name="def_layout">
    <h1>
      <getvar name="terminal.datetime" format="MM/DD
hh:mm:ss"/>
```

```
         - <getvar name="oebr.log_module"/>
         - <getvar name="oebr.log_severity"/>
      </h1>
      <p><getvar name="oebr.log_descr"/></p>
      <br/>
    </layout>
</logdcl>
```

<logrec>

Description

Used within the <screen> element (see page 85) to append one record to a log.

The attribute name defines the log to which the record is to be appended and thus should reference the existing log's name (see "<logdcl>" on page 79).

The contents of the log record are defined by the set of <vardcl> elements within the corresponding <logdcl> element.

Attributes

Name	Type	Description
name	NMTOKENS	Required attribute specifying the name of the log to which the record is to be appended.

Related elements

Parent elements	Child element
<screen>	none

Example

```
...
<logdcl name="my-log" ...>
  <vardcl .../>
...
</logdcl>
...
<screen id="append-another-record" ...>
  <logrec name="my-log"/>
</screen>
...
```

<next>

Description

Used within the <screen> element to specify the URI of the next screen. The URI can be selected form a number of choices defined using child <variant> elements. If none of the <variant> elements matches, the URI specified by the uri attribute is used.

Attributes

Name	Type	Description
uri	Valref	Required attribute defining the URI of the next screen. The attribute value can be represented by a constant or a variable reference. It can also be one of special URIs - back, menu or cancel (see on page 16).

Related elements

Parent elements	Child element
<screen>	<variant> *

Example

See example for the <screen> element on page 87.

Description

This tag is used to define the limits of an ordered list. An ordered list is a collection of items listed in a particular order. TML supports only numbered lists starting with 1.

Use <li> to display the item content.

Note: Ordered lists cannot be nested.

You can use the <ul> to create an unordered list and the <dl> to create a definition list.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<dd>, <display>, , <print>, <td>, <th>	*

Example

```
<ol class="c_normal">
  <li class="c_normal">Normal item 1</li>
  <li class="c_normal">Normal item 2</li>
</ol>
```

<p>

Description

Used to delimit a paragraph which is preceded by line break and carriage return.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

You can use only inline elements within the <p> element.

Parent elements	Child element
<baddata>, <dd>, <display>, , <print>, <prompt>, <td>, <th>	<a>, , <getvar>, <input>, , <textarea>

<pinentry>

Description

Used within the <tform> for the card PIN. The application switches to the secure mode and uses the security features of the terminal to accept user input and encrypt it according to the specified encryption algorithm. This ensures that the pin is never stored in an unencrypted form, and therefore can not be compromised.

The encryption algorithm is set by the type attribute. Possible types are icc, dukpt and dukpt3des.

`icc` type is used for offline ICC PIN verification. This should be done after the ICC application that is used for the transaction has been selected and CVM has been requested.

The PIN is then verified using the `verify` command of the `icc_emv` card parser.

See "Transaction flow for ICC EMV" on page 42 for a description of ICC EMV transaction process. The <card> element description on page 56 explains how to interact with the card parser.

`dukpt` and `dukpt3des` type encryption is used to encrypt the PIN with DUKPT or 3DES DUKPT encryption respectively. The process for both types is the same:

1. `card.pan` variable should be filled with the card PAN number. You can do this by reading it from the card or setting the PAN number manually.
2. `card.pin.array` variable may be set to the DUKPT array you wish to use. You can leave it set to the default value of `0`.
3. use the `<pinentry>` element with the appropriate encryption type. After the PIN is entered, it will be processed by the secure hardware of the terminal. This will set the following variables:
   - `card.pin` - encrypted PIN
   - `card.pin.smid` - key serial number
   - `card.pin.length` - the number of characters in the entered PIN
4. submit the `card.pin` and `card.pin.smid` variables to the acquirer for verification

You can specify the prompt to be displayed on a PIN pad display using the `prompt` attribute.

Attributes

Name	Type	Description
`prompt`	string	Specifies a short message to render on the PIN pad screen during PIN entry.
`length`	Number	Specifies the number of characters in the PIN. The default value is `4`.
`type`	token	Specifies the encryption algorithm that will be used for pin encryption: • `icc` - the PIN is used for offline icc verification • `dukpt` - the PIN will be encrypted using DUKPT[16] scheme • `dukpt3des` - 3DES DUKPT encryption

Related elements

Parent elements	Child element
<tform>	none

Example

DUKPT encryption:

```
<screen id="pin_dukpt" next="#submt_data" >
  <tform>
    <baddata max="1" next="#main_menu">
```

[16] **Derived Unique Key Per Transaction (DUKPT)** is a key management scheme in which for every transaction, a unique key is used. This key is derived from an initial PIN encryption key, injected into the terminal. Therefore, to use DUKPT encryption, the terminal should have been injected with an appropriate key.

```
        <table border="2" class="warning">
          <tr><td>
            <getvar name="err.baddata_reason"/>
          </td></tr>
        </table>
      </baddata>
      <pinentry type="dukpt" prompt="Enter PIN"/>
    </tform>
  </screen>
```

<postvar>

Description

This tag is used within <tmlpost> element to define the name and the value of the variable to be submitted to the Application Server. This element appears only on the pages generated by MicroBrowser.

Attributes

Name	Type	Description
name	string	The mandatory attribute defines the name of the variable to be submitted.
type	token	The attribute defines the type of the variable to submit. The default type is "string".
value	string	The mandatory attribute defines the value of the variable to be submitted.

Related elements

Parent elements	Child element
<tmlpost>	none

Example

See the example for <tmlpost> element on page 107.

<pre>

Description

This element is used to display preformatted text. The text specified within the element is rendered on the output device as is, including white spaces, tabs, and line breaks.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

You can use only inline elements within the <pre> element.

Parent elements	Child element
<baddata>, <dd>, <display>, , <print>, <prompt>, <td>, <th>	<a>, , <getvar>, <input>, , <textarea>

<print>

Description

This element defines the limits of the print screen. The content which is placed between the element tags is printed on the embedded terminal printer. You can type the text you want to be printed directly between the opening and closing tags.

When printing is completed, MicroBrowser switches to the screen specified in the <next> element.

The element is a child of the <screen> element.

Attributes

The element has no attributes.

Related elements

You can use block elements within the <print> element.

Parent elements	Child element
<screen>	, <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, <log>, , <p>, <pre>, <table>, , <form>

Example

```
<screen id="posts_p_full">
  <print>
    <hr/>
    <div class="c_bold_large_center">
      <getvar name="oebr.posts_print.header_label"/>
    </div>
    <hr/>
    <log name="oebr.posts_print" layout="full_post"/>
    <br/><br/><br/>
  </print>
</screen>
```

<prompt>

Description

Used within <tform> to render a short message on the terminal screen, prompting a user to swipe, insert, or remove a card. The message disappears when the requested action is completed.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

You can use block elements, except <form> within the <prompt> element.

Parent elements	Child element
<tform>	, <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>,

Example

See the example for the <tform> element on page 104.

<screen>

Description

Defines the screen, which is a logical unit of a TML page. Each screen must have a unique name, defined by its id attribute.

Important: in TML, the length of a screen name should not exceed 12 characters

TML supports several types of screens, including display, print, submit, terminal form and function call screens; see "Screens and navigation" on page 17. The type of the screen is specified using one of the child elements (<display>, <print>, etc.).

After the opening <screen> tag, you can list the variables whose values should be set using the <setvar> elements. Note that the values of corresponding variables are (re)set every time Incendo Online MicroBrowser comes onto the screen.

Use the <next> element, or next attribute of the <screen> element to define the URI of the next screen. If both the element and the attribute are specified, the element takes precedence.

Note: If the element's content includes hyperlinks, both the <next> element and the next attribute are ignored. However, the child <next> element's <variant> elements with a key attribute can still be used and be effective.

To specify the URI of the screen to switch to if an error occurs, use the <error> element (see "<error>" on page 66).

To associate other screens' URIs with the **Menu** (**F2**) and **Cancel** terminal keys, use the menu and cancel attributes.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.
menu	Valref	Specifies the URI of the screen to switch to if the user presses the **Menu (F2)** button while browsing the screen. The URI specified in the attribute has precedence over the URI defined in the <defaults> element.
next	Valref	Specifies the URI of the next screen to switch to when MicroBrowser finishes processing the current screen. The attribute value can be a constant or a variable reference. It can also be one of special URIs - back, menu or cancel (see on page 16). If the <next> element is also specified for the screen, it will have precedence over this attribute.
timeout	Number	Specifies the maximum amount of time (in seconds) MicroBrowser should wait for the user input. When the specified time has elapsed, MicroBrowser switches to the screen specified in the child <next> element or the next attribute of the <screen> element. The attribute is ignored if the screen contains hyperlinks.

Related elements

You can use block elements, except <form> within the <prompt> element.

Parent elements	Child element
<tml>	Sequence: <setvar> * <strtemplate> * <logrec> * <next> ? <error> ? <call_func> | <display> | <print> | <submit> | <tform> ?

Example

```
<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>
```

<setvar>

Description

This tag sets a variable to a specified value. MicroBrowser processes `<setvar>` elements when it switches to the screen except the situations when it returns to the screen after rendering an error message.

`<setvar>` can only operate on the variables that have been declared previously using the `<vardcl>` element, or are pre-defined by the Incendo Online Microbrowser.

The variables also must have the correct write permissions (see "Permissions" on page 51).

In TML you can place `<setvar>` elements only within the `<screen>` element.

Instead of specifying the variable value explicitly, you can define a simple expression of two operands, the result of which will be assigned to the variable. The operands and operation are defined by the element attributes and depend on the type of the variable.

The type of the variable and its default format are defined when the variable is declared (see "<vardcl>" on page 109).

If the result of the expression and the type of the result variable do not match, before calculating the result, MicroBrowser will attempt to cast operands to the data type of the variable, see "Casting TML variables" on page 25.

Note: Some of the predefined variables are read-only and their values cannot be changed using `<setvar>` elements, see the section "Pre-defined TML Variables" on page 117.

Attributes

Name	Type	Description
`name`	string	Specifes the variable that is being set. This attribute is required.
`format`	Formatter	Defines a pattern for the variable value for explicit assignment (when `ro` and `op` attributes are not used). Also used for some operations. The content of the attribute must be a constant. For string variables, the formatter must correspond to the type of variable referenced by the `lo` attribute. For all other types, the formatter must correspond to the type of the variable that is being set (referenced by the `name` attribute). If the `format` attribute is not used, or it is set to empty (""), the default formatter for the variable type is used. See "Formatter" on page 47 for a description of valid formatter patterns.

Name	Type	Description
lo	string / Valref	The required attribute defines the left operand of the expression. If ro and op attributes are omitted, the value specified in lo attribute is assigned to the variable. The attribute value can be a constant or a variable reference.
ro	string / Valref	The attribute represents the right operand of the expression. The attribute value can be a constant or a variable reference.
op	token	The attribute defines a logical operation that is performed on the left and right operands. The possible operation depend on the type of variable being set: • **opaque** no operations possible • **string** "plus", "minus" and "format" (see page 89) • **string list** "item" and "number" (see page 91) • **integer** "plus" and "minus" (see page 92) • **date** "plus" and "minus" (see page 94) Examples below explain each operation in more detail.

Related elements

Parent elements	Child element
<screen>	none

Assigning explicit values

To assign a value to a variable, simply use the required value for the lo attribute. ro and op attributes should be empty:

```
<setvar name="string1" lo="This is a string" />
```

lo can be either a constant or a variable reference.

If the referenced variable is of a different type to the variable defined by the name parameter, it will be cast to the required type. See "Casting TML variables" on page 25 for information on type casting.

Consider the following example:

```
<setvar name="string1" lo="05" />
<setvar name="integer1" lo="tmlvar:string1" />
```

integer1 will be equal to 5.

You can also apply formatting when assigning explicit values, by setting the format attribute to a valid formatter pattern. Unlike the <getvar> element (see page 67) format attribute of the <setvar> can only be a constant.

Note: for string variables, the formatter must correspond to the type of variable referenced by the lo attribute. For all other types, the formatter must correspond to the type of the variable that is being set (referenced by the name attribute).

The formatter is applied to the lo before the variable is set. Then, the de-formatted result is converted into the internal structure for the variable type. For instance, if the date1 is a date-type variable, after

```
<setvar name=”date1” lo=”29/06/08” format=”DD/MM/YY” />
```

`date1` variable will have an internal value of `2008/06/29 00:00:00`.

Likewise, `<setvar name="int_var" lo="$1.23" format="$0.00" />` will assign `123` to the integer variable `int_var`.

If the formatter pattern does not correspond to the contents of the `lo`, it will cause an error:

```
<!-- incorrect setvar formatter -->
<setvar name="string2" lo="John Smith" format="Bad c*" />
<!-- this setvar will cause an error -->
```

See the “Formatter” on page 47 for the description of the valid formatter patterns.

It is possible to combine formatting, variable reference and type casting:

```
<setvar name="string4" lo="tmlvar:terminal.datetime"
format="To\da\y i\s DD/MM"/>
```

For the example above, the following steps will be followed:

1. variable reference is resolved for the `lo` attribute. MicroBrowser takes note of the variable type.
2. the value of the `lo` is modified according to the `format` attribute pattern. Because the variable specified by the `name` attribute is of a string type, and the variable reverenced by the `lo` is of a date type, date-type formatter pattern must be used.
3. the result is cast to the type of the variable defined by the `name` attribute, and assigned to that variable

If the current date is 29th of June, then the value of the `string4` variable will be `Today is 29/06`.

Note: be very careful when using `setvar` to cast variables and to apply formatting at the same time. It is very easy to format a variable in such a way that the cast will be illegal.

String variable operations

String variables can be assigned values directly, or the following operations can be performed: `format`, `plus` and `minus`.

Note: for string variables, the valid formatter pattern for the `format` attribute must correspond to the type of the `lo` attribute. See “Formatter” on page 47 for the description of the valid patterns.

format

This operation applies the pattern of the `format` attribute to the `lo` attribute and assignes the result to the string variable, referenced by the `name` attribute. It is equivalent to assigning explicit value to a string-type variable and using format attribute at the same time (see above).

`lo` can be either a constant (it is assumed to be of a string type), or a variable reference. This variable can be of any type.

The value of the `format` attribute must be a constant. It must be a valid formatter pattern for the variable type referenced by the `lo` (see “Formatter” on page 47 for the description of the valid patterns). If the `format` attribute does not contain a valid formatter pattern, an error will occur.

`ro` attribute is not used.

The following example demonstrates the results of the format operation:

```
<screen id="str_format" next="#string_menu">
  <setvar name="integer1" lo="-567" />
  <setvar name="opaque1" lo="AB 0F" format="hex" />
```

```
  <!-- need to escape the 'c' character in the formatter with
'\' otherwise the result will be 'WelJome ohn Smith' -->
  <setvar name="string1" lo="John Smith" op="format"
format="Wel\come c*!" />
  <setvar name="string2" lo="tmlvar:integer1" op="format"
format="^-0.00" />
  <setvar name="string3" lo="tmlvar:opaque1" op="format"
format="hex" />
  <display>
    string1: <getvar name="string1" /><br/>
    string2: <getvar name="string2" /><br />
    string3: <getvar name="string3" /><br />
  </display>
</screen>
```

The variables will contain the following values:

Variable	Value
string1	Welcome John Smith!
string2	-5.67
string3	ab 0f

plus

The `plus` operation, when performed on string variables, concatenates the value of the `ro` attribute to the contents of the `lo` attribute. The values of `lo` and `ro` can either be constants or references to string-type variables. The `format` attribute is ignored.

```
<screen id="str_plus" next="#string_menu" >
  <setvar name="string1" lo="One" op="plus" ro="Two" />
  <setvar name="string2" lo="tmlvar:string1" op="plus"
ro="Three" />
  <setvar name="string3" lo="tmlvar:string2" op="plus"
ro="tmlvar:string1" />
  <display>
    <table>
      <tr><td>
        string1: <getvar name="string1" /><br/>
        string2: <getvar name="string2" /><br />
        string3: <getvar name="string3" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
string1	OneTwo
string2	OneTwoThree
string3	OneTwoThreeOneTwo

minus

The `minus` operation, when performed on string variables, removes the number of characters corresponding to the value of the `ro` attribute from the contents of the `lo` attribute.

`lo` must be a string or a reference to a string-type variable. `ro` must be a number or a reference to an integer-type variable.

If the value of the `ro` is positive, the characters are removed from the right-hand side of the `lo` string. If `ro` is negative, the characters are removed from the left-hand side.

If the value of the `ro` is greater than the length of the `lo` string, the result is an empty string.

```
<screen id="str_minus" next="#string_menu" >
  <!-- remove 3 characters from the right-hand side -->
  <setvar name="string1" lo="OneTwo" op="minus" ro="3" />

  <!-- remove 3 characters from the right-hand side -->
  <setvar name="string2" lo="OneTwo" op="minus" ro="-3" />

  <!-- use integer variable as the 'ro' attribute -->
  <setvar name="integer1" lo="2" />
  <setvar name="string3" lo="tmlvar:string2" op="minus"
ro="tmlvar:integer1" />

  <display>
    <table>
      <tr><td>
        string1: <getvar name="string1" /><br/>
        string2: <getvar name="string2" /><br />
        string3: <getvar name="string3" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
string1	One
string2	Two
string3	T

String list operations

String lists are strings that contain items separated by the `;` character. For example, a string `One;Two;Three` is a list of three elements - `One`, `Two` and `Three`. Therefore, everything that can be done with strings can be done with string lists.

However, two additional operations can be performed, `number` and `item`.

number

The `number` operation, returns the number of items in the string list, contained in the `lo` attribute.

The variable referenced by the `name` attribute must be of the integer type. `lo` attribute can contain a constant or reference a string-type variable.

```
<setvar name=”integer1” lo=”One;Two;Three;Four” op=”number” />
```

The example above will assign `4` to the `integer1` variable.

item

The `item` operation will assign the contents of an item from a string list to the string-type variable, defined by the `name` attribute.

`lo` attribute should contain a string list. It can be a constant or a reference to a string variable.

`ro` attribute is should be set to the index of the item that you are trying to get. It can be a constant or a reference to an integer variable. The numbering of the items starts from 0. If the value of the ro attribute is outside of the bounds of the string list, the variable defined by the name attribute is set to `;`.

```
<screen id="string_list" next="#main_menu" >
  <!-- define the list -->
  <setvar name="string1" lo="First item;Second item;Third
item"/>
```

```
  <!-- get the number of items in the list -->
  <setvar name="integer1" lo="tmlvar:string1" op="number" />

  <!-- set the index -->
  <setvar name="integer2" lo="2" />

  <!-- get first item from the list -->
  <setvar name="string2" lo="tmlvar:string1" op="item" ro="0"
/>
  <!-- get another item from the list -->
  <setvar name="string3" lo="tmlvar:string1" op="item"
ro="tmlvar:integer2" />

  <!-- if item index is greater than the number of items in the
list, ';' is returned -->
  <setvar name="string4" lo="tmlvar:string1" op="item" ro="10"
/>
  <display>
    <table>
      <tr><td>
        string1: <getvar name="string1" /><br/>
        number of items: <getvar name="integer1" /><br/>
        string2: <getvar name="string2" /><br />
        string3: <getvar name="string3" /><br />
        string4: <getvar name="string4" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
string1	First item;Second item;Third item
integer1	3
string2	First item
string3	Third item
string4	;

Integer variable operations

Integer variables can be assigned values directly, or the following operations can be performed: `plus` and `minus`.

Explicit value assignments are described in "Assigning explicit values" on page 88. `plus` and `minus` operations are described below.

Note: formatter pattern for setting integer variables must always correspond to the pattern for the integer type. See "Formatter" on page 47 for the description of the valid patterns.

plus

The `plus` operation for integer variables adds the `lo` and `ro` attributes and assigns the result to the variable, referenced by the `name` attribute.

Both `lo` and `ro` can be constants, or reference integer variables.

If `lo` or `ro` are constants, they must correspond to the integer formatter pattern defined by the `format` attribute. If format attribute is empty, default integer formatter (`-0*`) is used. Default formatter will allow you to use both negative and positive values.

Referenced variables ignore the `format` attribute.

Note: to keep the code simple and more readable, avoid using the `format` attribute when performing the `plus` operation.

```
<screen id="int_plus" next="#int_menu">

  <setvar name="integer1" lo="-10" op="plus" ro="5" />

  <!-- formatter is applied to the 'ro' or 'lo' if they are
constants. Referenced variables ignore the formatter -->
  <setvar name="integer2" lo="$250" op="plus"
ro="tmlvar:integer1" format="$0*" />

  <!-- Best practice is not to use the 'format' attribute and
use 'lo' and 'ro values that conform to the default formatter -
0*  Below is the same operation as for 'integer2', using best
practices -->
  <setvar name="integer3" lo="250" op="plus"
ro="tmlvar:integer1" />

  <display>
    <table>
      <tr><td>
        integer1: <getvar name="integer1" /><br/>
        integer2: <getvar name="integer2" /><br />
        integer3: <getvar name="integer3" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
integer1	-5
integer2	245
integer3	245

minus

The `minus` operation for integer variables substracts the `ro` attribute from the `lo` and assigns the result to the variable, referenced by the `name` attribute.

Both `lo` and `ro` can be constants, or reference integer variables.

If `lo` or `ro` are constants, they must correspond to the integer formatter pattern defined by the `format` attribute. If format attribute is empty, default integer formatter (`-0*`) is used. Default formatter will allow you to use both negative and positive values.

Referenced variables ignore the `format` attribute.

Note: to keep the code simple and more readable, avoid using the `format` attribute when performing the `minus` operation.

```
<screen id="int_minus" next="#int_menu">

  <setvar name="integer1" lo="20" op="minus" ro="100" />

  <!-- formatter is applied to the 'ro' or 'lo' if they are
constants. Referenced variables ignore the formatter -->
  <setvar name="integer2" lo="1.20" op="minus"
ro="tmlvar:integer1" format="0.00" />
```

```
  <!-- Best practice is not to use the 'format' attribute and
use 'lo' and 'ro values that conform to the default formatter -
0*  Below is the same operation as for 'integer2', using best
practices -->
  <setvar name="integer3" lo="120" op="minus"
ro="tmlvar:integer1" />
  <display>
    <table>
      <tr><td>
        integer1: <getvar name="integer1" /><br/>
        integer2: <getvar name="integer2" /><br />
        integer3: <getvar name="integer3" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
integer1	-80
integer2	200
integer3	200

Date variable operations

Date variables can be assigned values directly, or the following operations can be performed: `plus` and `minus`.

Additional notes on explicit date variable setting

Explicit value assignments are described in "Assigning explicit values" on page 88. `plus` and `minus` operations are described below. However, some additional points should be noted regarding the explicit variable setting:

- Default value of a date variable is `1970/01/01 00:00:00` (the fields are `YYYY/MM/DD hh:mm:ss`). When you change only some of the fields, the fields that are not modified are taken from this value.
- This default value is in the terminal local time. To set the value as the GMT value, you must use `GMT` as part of the formatter
- If you explicitly set the date variable to `0000/00/00 00:00:00` (the fields are `YYYY/MM/DD hh:mm:ss`) this acts as a special value. It is an equivalent of an empty string for `<input>` or `<getvar>` elements.
- If `lo` attribute is a constant value, or a reference to a string variable, it must correspond to the date formatter pattern, set by the `format` attribute. If the `format` attribute is empty, default formatter `YYYY/MM/DD` is used.
- If `lo` attribute is an integer reference, this integer represents the number of seconds since the `1970/01/01 00:00:00`. Integer format is assumed to be `0*` (unsigned integer) and the value of the `format` attribute is ignored.
- If `lo` attribute is a reference to a date variable, all variable fields are copied and the `format` attribute is ignored.

```
<screen id="date_set" next="#date_menu">
    <!-- setting with default formatter YYYY/MM/DD -->
  <setvar name="date1" lo="2008/09/27" />
    <!-- setting with formatter: only the fields that are
mentioned are changed from the default value 1970/01/01 -->
  <setvar name="date2" lo="59" format="ss" />
    <!-- when using integer variables to set dates, and vice
versa, the integer contains the number of seconds from the
```

```
1970/01/01 00:00:00  'integer1' is set to the number of
seconds, equivalent to 2 days 1h 20 min and 59s-->
  <setvar name="integer1" lo="177659" />
  <setvar name="date3" lo="tmlvar:integer1" />
    <!-- string is de-formatted according to the date-type
formatter to get the date variable values -->
  <setvar name="string1" lo="1108" />
  <setvar name="date4" lo="tmlvar:string1" format="MMYY" />
    <!-- for date variable references 'format' attribute is
ignored all fields are copied -->
  <setvar name="date5" lo="tmlvar:date1" format="MM"/>

  <display>
    <table>
      <tr><td>
        date1: <getvar name="date1" /><br/>
        date2 all fields: <getvar name="date2"
format="YYYY/MM/DD HH:mm:ss" /><br />
        date3 all fields: <getvar name="date3"
format="YYYY/MM/DD HH:mm:ss" /><br />
        date4: <getvar name="date4" /><br />
        date5: <getvar name="date5" /><br />
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
date1	2008/09/27 00:00:00
date2	1970/01/01 00:00:59
date3	1970/01/03 01:20:59
date4	2008/11/01 00:00:00
date5	2008/09/27 00:00:00

plus

The `plus` operation for date variables adds the `ro` attribute and the `lo` and assigns the result to the date variable, defined by the `name` attribute.

`lo` attribute must reference a date-type variable. `ro` attribute represents the number of seconds to be added to the `lo`. Therefore, `ro` must contain an integer constant or a reference to an integer variable. The integer can be positive or negative.

The `format` attribute is ignored, so `lo` must conform to the default integer formatter `-0*`.

```
<screen id="date_plus" next="#date_menu" >
    <!-- set a date -->
  <setvar name="date1" lo="2008/08/28 00:00:00"
format="YYYY/MM/DD 00:00:00" />
    <!-- add 3600 seconds (1 hour) -->
  <setvar name="date2" lo="tmlvar:date1" op="plus" ro="3600" />
    <!-- 'lo' must be a reference to a date variable
    'ro' can be an integer constant or integer reference -->
    <!-- 31622400 seconds is equivalent to 366 days -->
  <setvar name="integer1" lo="31622400" />
  <setvar name="date3" lo="tmlvar:date2" op="plus"
ro="tmlvar:integer1" />
    <!-- you can add negative values -->
  <setvar name="date4" lo="tmlvar:date3" op="plus" ro="-59" />

  <display>
    <table>
      <tr><td>
```

```
        date1: <getvar name="date1" format="YYYY/MM/DD
HH:mm:ss" /><br/>
        date2: <getvar name="date2" format="YYYY/MM/DD
HH:mm:ss" /><br/>
        date3: <getvar name="date3" format="YYYY/MM/DD
HH:mm:ss" /><br/>
        date4: <getvar name="date4" format="YYYY/MM/DD
HH:mm:ss" /><br/>
      </td></tr>
    </table>
  </display>
</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
date1	2008/08/28 00:00:00
date2	2008/08/28 01:00:00
date3	2009/08/29 01:00:00
date4	2009/08/29 00:59:01

minus

The `minus` operation for date variables substracts the `ro` attribute from the `lo` and assigns the result to the date variable, defined by the `name` attribute.

`lo` attribute must reference a date-type variable. `ro` attribute represents the number of seconds to be substracted from the `lo`. Therefore, `ro` must contain an integer constant or a reference to an integer variable. The integer can be positive or negative.

The `format` attribute is ignored, so `lo` must conform to the default integer formatter `-0*`.

```
<screen id="date_minus" next="#date_menu" >

    <!-- set a date -->
  <setvar name="date1" lo="2008/08/28 00:00:00"
format="YYYY/MM/DD 00:00:00" />

    <!-- substract 3600 seconds (1 hour) -->
  <setvar name="date2" lo="tmlvar:date1" op="minus" ro="3600"
/>
    <!-- negative integers can be used -->
  <setvar name="integer1" lo="-3600" />
  <setvar name="date3" lo="tmlvar:date2" op="minus"
ro="tmlvar:integer1" />

</screen>
```

When the example above is processed, the variables will have the following values:

Variable	Value
date1	2008/08/28 00:00:00
date2	2008/08/27 23:00:00
date3	2008/08/28 00:00:00

Description

Allows to wrap a portion of inline TML content, overriding the values of core attributes specified for spanned inline elements with the values specified in the core attributes of the `<span>` element.

For example, you could use the `class` core attribute to apply the effects of CSS to the portion of text.

You should use the `<div>` element when you want to apply attributes to a block portion of TML data.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
, <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <form>, <dd>, <display>, , <print>, <td>, <th>	<a>, , , <getvar>, <input>, <textarea>

Example

See "Using and elements" on page 141.

<strtemplate>

Description

This tag assigns a formatted string of text to a variable of the string type.

The name of a variable is defined by the `name` attribute of the `<strtemplate>` element.

The value assigned to a variable is defined by the `format` attribute of the `<strtemplate>` element and its child `<getvar>` elements.

The `<strtemplate>` element can contain from 0 to 9 `<getvar>` elements.

The value of the `format` attribute may contain pieces of text and placeholders representing results of the child `<getvar>` elements processing.

Each placeholder is specified like this: `%n`, where `n` is an integer number in the range from 1 to 9, representing the number of the child `<getvar>` element. For example, `%3` would mean the third of the `<getvar>` elements.

To insert the `%` character as part of the variable value, you should 'escape' the special meaning of `%` by placing the backslash character (`\`) in front of it: `\%`.

If used, the `<strtemplate>` element(s) should appear in the beginning of the `<screen>` element right after the `<setvar>` element(s) (if present).

Attributes

Name	Type	Description
`name`	string	Required attribute specifying the name of a variable.
`format`	string	Required attribute defining the value of a variable. The attribute's value may contain pieces of text and placeholders representing the results of the child `<getvar>` elements processing. Placeholders have the following form: `%n`, where `n` is an integer number from 1 to 9 corresponding to the child `<getvar>` element's number.

Related elements

Parent elements	Child element
<screen>	<getvar> The <strtemplate> element may contain 0

	to 9 <getvar> elements.

Example

```
<setvar name="person.first_name" lo="John"/>
<setvar name="person.last_name" lo="Smith"/>
<strtemplate name="tell_last_name" format="%1's last name is
%2">
  <getvat name="person.first_name"/>
  <getvat name="person.last_name"/>
</strtemplate>
```

In this example the string `John's last name is Smith` is assigned to variable `tell_last_name`.

<submit>

Description

This element defines a submit screen which is used to send the values of TML variables or a TML log to the Application Server for processing. The variables whose values should be submitted are specified by means of the child `<getvar>` elements that are listed one by one. In the case of a log the `<submit_log>` child element is used.

While parsing the `<submit>` element, MicroBrowser generates a dynamic TML page with a single `<tmlpost>` screen where the same variables are specified using `<postvar>` elements.

Note: Predefined variables `terminal.datetime` and `terminal.itid` are submitted to the Application Server using the `date` and `itid` attributes of the corresponding `<tmlpost>` element, so you don't need to specify them explicitly.

All values of the `date` type within the `<tmlpost>` will have this format: `DD MM YYYY hh:mm:ss GMT`, where the symbols `D`, `M`, `Y`, `h`, `m`, `s` have the usual meaning (see "Dates" on page 48) and `GMT` is the formatter that instructs to present the variable in the **G**reenwich **M**ean **T**ime.

Aside from specifying the variables or the log, you should provide some auxiliary data required to complete the submission, such as target server and the screen to switch to if the connection to the Application Server is lost. The data is defined using the child `<econn>` element and the `<submit>` element's attributes.

Attributes

Name	Type	Description
`tgt`	URI	The required attribute that specifies the URI of the host module component that is responsible for accepting terminal requests. The URI is embedded into the HTTP POST request generated by the `<tmlpost>` element. Relative URIs specified in the attribute are converted to absolute by the Incendo Online Gateway data converter, see *Incendo Online Gateway User Guide* for details.
`econn`	URI	Specifies the URI of the screen to switch if the connection to the Application Server is lost during data submission. If the attribute is specified, the child `<econn>` element can be omitted. It can be a constant or a variable reference. It can also be one of special URIs - `back`, `menu` or `cancel` (see on page 16).
`cache`	token	Specifies whether the HTTP request with the submitted information should be cached by the terminal. Possible values are `"allow"` and `"deny"`,

		see cache attribute of the <tmlpost> element on page 107. Default value is "deny".

Related elements

Parent elements	Child element
<screen>	Sequence: <econn> ? <getvar> * | <submit_log>

Remarks

By default, after generating the dynamic screen, MicroBrowser switches to the screen specified by URI in the <next> element. Note, that the host part of TML application can override this behaviour by responding with a dynamic TML page which is always processed by MicroBrowser immediately. The dynamic page can have a different next screen specified and this screen will take precedence.

Example

```
<submit tgt="/magcard/authtxn" econn="#mag_rm">
  <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="oebr.transid"/>
  <getvar name="payment.amount"/>
</submit>
```

<submit_log>

Description

Used within the <submit> element (see page 98) to submit a log to an Application Server.

The attribute name defines which of the logs is to be submitted.

Consequently, the name attribute should reference an existing log's name (i.e. be the same as the name attribute of the corresponding <logdcl> element, see page 79).

Attributes

Name	Type	Description
name	NMTOKENS	Required attribute specifying the name of the log that should be submitted to an Application Server. The attribute value should correspond to the name attribute of the corresponding <logdcl> element, see page 79.

Related elements

Parent elements	Child element
<submit>	none

Example

```
...
```

```
<logdcl name="my-log" ..
...
  <screen id="submit-my-log" ..
    <submit ..>
      < submit_log name="my-log"/>
    </ submit >
</screen>
...
```

<table>

Description

This element defines a table. A table is a structural presentation of data using rows and columns.

The table structure is built using the child elements.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`border`	Number	Instructs MicroBrowser to draw lines around the entire table and all of the rows and cells. You can specify the thickness of the border line in pixels.
`cellpadding`	Number	Sets the amount of white space (in pixels) between a cell border and the contents of the cell.
`cellspacing`	Number	Sets the amount of white space (in pixels) between adjacent cells and between a cell and the outer border of the table.
`rules`	TRules	Defines which border lines should appear inside the table. The possible values are `"none"` – to have no internal borders, `"rows"` – to have borders between rows, `"cols"` – to have borders between columns and `"all"` – to have borders between rows and columns. If rules attribute is omitted, but border attribute is set, it is considered that `rules="all"`. If border attribute is also omitted or set to zero, it is considered that `rules="none"`.
`width`	Length	Sets the width of a table. It can be declared either as an integer number of pixels or as a percentage of the width of the terminal display.

Related elements

Parent elements	Child element
<dd>, <display>, <form>, , <print>, <td>, <th>	Sequence: <col> | <colgroup> * <thead> | <tfoot> ? <tbody> | <tr> *

Example

```
<table class="c_small" width="100%">
  <tr>
    <td>
      <getvar name="oebr.posts_print.var_name"/>
    </td>
    <td>
      <getvar name="oebr.posts_print.var_val"/>
    </td>
  </tr>
```

```
</table>
```

<tbody>

Description

This tag defines the body portion of a table where the data is displayed. The body of a large table can be scrolled while both the header and footnote table sections (defined by `<thead>` and `<tfoot>` elements) remain visible.

Note: the correct order for using these elements is `<thead>`, then `<tfoot>`, and `<tbody>`.

You can use multiple body sections when you want to have borders between groups of table rows.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.
`class`	NMTOKENS	See "class attribute" on page 115.
`align`	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are `center`, `left` and `right`.
`valign`	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are `bottom`, `middle` and `top`.

Related elements

Parent elements	Child element
<table>	<tr>*

Example

```
<table border="1">
  <thead>
    <tr><th>Header 1</th><th>Header 2</th></tr>
  </thead>
  <tfoot>
    <tr><td colspan="2">footnote</td></tr>
  </tfoot>
  <tbody>
    <tr><td>data cell 1</td><td>data cell 2</td></tr>
  </tbody>
</table>
```

<td>

Description

This element is used to create cells with data or text that you want to display in the table. You can type the text directly between the opening and closing tags. The number of cells can be arbitrary.

The coding sequence is:

```
<tr><td> ... your data here ... </td></tr>
```

Use the `<th>` element to create a header cell for the column of cells and `<tr>` element to create table rows.

Attributes

Name	Type	Description

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.
colspan	Length	Allows a cell to span horizontally two or more columns (cells). The default value is 1. With the rowspan attribute you can create data cells that encompass several rows and columns.
rowspan	Length	The attribute allows a cell to extend down two or more rows. The default value is 1. With the colspan attribute you can create data cells that encompass several rows and columns.
align	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are center, left and right.
valign	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are bottom, middle and top.

Related elements

Parent elements	Child element
<tr>	<a>, , , <getvar>, <input>, <textarea>, , <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <form>, <dd>, <display>, , <print>, <td>, <th>

Example

See the example for the <tbody> element on page 101.

<textarea>

Description

This tag is used within the <form> element to define a multi-line field for entering text. You can use text areas when it is required to accept user input which takes more than a line of text (e.g. home or e-mail address).

The element has a number of validity check attributes that allow restricting user input.

If the validity check fails, MicroBrowser switches to the URI specified in the child <baddata> element. If you omit <baddata>, validity check is skipped.

If the input data is valid, the value entered by the user is assigned to the variable specified in the mandatory name attribute.

The way the data is displayed is controlled by means of the format attribute.

When MicroBrowser encounters <textarea> element, it switches the terminal to the form processing mode, see "Form processing" on page 30.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.
alt	string	Provides the alternate text to the terminals which are not capable of displaying images and forms.
name	string	Specifies the name of the variable which receives the value entered by the user.
rows	Length	Defines the width of the input fields in characters
cols	Length	Defines the height of the input field in characters.

Name	Type	Description
readonly	token	Displays the field value that cannot be changed by the user.
equal	string	This is a validity check attribute. The data entered by the user is checked whether it is equal to the value specified by the attribute. If not, the flow control is passed to the child <baddata> element.
not_equal	string	This is a validity check attribute. The data entered by the user is checked whether it is unequal to the value specified by the attribute. If not, the flow control is passed to the child <baddata> element.
max	string	This is a validity check attribute. The data entered by the user is checked whether it is less than the maximum value specified by the attribute. If not, the flow control is passed to the child <baddata>. For strings, the length of the entered string is checked.
min	string	This is a validity check attribute. The data entered by the user is checked whether it is more than the minimum value specified by the attribute. If not, the flow control is passed to the child <baddata>. For strings, the length of the entered string is checked.
value	string	Assigns an initial value, such as a text or number, to the input field. The value can be a constant or a variable reference.
format	Formatter	Defines a pattern for formatting and displaying the input value.

Related elements

Parent elements	Child element
<form>	<baddata>?

See also <input> element on page 69.

Example

```
<form>
  <p>New address:<br/>
    <textarea name="merchant_address" rows="8" cols="16"
    value="tmlvar:payment.merchant_address"/>
  </p>
</form>
```

<tfoot>

Description

This tag defines the footer portion of a table. The body of a large table (defined by <tbody> element) can be scrolled while both the header (defined by <thead> element) and footer table sections remain visible.

Note: the correct order for using these elements is <thead>, then <tfoot>, and <tbody>.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Name	Type	Description
`align`	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are `center`, `left` and `right`.
`valign`	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are `bottom`, `middle` and `top`.

Related elements

Parent elements	Child element
<table>	<tr>*

Example

See the example for the `<tbody>` element on page 101.

<tform>

Description

This element defines a terminal form screen which is used for accepting user card data. The data can be either read from a card or entered using a terminal PIN pad. The read data is assigned to the predefined variables according to the input type.

Using a child `<prompt>` element, you can define a short message to be rendered on the terminal screen, prompting a user for input.

The element has no attributes.

Related elements

Parent elements	Child element
<screen>	Sequence: <baddata> ? (<card> * <prompt>?) | <pinentry>

Example

```
<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 icc="icc" prompt="Enter PIN"/>
</tform>
```

<th>

Description

This tag is used to create a header cell for a column of cells within the `<table>` element.

The coding sequence is:

```
<tr><th> ... your header here ... </th></tr>
```

Use the `<td>` to create table cells and `<tr>` to create table rows.

Attributes

Name	Type	Description
`id`	ID	See "id attribute" on page 115.

Name	Type	Description
class	NMTOKENS	See "class attribute" on page 115.
colspan		Allows a cell to span horizontally two or more columns (cells). The default value is 1. With the rowspan attribute you can create data cells that encompass several rows and columns.
rowspan		The attribute allows a cell to extend down two or more rows. The default value is 1. With the colspan attribute you can create data cells that encompass several rows and columns.
align	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are center, left and right.
valign	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are bottom, middle and top.

Related elements

Parent elements	Child element
<tr>	<a>, , , <getvar>, <input>, <textarea>, , <dl>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hr>, <img>, , <p>, <pre>, <table>, , <form>, <dd>, <display>, , <print>, <td>, <th>

Example

See the example for the <tbody> element on page 101.

<thead>

Description

This element defines the header portion of a table. The body of a large table (defined by <tbody>) can be scrolled while both the header and footnote (defined by <tfoot>) table sections remain visible.

Note: the correct order for using these elements is <thead>, then <tfoot> and <tbody>.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.
align	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are center, left and right.
valign	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are bottom, middle and top.

Related elements

Parent elements	Child element
<table>	<tr>*

Example

See the example for the <tbody> element on page 101.

<tml>

Description

The element immediately follows the XML declaration and indicates the beginning and the end of a TML page enclosing the whole page content. It is the root element of a TML document.

Attributes

Name	Type	Description
`cache`	token	Specifies whether the TML page should be cached within the terminal. Possible values are `"allow"` and `"deny"`. Default value is `"allow"`.
`xmlns`	URI	Specifies a unique URI pointing to a TML page that defines the namespace for all elements and attributes valid within your page.

Related elements

Parent elements	Child element
none	Sequence: <head> ? <vardcl> * <logdcl> * <screen> *

Example

```
<tml xmlns="http://localhost:8080/tml">
.... [the page content] ....
</tml>
```

<tmlpost>

Description

When MicroBrowser encounters a `<submit>` element, it generates a dynamic TML page with a single `<tmlpost>` element inside.

The `<tmlpost>` element posts the variable values specified in the corresponding `<submit>` element to the Application Server. The element can appear only within dynamic screens generated by MicroBrowser.

The posted variables are listed one by one (using the `<postvar>` element) in the same order as they appear within the corresponding `<submit>` element. The terminal ID and current date required for submission authorisation are specified by the element attributes.

The posted data is wrapped into the HTTP POST request which is submitted to the server identified by the URI.

The request can be submitted in online or offline mode, which depends on the value of predefined variable `oebr.submit_mode`. Note, that terminal has a limited amount of memory to store offline submissions. If the memory is full, the offline submission is turned into online and the value of the `submit_mode` variable is changed automatically.

Online submissions are posted to the Application Server immediately.

Attributes

Name	Type	Description
`date`	string	The required attribute which specifies the current date and time taken from the predefined variable `terminal.datetime`. This and all other dates (in corresponding `<postvar>` elements) are sent in the following format: `DD MM YYYY hh:mm:ss GMT`, where the symbols `D`, `M`, `Y`, `h`, `m`, `s` have the usual meaning (see " Dates" on page 48) and GMT stands for **G**reenwich **M**ean **T**ime.
`itid`	string	Specifies the Ingenico terminal ID taken from the predefined variable `terminal.itid`.
`postid`	string	Specifies a unique transaction identifier taken from the predefined variable `oebr.transid`. The identifier is generated by MicroBrowser.
`cache`	token	Specifies whether the TML page should be cached within the terminal. Possible values are `"allow"` and `"deny"`. Default value is `"allow"`.

Related elements

Parent elements	Child element
none	<postvar>*

Example

```
<tmlpost post_id="3" date="25 07 2007 20:22:08 GMT" itid="100"
xmlns="http://localhost:8080/tml">
  <postvar name="oebr.submit_mode" type="string"
value="online"/>
  <postvar name="payment.trans_type" type="string"
value="debit"/>
  <postvar name="card.pan" type="string"
value="3540829999421012"/>
  <postvar name="card.issue_number" type="integer" value="0"/>
  <postvar name="card.expiry_date" type="date" value="10 09
2009 00:00:00 GMT"/>
  <postvar name="card.effective_date" type="date" value="10 09
2006 00:00:00 GMT"/>
  <postvar name="oebr.transid" type="integer" value="1"/>
  <postvar name="payment.amount" type="integer" value="12"/>
  <postvar name="payment.amount_other" type="integer"
value="0"/>
  <postvar name="currency_code" type="string" value="GBP"/>
  <postvar name="card.input_type" type="integer" value="2"/>
  <postvar name="card.mag.iso2_track" type="string"
value="4475200254859014=05051261001363500000"/>
  <postvar name="card.emv.aid" type="string"
value="A0000000651010"/>
  <postvar name="card.emv.aip" type="integer" value="31744"/>
  <postvar name="card.emv.auc" type="integer" value="65472"/>
  <postvar name="card.emv.atc" type="integer" value="5"/>
  <postvar name="card.emv.aac" type="opaque" value=""/>
  <postvar name="card.emv.tc" type="opaque" value=""/>
  <postvar name="card.emv.arqc" type="opaque" value="23 48 00
00 00 00 00 00"/>
  <postvar name="card.emv.iad" type="opaque" value=""/>
  <postvar name="card.emv.tvr" type="opaque" value="00 00 00 08
00"/>
  <postvar name="card.emv.unumber" type="integer" value="41"/>
```

```
  <postvar name="payment.txn_date" type="date"
value="1108592990"/>
  <postvar name="oebr.time_zone" type="string" value="0"/>
</tmlpost>
```

<tr>

Description

This element defines a row in a <table> element. The number of rows can be arbitrary.

A row can contain one or more cells created using either the <td> or <th> element.

The coding sequence is:

```
<tr><th> ... your header here ... </th></tr>
<tr><td> ... your data here ... </td></tr>
```

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.
align	token	Sets the horizontal alignment of the cell contents for all the cells in a single row. The possible values are center, left and right.
valign	token	Sets the vertical alignment of the cell contents for all of the cells in a single row. The possible values are bottom, middle and top.

Related elements

Parent elements	Child element
<table>	<th>* <td>*

Example

```
<table width="85%" border="2" cellpadding="3" cellspacing="5"
align="center">
  <tr>
    <th colspan="2">table header</th>
  </tr>
  <tr>
    <td>The <b>tr</b> tag creates the row</td>
    <td>The <b>td</b> tag creates cells</td>
  </tr>
  <tr>
    <td>cell data</td>
    <td>cell data</td>
  </tr>
  <tr>
    <td>cell data</td>
    <td>cell data</td>
    </tr>
</table>
```


Description

This tag is used to define an unordered list which represents a collection of list items that do not follow a particular order. Each item in the list is preceded by asterisk (*).

Use <li> element to display content of each item.

Note: Unordered lists cannot be nested.

You can use the <ol> element to create an unordered list and the <dl> element to create a definition list.

Attributes

Name	Type	Description
id	ID	See "id attribute" on page 115.
class	NMTOKENS	See "class attribute" on page 115.

Related elements

Parent elements	Child element
<baddata>, <form>, <prompt>, <dd>, <display>, , <print>, <td>, <th>	*

Example

```
<ul class="c_normal">
  <li class="c_normal">Normal item 1</li>
  <li class="c_normal">Normal item 2</li>
</ul>
```

<vardcl>

Description

The element is used to declare a TML variable (see "Variables and constants" on page 21).

When declaring a variable you specify its name and type by means of the `name` and `type` attributes. If the `type` attribute is omitted, the variable is assumed to be of the `string` type (`type="string"`).

The value initially assigned to the variable is defined by the `value` and `format` attributes. In the absence of the `format` attribute, the value of the `value` attribute will be stored in the variable as its value.

The `format` attribute defines the 'default' formatting pattern for the variable (see "Formatter" on page 47) – one used for displaying and printing the variable's value if no other pattern is specified in corresponding `<getvar>` elements (see "<getvar>" on page 67). The `format` attribute also specifies how the value of the `value` attribute should be formatted (or de-formatted – depending on the variable's type[17]) prior to assigning an initial value to the variable.

The element's `volatile` attribute (with the possible values `"yes"` or `"no"`) is used to specify whether or not the variable's value should be kept when the terminal is turned off or rebooted, see "Volatile and non-volatile variables" on page 23. By default, a variable is assumed to be volatile (`volatile="yes"`) meaning that you don't mind if the variable's value is lost when the terminal is rebooted.

Note: if a non-volatile (`volatile="no"`) variable is declared from within a non-cached page, it may still be destroyed if the terminal cache is cleared. To avoid this situation, declare your persistent (non-volatile) variables from within the TML pages where the `cash` attribute of the `<tml>` tag is set to `"allow"`.

You can define access permissions for the variable (see "Permissions" on page 51) using the `perms` attribute. By default, `perms="rw-rw"` is assumed.

Variable declarations (that is, the `<vardcl>` elements) should be placed at the beginning of the page between the `<head>` element (see "<head>" on page 68) and the `<screen>` elements (see "<screen>" on page 85).

Attributes

[17] De-formatting is a reverse operation with regard to formatting, see "Formatting and deformatting"on page 21. Prior to assigning a value to a variable, string values are formatted, while date and integer values are de-formatted.

Name	Type	Description
`name`	string	The required attribute defining the name of the variable.
`type`	token	The variable's type. Possible values are: `"string"`, `"integer"`, `"date"` and `"opaque"`. The default value is `"string"`.
`volatile`	token	Defines the place where the variable is physically stored. Two options are available: `"yes"` – the variable is stored in the terminal RAM (**R**andom **A**ccess **M**emory). The RAM is fast but the variable's value is lost when the terminal is switched off or rebooted. `"no"` – the variable is stored in the terminal flash memory. Flash memory is slower but the variable value is preserved even if the terminal is switched off. The default value is `"yes"`.
`value`	string	Defines a value which – after processing according to the value of the `format` attribute – is assigned to the variable. In the absence of the `format` attribute, the value of this attribute corresponds to the initial value of the variable. If the attribute is omitted, the value assigned to a variable depends on its type: • An empty string is assigned to a `string` variable. • The value of the Epoch `1970/01/01 00:00:00` is assigned to a variable of the `date` type. • `0` is assigned to an integer variable.
`format`	Formatter	Defines a formatting pattern for the variable (see "Formatter" on page 47). This attribute is used for a number of purposes. First of all, it defines the 'default' formatting pattern for the variable: if, for example, the formatting pattern is not specified in a `<getvar>` element (see "<getvar>" on page 67), this pattern will be used to display or print the variable's value. The attribute is also used to specify how to process (format or de-format) the value of the `value` attribute prior to assigning an initial value to the variable. For `string` variables, the value of the `value` attribute is formatted according to the specified formatting pattern and then is assigned to the variable. For date and integer variables: the value of the `format` attribute is used to de-format the value of the `value` attribute. The result is then assigned as initial value to the variable. If the attribute is omitted, the values `c*`, `YYYY/MM/DD` and `0*` and `base64` are assumed by default for variables of the `string`, `date`, `integer` and `opaque` types respectively.
`perms`	Permissions	Defines access permissions for the variable. The default value is `"rw-rw"`. For more information, refer to "Permissions" on page 51.

Related elements

Parent elements	Child element
<tml>	none

Remarks

- To specify that an integer variable can take negative values you should define a formatting pattern starting with the minus sign (–) in the variable declaration, for example, `format="–0*"`.
- To fully define the date and time, a variable of the `date` type should contain information about the year, month and day as well as hours, minutes and seconds. If some of this information is missing it is taken from the 'default' date `1970/01/01 00:00:00`.

Example

```
<vardcl name="string-example" value="Ringo Starr"
format="c#6c*c#"/>

<vardcl name="number-example-1" type="integer" value="12.99"
format="0*.00"/>

<vardcl name="number-example-2" type="integer" value="-100"/>

<vardcl name="date-example" type="date" value="20"
format="YY"/>
```

In this example the following values are assigned to the variables:

- `******Star*` is assigned to `string-example`: The `type` attribute is omitted so the variable, by default, is assumed to be of the `string` type. The first six characters and the last character within the string Ringo Starr are replaced with asterisks (*). The result is then assigned as a value to the variable.
- `1299` is assigned to `number-example-1`: Application of the formatting pattern `0*.00` to the value `1299` would give `12.99` (the dot would be added in front of the last two digits). Thus, the reverse operation, that is, de-formatting of the value `12.99` with the pattern `0*.00` gives `1299`.
- `100` is assigned to `number-example-2`: The `format` attribute is omitted so the default formatting pattern for numbers (`0*`) is used. Since this pattern does not start with the minus sign, the variable's value can not be negative. Thus, the minus in front of `100` is discarded prior to assigning a value to the variable.
- `2020/01/01 00:00:00` is assigned to `date-example`: The value of the `format` attribute (`"YY"`) tells that the value of the `value` attribute should be interpreted as the last two digits of the year. Since 20 is less than 50, the `20` here corresponds to the year 2020 (rather than 1920). The rest of the information defining the date and time is taken from the 'default' date (that is, `1970/01/01 00:00:00`).

<variant>

Description

An element defining destination screen's URI – alternative to the URI specified by the `uri` attribute of the parent element.

The main purpose of this element is to create branches in the application logic. You can define a number of `<variant>` elements to create multiple branching choices.

The usage pattern for this element is

```
<variant uri="[destination screen's URI]" [condition]/>
```

where the [destination screen's URI] is the URI of the screen Incendo Online MicroBrowser should switch to if the [condition] is met.

There are two possible way of specifying the [condition]. It may be specified as a logical expression that needs to be evaluated, or it may be associated with a specific terminal keypad key press.

In the first of the cases you put the following construction in place of the [condition]:
lo="[left operand]" op="[operation]" ro="[right operand]" format="[formatter]"

Then, if the statement defined in such a way is found to be true, the [condition] is considered to be satisfied.

Please note that if the op="[operation]" is missing, the op="equal" is assumed; the part format="[formatter]" is optional.

Conditions specified in this way are evaluated one by one starting from the first of the <variant> elements. As soon as the first true statement is found, a jump to the appropriate next screen is performed.

If neither of the conditions specified by corresponding <variant> elements is true, the URI of the next screen is defined by the uri attribute of the parent element.

The other possibility is to specify the [condition] in the following way:
key="[key name]"
where the [key name] is the name of a terminal keypad key such, for example, as 1, 2, 3 and so on – for an alphanumeric key. In this case the [condition] corresponds to the situation when the terminal keypad key specified by the key attribute is pressed.

If a user then presses a key specified in the <variant> element, its uri attribute will define the screen Incendo Online MicroBrowser should switch to.

<variant> elements of both types (that is, with a logical expression and with the key attribute) can be present within one parent element. See "Remarks" on page 113 for information on how these two different types of <variant> elements are processed.

Note: the contents of ro and lo attributes must be of the same type.

If one attribute contains a variable reference and the other is a constant, the type of the constant is assumed to be the type of the referenced variable. Formatter of the appropriate type is applied to that constant, before evaluating the condition.

If both attributes are variable references, they must reference same type of variables. If both lo and ro values are constants, they are evaluated as strings

Attributes

Name	Type	Description
uri	Valref	The *required* attribute specifying the URI of the screen Incendo Online MicroBrowser should switch to if the condition defined by the other attribute(s) of the element is satisfied. The value of the attribute can be represented by a constant or a variable reference. It can also be one of special URIs - back, menu or cancel (see on page 16).
lo	string	The attribute representing the left operand of the logical expression. The attribute value can be a constant, or a variable reference. The attribute is *required* if the key attribute is not present. Its use is illegal in presence of the key attribute.

Name	Type	Description
ro	string	The attribute representing the right operand of the logical expression. The attribute value can be a constant, or a variable reference. The attribute is *required* if the key attribute is not present. Its use is illegal in presence of the key attribute.
op	token	The attribute defining a logical operation that is performed on the left and right operands. The possible values are strings "equal", "not_equal", "less" and "less_or_equal". For string operands the "contains" operation can also be used. This operation returns true, if the string represented by the left operand contains as its part the string represented by the right operand. If missing, the op="equal" is assumed. In presence of the key attribute, this attribute is ignored.
format	Formatter	If one of the operands (lo or ro) is a constant, this attribute defines the formatting pattern (see "Formatter" on page 47) applied to this constant prior to performing the operation defined by the op attribute. If both operands are constants or variable references, the attribute is ignored. It is also ignored if the key attribute is present.
key	token	The attribute specifying a terminal keypad key which, if pressed, initiates a jump onto the screen whose URI is defined by the uri attribute. Possible values are: "0", "1", "2", …, "9", "00", "f1", "f2", …, "f9", "down", "up", "menu", "stop", "cancel", and "enter". All these correspond to keypad key names. The attribute is *required* if the lo and ro are not present. The use of the attribute is *illegal* if the lo and ro are present.

Related elements

Parent elements	Child element
<baddata>, <form>, <prompt>, <dd>, <display>, , <print>, <td>, <th>	*

Remarks

There are two possible ways of specifying the condition for the <variant> element. It is either specified as a logical expression by the means of lo, ro, and op attributes, or it may correspond to a key press, and in this case the only the key attribute is used. The condition is assumed to be met, if the result of logical expression evaluation is true, or the specified key is pressed respectively.

The logical expression and the key attribute are mutually exclusive, that is, they can not be used in the same <variant> element. In other words, if the key attribute is present, the use of the lo, ro, and op attributes is illegal, and vice versa.

<variant> elements with a logical expression and the ones with the key attribute are processed in different ways. The main difference is basically *when* those two different types of <variant> elements are processed.

The <variant> elements with the key attribute are processed when the corresponding screen is *active*, while the type with a logical expression is processed when Incendo Online MicroBrowser is *about to switch the next screen*.

Whenever a user presses a key, Incendo Online MicroBrowser checks if within the active screen there is a <variant> element which associates this key with certain URI. If such element is found, the MicroBrowser jumps onto the screen with the URI specified by the uri attribute of that <variant> element.

The <variant> elements containing logical expressions are processed when Incendo Online MicroBrowser 'decides' or is instructed to *leave the current screen* and switch onto another. This can, for example, happen when printing (i.e. processing of a print screen) is completed, or when the timeout specified for a screen without hyperlinks elapses. In such cases to find out which screen is going to be next, Incendo Online MicroBrowser starts analysing the <next> element to consider various possible alternatives (that is, the <variant> elements with a logical expression – if there are such).

Example

The following very simple TML application illustrates the use of <variant> elements.

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

  <head>
    <defaults menu="emb://embedded.tml"
    cancel="emb://embedded.tml"/>
  </head>

  <vardcl name="my-name"/>
  <vardcl name="my-phone-number"/>

  <screen id="home" timeout="5">
    <setvar name="my-name" lo="Jane"/>
    <setvar name="my-phone-number" lo="223-33-22"/>
    <next uri="#other">
      <variant uri="#variant-1" key="1"/>
      <variant uri="#variant-2" op="equal" lo="Jane"
      format="c5" ro="tmlvar:my-name"/>
      <variant uri="#variant-3" op="equal" lo="Jane"
      format="c5" ro="Jane"/>
    </next>
    <display>
      <h1>My Home Page</h1><hr/>
        Press "1" to see my name.<br/>
        Just wait and do nothing<br/> to know my phone number
    </display>
  </screen>

  <screen id="variant-1">
    <display>
      <h1>My Name</h1><hr/>
      My name is <getvar name="my-name"/>.<br/>
      <a href="#home">Go to my home page</a>
    </display>
  </screen>

  <screen id="variant-3">
    <display>
      <h1>My Phone Number</h1><hr/>
```

```
      Here it is: <getvar name="my-phone-number"
      format="c#4c*"/><br/>
      Oops!..Can you read it?<br/>
      <a href="#home">Go to my home page</a>
    </display>
  </screen>

</tml>
```

Within the `home` screen, the variables `my-name` and `my-phone-number` are set to `"Jane"` and `"223-33-22"` respectively.

If when viewing this screen a user presses `1`, the first of the `<variant>` elements 'fires' (`key="1"`), and Incendo Online MicroBrowser jumps onto the `variant-1` screen.

If, when staying on the `home` screen, a user – within 5 seconds (`timeout="5"`) – does nothing or presses a key other than `1`, `menu` or `cancel` (`<defaults menu="emb://embedded.tml" cancel="emb://embedded.tml"/>`), processing of the `<next>` element is initiated.

First, the logical expression within the second of the `<variant>` elements is evaluated.

Since one of the operands is a constant (`lo="Jane"`), the formatting pattern `c5` (`format="c5"`) is applied. The intermediate result is a string `"Jane-"` which is then compared with the value of the variable `my-name` (`op="equal" ro="tmlvar:my-name"`).

The statement being evaluated is: `"Jane-"` is equal to `"Jane"`.

Since the result of evaluation is *false*, Incendo Online MicroBrowser switches onto the next `<variant>` element.

Both operands are constants within the logical expression of this `<variant>` and, consequently, the `format` attribute is ignored. The statement being evaluated, thus, is `"Jane"` is equal to `"Jane"` which, obviously, is *true*.

The `<variant>` element 'fires,' and Incendo Online MicroBrowser jumps onto the screen `variant-3`.

Core TML attributes

class attribute

Syntax

`class="name"`

Description

The attribute is used to assign the name of a style sheet class to a particular TML element.

Example

See "Using the class attribute" on page 141.

id attribute

Syntax

`id="name"`

Description

In `<screen>`, the attribute is used for defining the screen name. When you need to define a link to a particular screen you use the screen name (preceded with # symbol) as part of its relative URI.

The `id` must be unique within the document and every element should have only one id.

The `id` must begin with a letter or an underscore (_). The rest of the value can contain any alphanumeric characters. Note that for many elements this attribute is mandatory.

Important: in TML, the length of a screen name should not exceed 12 characters

Example

```
<screen id=submit cancel="#init_prompt" next="#check_res">
...
<screen/>
```

Pre-defined TML Variables

7

This chapter lists and describes predefined TML variables – the ones declared on the page `embedded.tml` (see "Predefined variables" on page 22 and "Embedded terminal software" on page 39).

The tables used for presentation of information about the variables contain the following columns:

• **Name.** This column contains the names of the variables arranged in alphabetical order.

• **Type.** In this column the types of predefined variables are indicated. (For the list of data types supported by TML refer to "Variables and constants" on page 21.)

• **Perm.** Permissions - in this column the access permissions for a variable are specified (see "Scopes of variables" on page 22 and "Permissions" on page 51).

• **Description.** In this column the meaning and/or the purpose of the variable is explained. Where appropriate, the possible values are enumerated and default value is presented.

Audio variables

This section describes predefined TML variables related to various audio options. Most of variables in this group are used to assign different kinds of sounds to different events such, for example, as a press of a key on the terminal keypad, swiping of a magnetic card, insertion of a smart card, and so on.

Possible values for audio variables are:

- `"sound_click_low"`, `"sound_click_midtone"`, `"sound_click_high"`
- `"sound_short_low"`, `"sound_short_midtone"`, `"sound_short_high"`
- `"sound_long_low"`, `"sound_long_midtone"`, `"sound_long_high"`

The sound names in this list are self-explanatory; they describe the sounds in terms of duration and pitch. The sound duration increases in the order click – short – long. The sound pitch increases in the row low – midtone – high.

Name	Type	Perm	Description
`audio.any_key`	string	`rwxr-`	The sound that the terminal produces when an alphanumeric, the **DOWN**, the **UP**, or the navigation key on the terminal keypad is pressed. Possible values are described above. Default value is `"sound_click_midtone"`.
`audio.app_error`	string	`rwxr-`	The sound that the terminal produces when Incendo Online MicroBrowser encounters an error. Possible values are described above. Default value is `"sound_click_midtone"`.
`audio.app_start`	string	`rwxr-`	The sound that the terminal produces when Incendo Online MicroBrowser is started. Possible values are described above. Default value is `"sound_click_midtone"`.
`audio.app_stop`	string	`rwxr-`	The sound that the terminal produces when Incendo Online MicroBrowser is stopped. Possible values are described above. Default value is `"sound_click_midtone"`.

Name	Type	Perm	Description
audio.barcode_failed	string	rwxr-	The sound that the terminal produces when the bar code scanner fails to read a bar code. Possible values are described on page 117. Default value is "sound_short_low".
audio.barcode_read	string	rwxr-	The sound that the terminal produces when the bar code scanner has successfully read the bar code. Possible values are described on page 117. Default value is "sound_short_high".
audio.cancel_key	string	rwxr-	The sound that the terminal produces when the **Cancel** (red) key on the terminal keypad is pressed. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.clear_key	string	rwxr-	The sound that the terminal produces when the **C** (yellow) key on the terminal keypad is pressed. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.connection_close	string	rwxr-	The sound that is produced when the terminal disconnects from Incendo Online Gateway. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.connection_open	string	rwxr-	The sound that is produced when the terminal connects to Incendo Online Gateway. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.emv_insert	string	rwxr-	The sound that the terminal produces when a smart card is inserted into the card reader. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.emv_remove	string	rwxr-	The sound that the terminal produces when a smart card is removed from the card reader. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.enter_key	string	rwxr-	The sound that the terminal produces when the **OK** (green) key on the terminal keypad is pressed. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.focus_get	string	rwxr-	The sound that the terminal produces when an element on the terminal screen such, for example, as data field is brought in focus. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.mute	string	rwxr-	The variable whose value defines whether the mute option is on or off. This variable is used to turn the sounds for all events on or off. Possible values are "on" (the sounds for all events are off) or "off" (the sounds for all events are on). Default value is "off".

Name	Type	Perm	Description
audio.next_ref	string	rwxr-	The sound that the terminal produces when the next link on the terminal screen is brought in focus. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.next_url	string	rwxr-	The sound that the terminal produces when the link which is in focus is selected. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.paper_feed_key	string	rwxr-	The sound that the terminal produces when the paper feed key (the one with the arrow pointing up) on the terminal keypad is pressed. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.swipe	string	rwxr-	The sound that the terminal produces when a magnetic card is swiped through the card reader. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.swipe_failed	string	rwxr-	The sound that is produced when the terminals fails to read the data from the magnetic card. Possible values are described on page 117. Default value is "sound_click_low".
audio.tml_key_press	string	rwxr-	The sound that the terminal produces when “a TML key,“ that is, a button shown on the terminal screen is selected. Possible values are described on page 117. Default value is "sound_click_midtone".
audio.volume	integer	rwxr-	The variable whose value defines the volume (loudness) of sounds for all events. When set to 0, the sounds are off. The maximum possible value is 100. Default value is 100.

Variables used by card parsers

The section describes variables that can be used by card parsers during transaction processing.

Card-related variables

These variables are used to store card-related data read by parsers.

Name	Type	Perm	Description
card.cardholder_name	string	rwxrw	Cardholder name.
card.effective_date	date	rwxrw	Card effective date.
card.expiry_date	date	rwxrw	Card expiration date.
card.input_type	integer	rwxrw	Specifies how card details are entered: 1 – magnetic card is swiped 2 – card details are read from ICC EMV 3 – card details are entered manually using terminal keyboard. When parser is called with the command "read_data", this variable should be set to 1. If the parser is called with the command "risk_mgmt", the variable value should be checked, and if it is 3, card number and expiration date should be verified and card.pan should be filled.
card.issue_number	integer	rwxrw	Card issue number.
card.issuer_name	string	rwxrw	Card issuer name.
card.pan	string	rwxrw	Primary Account Number in ASCII.
card.scheme	string	rwxrw	Card scheme.

Variables used by “icc_emv” parser

These variables are used to store the “icc_emv” card parser-related data. Refer to *EMV Specification Book 3* for explanation of ICC EMV terms.

Name	Type	Perm	Description
card.emv.aac	opaque	rwxrw	AAC – **A**pplication **A**uthentication **C**ryptogram generated during the risk management (see “risk_mgmt” on page 57).
card.emv.aid	string	rwxrw	Application Identifier.
card.emv.aip	integer	rwxrw	Application Interchange Profile.
card.emv.app_pan_seq	integer	rwxrw	Application PAN Sequence Number.
card.emv.arqc	opaque	rwxrw	Authorisation Request Cryptogram generated during the risk management (see “risk_mgmt” on page 57).
card.emv.atc	integer	rwxrw	Application Transaction Counter.
card.emv.auc	integer	rwxrw	Application Usage Control.
card.emv.cvmr	opaque	rwxrw	Cardholder Verification Method Result.
card.emv.iac_default	opaque	rwxrw	Issuer Action Code (default).
card.emv.iac_denial	opaque	rwxrw	Issuer Action Code (denial).
card.emv.iac_online	opaque	rwxrw	Issuer Action Code (online).
card.emv.iad	opaque	rwxrw	Issuer Application Data.
card.emv.iso_track2	opaque	rwxrw	Contents of the Track 2 on the card.
card.emv.last_attempt	integer	rwxrw	If set to 1, indicates the last PIN entry attempt.
card.emv.signature	integer	rwxrw	If non-zero, the customer’s signature should be verified offline, using the printed receipt.

Name	Type	Perm	Description
card.emv.tc	opaque	rwxrw	TC – Transaction Certificate generated during the risk management (see "risk_mgmt" on page 57).
card.emv.tvr	opaque	rwxrw	Terminal Verification Results.
card.emv.unumber	integer	rwxrw	Unpredictable Number.

Smart-card PIN-related variables

These variables are used to store the data required for online validation of a smart card holder's PIN (**P**ersonal **I**dentification **N**umber).

Name	Type	Perm	Description
card.pin	string	rwxr-	A cryptogram containing the PIN entered by a smart card holder.
card.pin.length	integer	rwxr-	The number of characters in the entered PIN.
card.pin.smid	string	rwxr-	SMID[r] for PIN's DUKPT[s].
card.pin.array	integer	rwxrw	This variable is used to select the key array for DUKPT and 3DES encryption.

Variables used by "mag" parser

These variables are used to store the "mag" card parser-related data.

Name	Type	Perm	Description
card.mag.iso1_track	string	rwxrw	ISO 1 track as read from the card.
card.mag.iso2_track	string	rwxrw	ISO 2 track as read from the card.
card.mag.iso3_track	string	rwxrw	ISO 3 track as read from the card.
card.mag.service_code	string	rwxrw	The service code read from track two of the magnetic card.

Card parsers-related variables

These variables are used to store the card parser-related data.

Name	Type	Perm	Description
card.parser.type	string	rwxrw	Specifies the card parser used: "mag" or "icc_emv".
card.parser.reject_reason	string	rwxrw	Reason of the transaction rejection.
card.parser.verdict	string	rwxrw	Risk management verdict: "online", "offline", or "reject".
card.parser.cvm	string	rwxrw	Cardholder Verification Method.
card.parser.cvr	string	rwxrw	Cardholder Verification Result.

Variables for managing card parsers configuration updates

This section describes predefined TML variables used to manage updates of configuration data in use by the terminal ICC EMV and magnetic card parsers and to check if the versions of these data are up-to-date and correspond to the most recent

[r] SMID is an internal Ingenico's term referring to a block of data which acts as *cryptographic salt* for the PIN entered by the smart card holder.
To secure PIN transmission, the salt, normally representing a random string of characters, is added to the entered PIN. One of cryptographic algorithms is then applied to the resulting string prior to its transmission (to other application, device, etc.).

[s] DUKPT – **D**erived **U**nique **K**ey **P**er **T**ransaction – is a key management scheme in which for every transaction, a unique key is used which is derived from a fixed key.

ones available at the server. The values of most of these variables are set by the server[t].

Name	Type	Perm	Description
`cfgm.emv.timestamp`	integer	`rwxr-`	The variable used to keep track of the version of configuration data in use by the terminal ICC EMV card parser. Default value is `0`.
`cfgm.mag.timestamp`	integer	`rwxr-`	The variable used to keep track of the version of configuration data in use by the terminal magnetic card parser. Default value is `0`.
`cfgm.scan.interval`	integer	`rwx--`	The minimum amount of time in seconds between subsequent configuration requests sent to Incendo Online Gateway when the connection between the terminal and Incendo Online Gateway is active. Default value is `1`.

Variables related to the COM connection

This section describes predefined TML variables related to transmission of data via the terminal's COM (RS232) port.

Name	Type	Perm	Description
`com.data_size`	string	`rwxr-`	COM Data Size – an asynchronous connection parameter defining how many bits of data there are in one byte. Possible values are `"5"`, `"6"`, `"7"`, or `"8"`. Default value is `"8"`.
`com.name`	string	`rwxr-`	COM Name – the name of the terminal's COM port used for data transfer. Possible values are `"COM1"` or `"COM2"`. Default value is `"COM1"`.
`com.parity`	string	`rwxr-`	COM Parity – an asynchronous connection parameter – one of the parameters used to control the data transfer via a COM port. Possible values are `"none"`, `"odd"`, or `"even"`. Default value is `"none"`.
`com.speed`	string	`rwxr-`	COM Speed – an asynchronous connection parameter defining the speed of data transfer in bits per second. Possible values are `"115200"`, `"57600"`, or `"9600"`. Default value is `"115200"`.

[t] When the server sends to the terminal the card parsers configuration data, it also sets the values of the `cfgm.emv.timestamp` and `cfgm.mag.timestamp` variables so that they contain the information about the versions of configuration data being sent. Each next time the terminal requests the updates of configuration data from the server, it informs the server about the versions of data being used by sending to the server the variables' values. The server checks the "version numbers" received from the terminal against those of the current versions and either sends to the terminal the updated versions of the requested resources or replies that the resources have not changed.

Name	Type	Perm	Description
com.stop_bits	string	rwxr-	COM Stop Bits – an asynchronous connection parameter – one of the parameters used to control the data transfer via a COM port. Possible values are "1" or "2". Default value is "1".

Error handling variables

This section describes predefined TML variables related to error handling. General discussion of these variables can be found in "Handling incorrect user input" on page 30 and "Error handling" on page 31.

Name	Type	Perm	Description
err.baddata_reason	string	rwxrw	A reason returned by the terminal if the data entered by the user is invalid. See "<baddata>" on page 53.
err.code.high	integer	rwxr-	Numeric code of the last error detected by Incendo Online MicroBrowser. The value of this variable is set by the main module of Incendo Online MicroBrowser. Error codes are listed and described in Appendix A on page 142
err.code.low	integer	rwxr-	The numeric code of the 'previous' , that is, the last but one error detected by Incendo Online MicroBrowser. The value of this variable is set by the main module of Incendo Online MicroBrowser. Error codes are listed and described in Appendix A on page 142.
err.description	string	rwxrw	A brief textual description of the last error occurred in the system. The value of this variable is set by the module in which the error occurred.

GPRS-related variables

This section describes predefined TML variables used to store the settings for modem connection over GPRS (**G**eneral **P**acket **R**adio **S**ervice) communications channels.

Name	Type	Perm	Description
gprs.apn	string	rwxr-	APN – **A**ccess **P**oint **N**ame. Default value is "internet.msk".
gprs.gsm_pin1	string	rwxr-	GSM PIN1 – the **G**lobal **S**ystem for **M**obile Communications **P**ersonal **I**dentification **N**umber that the terminal uses to prove its identity when trying to establish the connection with mobile (GSM/GPRS) communications service provider. Default value is "0000".
gprs.gsm_puk1	string	rwxr-	GSM PUK1 – **P**ersonal **U**nblocking **K**ey – numeric code used to unblock a SIM card after incorrect entry of GSM PIN1. Default value is "00000000".

Name	Type	Perm	Description
gprs.selection_to	integer	rwxr-	Selection Timeout – the amount of time in seconds within which the terminal is waiting for registration in mobile communications service provider's network. Default value is 60.

IP-related variables

This section describes predefined IP (**I**nternet **P**rotocol)-related TML variables.

Name	Type	Perm	Description
ip.default_gateway	string	rwxr-	Default Gateway IP – an IP address of the gateway, a router or a host computer, that routes TCP packets from the terminal to the Internet and, hence, to Incendo Online Gateway. Default value is "192.168.0.1".
ip.dns1	string	rwxr-	IP address of a DNS – **D**omain **N**ame **S**erver – a server that translates hostnames (domain names) into IP addresses. Default value is "0.0.0.0".
ip.dns2	string	rwxr-	IP address of an alternative DNS – **D**omain **N**ame **S**erver – a server that translates hostnames (domain names) into IP addresses. Default value is "0.0.0.0".
ip.local_addr	string	rwxr-	The variable used to store the terminal's IP address which the terminal receives from the server. Default value is "0.0.0.0".
ip.media	string	rwxr-	Media Type – the type of the communications channel used by the terminal for data transmission. Possible values are "com", "dsl", "ethernet", "gprs", or "modem". Default value is "com".
ip.net_conn_timeout	integer	rwxr-	The amount of time in seconds within which the terminal is waiting for the receipt of the network settings (for example, an IP address) from the server. Default value is 120.
ip.net_mask	string	rwxr-	Net Mask – the net mask of the network to which the terminal is connected. Default value is "255.255.255.0".
ip.persistent	string	rwxr-	Persistent Connection – parameter that defines whether or not the connection with Internet service provider's (ISP) server should be kept active after the terminal disconnects from Incendo Online Gateway. Possible values are "yes" or "no". Default value is "yes".
ip.so_timeout	integer	rwxr-	Socket Timeout – the amount of time in seconds within which the terminal is waiting for the receipt of data from the host. Default value is 30.

Name	Type	Perm	Description
`ip.static_addr`	string	`rwxr-`	Static IP Address – the static IP address of the terminal; the one used by the terminal when the value of the `ip.term_ip` variable is set to `"static"`. Default value is `"192.168.0.10"`.
`ip.term_ip`	string	`rwxr-`	Defines which of the IP addresses the terminal uses – the *dynamic* IP address received from the server or the *static* IP address defined by the value of the variable `ip.static_addr`. Possible values are `"dynamic"` or `"static"`. Default value is `"dynamic"`.

Incendo Online MicroBrowser-related variables

This section describes predefined TML variables related to Incendo Online MicroBrowser and various aspects of its behaviour including its interaction with other terminal applications, terminal memory management, etc.

Name	Type	Perm	Description
`oebr.appversion`	string	`r-xr-`	MicroBrowser version as seen by the terminal OS.
`oebr.backlight.off_strength`	integer	`rwxr-`	The variable whose value defines how bright the terminal screen is when the backlight is off. The higher the value, the brighter the screen is. When set to `100`, the screen is as bright as when the backlight is on. Default value is `0`.
`oebr.backlight.timeout`	integer	`rwxr-`	The amount of time in seconds within which the terminal screen backlight stays on after a user performs some action with the terminal (for example, presses a key on the keypad, etc.). When this time expires, the backlight goes off, and stays off until the user does something with the terminal again (for example, presses another key, etc.). Default value is `600`.

Name	Type	Perm	Description
`oebr.cache_update_policy`	string	`rwx--`	Cache update policy defines how the resources cached in the terminal memory (TML pages, images, etc.) are updated. Possible values are `"all"` or `"expired"`. If `oebr.cache_update_policy="all"`, all the resources cached in the terminal are requested from the server each time the terminal connects to Incendo Online Gateway. If `oebr.cache_update_policy="expired"`, out of all the resources cached in the terminal only those that have expired are requested from the server when the terminal goes online. If the connection with Incendo Online Gateway is initiated by a call of the `connect_to_server` C function, the value of this variable does not matter: all the resources cached in the terminal are requested from the server. Default value is `"expired"`.
`oebr.cache.storage`	integer	`rwxr-`	The size of terminal memory cache (in kilobytes) reserved for storing HTTP/TML resources (TML pages, image and CSS files, etc.). This variable is used to limit the total size of all HTTP/TML resources downloaded by the terminal from the Application Server and cached in the terminal memory. The limit defined by this variable is checked only when 'new' resources are being downloaded by the terminal. That is, this limit is not checked when the resources that have once been downloaded are updated. When 75% of the limit set by this variable is reached, Incendo Online MicroBrowser will try to remove from the terminal cache all TML pages with the `cache` attribute set to `deny`. As soon as the limit defined by this variable is reached, no new resource will be downloaded. It's worth mentioning that the number of files stored in HTTP cache can not exceed 192. This number is fixed and can not be controlled via a TML application. Default values: `256` for Ingenico 5100 terminals and `1024` for Ingenico 8550 terminals.

Name	Type	Perm	Description
`oebr.connection.endstate`	string	`rwxrw`	Variable whose value specifies what should be the state of the connection between the terminal/Incendo Online MicroBrowser and Incendo Online Gateway host/ Incendo Online Gateway after Incendo Online MicroBrowser and Incendo Online Gateway completed the data exchange (for example, Incendo Online MicroBrowser finished downloading a TML resource, submitting the data to Application Server, etc.). Possible values are: `"down"`, `"up"` or `"connected"`. The values have the same meaning as in the case of the `oebr.connection.state` variable (see description of this variable later in this table). Default value is `"up"` which means that Incendo Online MicroBrowser should disconnect from Incendo Online Gateway (that is, close the socket connection), however, the terminal should stay connected to the network/Incendo Online Gateway host.
`oebr.connection.state`	string	`rwxrw`	Variable used to monitor and control the state of the connection between the terminal/Incendo Online MicroBrowser and Incendo Online Gateway host/Incendo Online Gateway. Possible values are: • `"down"` – the network interface is down which means that the terminal is in offline mode and is not trying to connect to the network • `"going_up"` – the terminal is in transition from `"down"` to `"up"` which means that the terminal is trying to connect to the network/Incendo Online Gateway host, for example, when dialing the telecommunications/Internet service provider up, etc. • `"up"` – the network interface is up which means that the terminal has connected to the network/Open Estate Gateway host and the applications – Incendo Online MicroBrowser and Incendo Online Gateway – are ready to start establishing the connection between each other (the so-called socket connection) • `"connecting"` – the terminal/Incendo Online MicroBrowser is in transition from `"up"` to `"connected"` which means that the terminal has already connected to Incendo Online Gateway host and Incendo Online MicroBrowser is trying to establish the socket connection with Incendo Online Gateway • `"connected"` – Incendo Online

Name	Type	Perm	Description
			MicroBrowser has connected to Incendo Online Gateway and the applications are ready to start exchanging the data • `"error"` – connection error occurred, that is, the terminal/Incendo Online MicroBrowser failed to connect to Incendo Online Gateway host/Incendo Online Gateway. The value of this variable may be set by a TML application and Incendo Online MicroBrowser. By changing the value of this variable a TML application tells Incendo Online MicroBrowser which connection state should be achieved. Incendo Online MicroBrowser, in its turn, uses this variable to reflect the current connection state of the terminal. A TML application can then check the variable value and take the actions appropriate to the current connection state. This is how all this may work: A TML application sets the `oebr.connection.state` to `"connected"`. By doing so it instructs Incendo Online MicroBrowser to connect to Incendo Online Gateway (say, for submitting the payment transaction data to some transaction authorisation service). Incendo Online MicroBrowser detects the variable value change, reads the value and starts trying to achieve the specified state. As soon as the connection state changes, Incendo Online MicroBrowser updates the variable value. A TML application can then check the variable value to find out what the current connection state is and act accordingly. Out of all the possible values, a TML application can set the `oebr.connection.state` to `"down"`, `"up"` or `"connected"`.
`oebr.connect.pool_off`	string	`rwxrw`	Defines whether or not all transactions performed offline and currently stored in the terminal memory should be sent to Incendo Online Gateway when the connect_to_server C function is called. Possible values are `"yes"` or `"no"`. Default value is "yes".

Name	Type	Perm	Description
`oebr.connect.sync_cache`	string	`rwxrw`	Defines whether or not all the resources cached in the terminal memory (TML pages, images, etc.) should be requested from Incendo Online Gateway when the `connect_to_server` C function is called. Possible values are `"yes"` or `"no"`. Default value is `"yes"`.
`oebr.connect.sync_config`	string	`rwxrw`	Defines whether or not the configuration data in use by ICC EMV and magnetic card parsers should be requested from Incendo Online Gateway when the `connect_to_server` C function is called. Possible values are `"yes"` or `"no"`. Default value is `"yes"`. See also description of `cfgm.scan.interval`.
`oebr.current_uri`	string	`rwxr-`	Current URI. The value of this variable is updated each time the application switches onto the next screen.
`oebr.econn`	integer	`rwxrw`	Reason of data submitting failure; filled by the HTTP client.
`oebr.indicators`	string	`rwxr-`	Defines the positions of the indicators on the terminal display. Four different indicators can be present showing: • radio signal strength (for terminal models supporting wireless communications) • main terminal battery charge • external power supply connection (the indicator appears when the terminal is placed into the cradle and the terminal battery is being charged) • secondary terminal battery charge Possible values are: • `"none"` – the indicators are not shown. • `"default"` – the radio signal strength indicator is in the top left corner; all the rest of the indicators are in the top right corner. • `"top-left"` – all indicators are aligned along a *horisontal* line which is placed into the top left corner of the screen; other possibilities for horisontally arranged indicators are: `"top-middle"`, `"top-right"`, `"bottom-left"`, `"bottom-middle"`, or `"bottom-right"`. • `"left-top"` – all indicators are aligned along a *vertical* line which is placed into the top left corner of the screen; other possibilities for vertically arranged indicators are: `"left-middle"`, `"left-bottom"`, `"right-top"`, `"right-middle"`, or `"right-bottom"`. Default value is `"default"`.

Name	Type	Perm	Description
`oebr.languages`	string	`rwxrw`	The list of supported languages. Possible values: a list composed of the `"english"`, `"french"` and `"spanish"` where the items are separated with semicolons (`;`). The first item in the list defines the 'default' language. Default value is `"english;french;spanish"`. For more information, see "Controlling multilingual input support" on page 38.
`oebr.last_connection_dt`	date	`rwxrw`	Last date and time when the terminal connected to the Incendo Online Gateway host.
`oebr.offline.size`	integer	`rwxr-`	The total size (in kilobytes) of all offline posts – postponed terminal transactions – currently stored in the terminal memory. See also description of `oebr.offline.storage`.
`oebr.offline.storage`	integer	`rwxr-`	The size of terminal memory (in kilobytes) reserved for storing offline posts – postponed terminal transactions. This variable is used to limit the amount of terminal memory available for storing offline posts. As soon as the total size of postponed transactions in the terminal memory reaches the limit defined by the value of this parameter, the terminal goes online, sends the postponed transactions to Incendo Online Gateway, and then deletes them from the terminal memory. Default value is `64`. See also description of `oebr.offline.size`.
`oebr.pin_init`	string	`rwx--`	Security Key to be sent to Incendo Online Gateway during initialisation of the terminal. Default value is `"100"`.
`oebr.post_id`	integer	`rwxrw`	Identifier of the postponed terminal transaction to be cancelled.
`oebr.posts_number`	integer	`rwxr-`	Total number of postponed terminal transactions currently stored in the terminal memory.
`oebr.posts_number_tmp`	integer	`rwxr-`	Temporary offline posts count.

Name	Type	Perm	Description
oebr.posts_print_mode	string	rwxr-	Defines what information should be included when the contents of offline posts – postponed terminal transactions – are printed out. Possible values are: • "header" – the date and time, ID and destination URI is printed for each offline post. • "name" – in addition to what is printed when the value is set to "header", the list of variables contained in the post is also included. • "value" – in addition to what is printed when the value is set to "name", the values of all variables contained in the post are also included. Default value is "header". **Important:** For security reasons, the names of variables are not to be exposed to average terminal users. So setting this variable's value to anything other than the "header" is acceptable only for the purpose of internal application testing and, generally, should be avoided.
oebr.prev_screen	string	rwxr-	The URI of the last rendered screen.
oebr.run_on_reboot	string	rwx--	Run on Reboot – parameter whose value defines whether or not Incendo Online MicroBrowser should start automatically after the terminal is rebooted. Possible values are "yes" or "no". Default value is "yes".
oebr.run_on_reboot_str	string	rwx--	Auxiliary variable used to store intermediate results of working with the Run on Reboot parameter. Possible values are "yes" or "no". Default value is "yes".
oebr.start_page	string	rwxrw	The URI of Incendo Online MicroBrowser start page. Default value is "/" which is the URI of the default TML page on the Application Server. **Note:** If the terminal has not been initialised, the value of this variable is ignored: when Incendo Online MicroBrowser is started it loads emb://embedded.tml.
oebr.submit_mode	string	rwxrw	The variable defining one of the two possible ways of submitting the transaction data. Possible values are "online" (the data is sent to Incendo Online Gateway) or "offline" (the data is (temporarily) stored in the terminal memory). Default value is "online".
oebr.supervisor_passwd	string	rwx--	Supervisor Password – the terminal administrator's password. Default value is "123".

Name	Type	Perm	Description
oebr.time_zone	string	rwxr-	Time zone of the terminal location. Default value is "0". See also "Managing time information: standard local time, daylight saving time and GMT" on page 31.
oebr.transid	integer	rwxrw	Unique numeric transaction identifier which is set for each transaction by means of the following construction: <setvar name="oebr.transid" lo="tmlvar:oebr.unique_id"/> Default value is 1.
oebr.unique_id	integer	r-xr-	Variable used for generating a unique positive integer number. Each reference of this variable in TML code updates its value. To be used in constructions such as: <setvar name="oebr.transid" lo="tmlvar:oebr.unique_id"/> to make sure that, for example, a transaction identifier is a unique one.
oebr.version	string	r-xr-	Incendo Online MicroBrowser version in use by the terminal.

Variables related to working with TML logs

These variables are used to store the information necessary for working with TML logs.

Please note that almost all variables listed in this section are system log-related and can only be accessed from the Embedded TML Application (see the *Permissions* column).

The only variable that can be accessed by TML applications other than the Embedded Application is the oebr.log_id.

Name	Type	Perm	Description
oebr.log_descr	string	rwx--	Error description. The value of this variable is set by Incendo Online MicroBrowser.
oebr.log_id	string	rwxrw	This variable defines the name of the log that is to be cleared when the clear_log function is called. The value of this variable should be properly set prior to performing the function call.
oebr.log_module	string	rwx--	The name of the software module that originated the error. The value of this variable is set by Incendo Online MicroBrowser.
oebr.log_severity	string	rwx--	Error severity, e.g. FATAL, ERROR, WARNING, etc. The value of this variable is set by Incendo Online MicroBrowser.
oebr.log_size	integer	rwx--	The total size (in bytes) of all log files being used in the system. The value of this variable is set by Incendo Online MicroBrowser.
oebr.log_size_limit	integer	rwx--	This variable sets the maximum possible size (in kilobytes) of all log files. The value of this variable is set by Incendo Online MicroBrowser. Default value is 128.

oebr.ulog_cntr	integer	rwx--	Auxiliary log counter. The value of this variable is set by Incendo Online MicroBrowser.

Variables related to working with GMA events

These variables are used to define which of the GMA[u] events Incendo Online MicroBrowser should be informed of as well as to store the information related to such events.

For general discussion of GMA events and their processing in TML applications, see "Processing GMA events" on page 33.

Name	Type	Perm	Description
gma.event.subscribed	string	rwxrw	Variable containing the list of GMA events your want Incendo Online MicroBrowser to be informed of. In other words, this is the list of the events 'reported' by GMA that you are going to process in your TML application. The value of this variable can be a list composed of the "anykey", "mag", and "icc" where the items are separated with a semicolon (;) or an empty string (""). "anykey", "mag", and "icc" are the names of the events corresponding to pressing of certain keypad keys (normally – all keys with the exception of alphanumeric keys and the **menu** key), swiping of a magnetic card through the card reader, and insertion of a smart card into the terminal respectively. Default value is "",meaning that Incendo Online MicroBrowser is not subscribed to (i.e. not informed of) any of the GMA events.
gma.event.occured	string	rwxr-	The name of a GMA event that took place. Possible values: "menu", "anykey", "mag", or "icc" where the "menu" corresponds to the **menu** key press and the rest of the names have the same meaning as in the case of the variable gma.event.subscribed. Default value is "menu".
gma.event.key.pressed	string	rwxr-	Variable whose value tells which of the keypad keys was pressed (if Incendo Online MicroBrowser is subscribed to key presses and one of the corresponding keys in fact was just pressed). Possible values: "enter", "stop", "00", "cancel", "sys", "lfeed", "f1", "f2", "f3", "f4", "f5", "f6", "f7", "f8", or "f9". All these values correspond to the keypad key names adopted in TML. Default value is "".

[u] GMA – **G**lobal **M**aster **A**pplication.

gma.event.icc. card_answer	opaque	rwx--	If the event with the name "icc" takes place, this variable will contain the smart card answer. The variable is only for internal use within Incendo Online MicroBrowser. You *must not* use this variable in your TML applications.

Variables related to data exchange with third-party applications

These variables are used to store the information necessary for data exchange with third-party UNICAPT32 applications running in the terminal.

Name	Type	Perm	Description
oebr.3rdparty.app_name	string	rwxrw	The name of an application (assigned to it in IngeDev) to which the data are to be sent. Default value is "GB000400_Ima".
oebr.3rdparty.timeout	integer	rwxrw	Timeout in tens of milliseconds. This is an auxiliary variable whose value does not affect the data exchange per se. Its purpose is to pass to a third-party application a recommended amount of time within which the third-party application may stay idle. It is assumed that if there is no user activity in the third-party application and the timeout elapses, the control should be passed back to Incendo Online MicroBrowser. The third-party application may just ignore this variable (as well as take it into account). Default value is 0.
oebr.3rdparty.var_list	string	rwxrw	The list of variables to be sent to an application defined by oebr.3rdparty.app_name. Names of variables in the list should be separated with a semicolon (;). Default value is "oebr.3rdparty.app_name; oebr.3rdparty.timeout; card.emv.aac; oebr.last_connection_dt".

Incendo Online Gateway variables

This section describes predefined TML variables related to Incendo Online Gateway.

Name	Type	Perm	Description
oegw.certificate	string	rwx--	Incendo Online Gateway's X.509 public key certificate. By default, this variable stores the default Incendo Online Gateway's certificate that comes together with Incendo Online.
oegw.init_port	integer	rwxr-	Incendo Online Gateway Initialisation Service's port number. Default value is 61001.
oegw.init_resp_timeout	integer	rwxr-	Incendo Online Gateway Initialisation Service response timeout in seconds. Default value is 25.

Name	Type	Perm	Description
oegw.ip	string	rwxr-	Incendo Online Gateway's hostname (domain name) or the IP address of the computer on which Incendo Online Gateway runs. The value of this variable must be the same as the Common Name (CN) of certificate owner in Incendo Online Gateway's X.509 public key certificate. Default value is "127.0.0.1".
oegw.ip.resolved	string	rwxr-	Incendo Online Gateway's IP address received from a DNS (Domain Name Server) as a result of translation of Incendo Online Gateway's hostname (domain name) into an IP address. Default value is "unknown".
oegw.port	integer	rwxr-	Incendo Online Gateway data port number. Default value is 61000.
oegw.resp_timeout	integer	rwxr-	Incendo Online Gateway response timeout in seconds. Default value is 25.

Variables related to payment processing

This section describes predefined TML variables related to payment processing.

Name	Type	Perm	Description
payment.amount	integer	rwxrw	Transaction amount.
payment.amount_other	integer	rwxrw	Cashback amount.
payment.auth_code	string	rwxrw	Contains Authorisation Code. Used for ICC online authorisation, see auth parser command on page 60.
payment.auth_resp_code	integer	rwxrw	Contains Authorisation Response Code. Used for ICC online authorisation, see auth parser command on page 60. Default value is 0.
payment.emv.arpc	opaque	rwxrw	Authorisation Response Cryptogram.
payment.emv. issuer_auth	opaque	rwxrw	Used during online authorisation, see auth parser command on page 60. Contains Issuer Authentication Data.
payment.emv. issuer_script1	opaque	rwxrw	Used during online ICC authorisation, see auth parser command on page 60. Contains an issuer's script which should be executed on the ICC EMV to update the card data, change application or block the card.
payment.emv. issuer_script2	opaque	rwxrw	Contains an issuer's script which should be executed on the ICC EMV to update the card data, change application or block the card.
payment.emv. issuer_script_results	opaque	rwxrw	Used during online authorisation, see auth parser command on page 60. Contains the result returned by ICC EMV after executing an issuer's script.
payment. merchant_number	integer	rwxr-	Merchant's phone number.

Name	Type	Perm	Description
payment.trans_type	string	rwxrw	This variable is used by the card parsers. To perform a transaction, it should be set to one the generic transaction types: either debit, cash, cashback, reversal, or credit. Additionally, in special cases, it can be expanded with a numeric value. For more information, refer to the *Technical Note 04. EMV Transaction Types – using payment.trans_type variable*.
payment.txn_result	integer	rwxrw	Indicates the transaction authorisation result: 1 – transaction approved, 0 – transaction declined.

Point-to-Point Protocol (PPP) connection variables

This section describes predefined TML variables related to PPP connection.

Name	Type	Perm	Description
ppp.authtype	string	rwxr-	PPP Authentication Type – the type of authentication used to check the terminal's identity when the terminal tries to establish the PPP connection with a server using the PSTN[v] modem. Possible values are "PAP" or "CHAP". Default value is "PAP".
ppp.conn_timeout	integer	rwxr-	Connection Timeout – the amount of time in seconds within which the terminal is trying to establish the connection with the service provider's server via the PSTN modem. Default value is 120.
ppp.login	string	rwx--	Login – the name that the terminal uses to identify itself when connecting to the Internet service provider's server via a modem. Default value is "ingenico_oe".
ppp.password	string	rwx--	The password that the terminal provides to prove its identity when connecting to the Internet service provider's server via a modem. Default value is "1234567890".
ppp.phone	string	rwxr-	The phone number that the terminal's PSTN modem dials to establish the connection with the service provider's server. Default value is "908450885336".
ppp.retries	integer	rwxr-	The number of times the terminal is trying to establish the connection with the service provider's server via the PSTN modem. Default value is 3.

Terminal variables

This section describes predefined TML variables related to a terminal.

[v] **P**ublic **S**witched **T**elephone **N**etwork

Name	Type	Perm	Description
terminal.certificate	string	rwx--	Terminal's X.509 public key certificate corresponding to the private key stored in the variable terminal.privkey. By default, this variable stores the default terminal's public key certificate that comes together with Incendo Online.
terminal.datetime	date	rwxrw	System local date and time. See also "Managing time information: standard local time, daylight saving time and GMT" on page 31.
terminal. datetime_start	date	r-xr-	An auxiliary variable used for checking whether or not the terminal date and time have been set. Default value is 2006/01/01.
terminal.itid	string	rwxr-	ITID – **I**ngenico **T**erminal **I**dentifier. Default value is "100".
terminal.model	integer	r-xr-	The model of the terminal, for example, 8550, 5100, and so on.
terminal.os	string	r-xr-	The version of terminal operating system.
terminal.part_number	string	r-xr-	Terminal part number.
terminal.password	string	rwx--	The password used for authentication of a terminal by Incendo Online Gateway. This password is received from Incendo Online Gateway as a result of terminal initialisation.
terminal. pinpad_present	integer	r-x--	The variable whose value defines whether or not the terminal has a PIN pad attached to it. Possible values are 0 (PIN pad is missing) or 1 (PIN pad is present). Default value is 0.
terminal.privkey	string	rwx--	Terminal's RSA[w] private (asymmetric cryptographic) key. By default, this variable stores the default terminal's RSA private key that comes together with Incendo Online.
terminal.serial_number	string	r-xr-	Terminal serial number.
terminal. serial_number_default	string	r-x--	A serial number assigned to a development terminal. This is an auxiliary variable used for checking whether or not the terminal is a development one. Default value is "123456789012".
terminal.sn_check	integer	rwx--	Defines whether or not the terminal serial number should be checked to make sure that the terminal is not a development one. Possible values are 0 (serial number should not be checked) or 1 (serial number should be checked). Default value is 1.

[w] RSA – a short for **R**ivest-**S**hamir-**A**dleman encryption system: the patented public key encryption algorithm, introduced by Ronald Rivest, Adi Shamir, and Leonard Adleman.

Bar code scanner (imager)-related variables

This section describes predefined TML variables used when working with bar code scanner (imager).

Name	Type	Perm	Description
imager.aim_id	string	rwxrw	AIM Code Character – the second character in the symbology identifier string according to *International Technical Specification – Symbology Identifiers* by AIM Inc.[x]
imager.aim_modifier	string	rwxrw	AIM Modifier Character – the third character in the symbology identifier string according to *International Technical Specification – Symbology Identifiers* by AIM Inc. (see footnote [a] on page 138).
imager.code_id	string	rwxrw	Single-character symbology identifier according to the coding system used by Hand Held Products, Inc., the company manufacturing bar code scanners that are used in Ingenico terminals.
imager.data	string	rwxrw	The data contained in a bar code.

Auxiliary variables

This section describes auxiliary variables – the ones that temporarily store various intermediate results or information.

Name	Type	Perm	Description
passwd	string	rw---	The variable that stores intermediate results of working with the Supervisor Password parameter.
screen_after_call	string	rw---	The variable that stores the URI of the screen that should be processed after the call of a function such as Clear HTTP Cache, Disconnect from OEGW, and so on.

[x] According to *International Technical Specification – Symbology Identifiers* by AIM Inc. (the Association for Automatic Identification and Data Capture Technologies), the bar code reading equipment should prefix the symbology identifier to the data contained in a bar code symbol. The symbology identifier is a three character string having the following structure:

```
]cm
```

where

-] is the character assigned to ASCII value 93 in the United States ASCII character set in accordance with ISO 646, which represents the symbology identifier flag character. The flag character indicates to the host receiving the data that it and the two following characters are the symbology identifier characters.
- c represents the code character, which indicates to the host the bar code symbology of the symbol which has been read. For example, the code characters "A", "C", and "E" correspond to the Code 39, Code 128, and EAN/UPC symbologies respectively.
- m represents the modifier character for the symbology defined by the code character. The modifier character indicates to the host the mode in which the symbology is used. The modifier characters are symbology-specific.

For example, for the Code 128 symbology, the possible modifier characters are "0", "1", "2", and "4", so the overall identifier for this symbology may look like "]C0", "]C1", "]C2", and "]C4".

For more information, refer to *International Technical Specification – Symbology Identifiers* and specifications for particular symbologies.

TML CSS

8

Due to the terminal hardware limitations TML CSS offers only the most essential capabilities of the full CSS 2.0 specification. CSS is well documented on the web and in various books. This chapter describes only the limitations and specifics of CSS introduced with TML.

TML CSS limitations

Syntax and general rules

Basic syntax

Only the following syntax is allowed in TML CSS:

```
selector {property 1: value 1; ... ; property N: value N;}
```

For example:

```
.c_bold { font-weight: bold;}
.c_large { font-size: large; text-align: left;}
.c_right { text-align: right;}
.c_left { text-align: left; }
.c_center { text-align: center;}
```

Only a single statement is allowed within a line. It is not allowed to reference selectors by `id` attribute.

Important: CSS selectors, properties and values are case-sensitive.

Pseudo-classes and pseudo-elements

Pseudo-classes and pseudo-selectors are not allowed in TML CSS.

Contextual selectors

Contextual selectors are not allowed in TML CSS.

Comments

Comments are not allowed in TML CSS.

Cascading order

Cascading order is the same as specified in CSS2 with the only exception that `!Important` statement is not allowed.

Style directive

Style XHTML element is not allowed in TML.

TML CSS Properties

Background

Supported Properties	Description
`background-color`	Numerical colour specification in #RRGGBB format. Note that in TML CSS you can use only upper case letters and hexadecimal colour codes, e.g. `#FFFF00`.

Border

Supported Properties	Description
`border-style`	`none`, `dotted`, `dashed`, `solid`, `double`

Classification

Classification properties are not supported in TML CSS.

Dimensions

Supported Properties	Description
height	npx or n% where n is a positive integer number specifying the height either as a number of pixels (absolute height) or as a number of percent of the screen height (relative height).
width	npx or n% where n is a positive integer number specifying the width either as a number of pixels (absolute width) or as a number of percent of the parent element's width (relative width).

Font

Supported Properties	Description
font-family	arial, courier, helvetica, sans-serif, times
font-size	small, medium, large
font-style	normal, italic
font-weight	normal, bold

Generated content

Generated content properties are not supported in TML CSS.

List and marker

List and marker properties are not supported in TML CSS. Unordered list items are preceded with asterisk (*).

Margin

Supported Properties	Description
margin	percentage, auto
margin-bottom	percentage, auto
margin-left	percentage, auto
margin-right	percentage, auto
margin-top	percentage, auto

Outlines

Outline properties are not supported in TML CSS.

Padding

Padding properties are not supported in TML CSS.

Positioning

Positioning properties are not supported in TML CSS.

Table

Table properties are not supported in TML CSS.

Text

Supported Properties	Description
color	Numerical colour specification in #RRGGBB format. Note that in TML CSS you can use only upper case letters and hexadecimal colour codes, e.g. #FFFF00.
text-align	left, right, center
text-decoration	none, underline
text-indent	percentage
vertical-align	top, bottom, middle, baseline
white-space	normal, pre

What can be controlled with styles

You can use the styles to control:

- Placement of elements on a screen or within its parts (for example, within the cells of a table) by means of margin- and/or alignment-related properties
- Absolute or relative size (height and width) of screen elements
- Appearance of the text (typeface, font size, weight, and so on)
- Colour on devices with colour capability
- The line styles for borders of the elements (solid, dashed, dotted, and so on)

Applying styles

Incendo Online MicroBrowser uses a default style sheet (`default.css`) when the processed TML document contains no reference to an external style sheet, or the specified external style sheet fails to be loaded.

Even if you specify no styles or only a few, the entire document is displayed using styles specified in the default style sheet. Styles defined in the default style sheet have the lowest precedence in the cascading order, so whenever you specify your own style it overrides the style set by the default style sheet.

The external CSS is loaded in the text format by the terminal alongside with the related TML document. If, for some reason the loading cannot be completed, MicroBrowser will use the default CSS.

Using the class attribute

It is useful to define a class containing the style properties you wish to apply to multiple elements, for example:

```
.c_left { text-align: left; }
```

Then, in your TML code you can apply the style by referencing it in the `class` attribute:

```
<p class="c_left">Signature:_____</p>
```

Using and elements

You can also apply the styles, defined in a style sheet to a sequence of elements by using the `<div>` element (see " " on page 64) or to a sequence of inline characters by using the `<span>` element (see "" on page 96). For example, let us say a style sheet contains the following:

```
div.bold_text {text-align: center; font-weight: bold}
span.normal_text {font-weight: normal}
```

Then, within the TML page, you can apply the style specified by the `<div>` element to a series of paragraphs and the style specified by the `<span>` element to a series of characters:

```
<div class="bold_text">
<p>All letters in this paragraph are bold</p>
<p>Some <span class="normal_text">normal weight</span>
letters.</p>
</div>
```

Incendo Online MicroBrowser Error Codes

A

This Appendix provides a listing of all the error codes generated by the Incendo Online MicroBrowser. The error codes are 5 digit negative numbers. The first three digits correspond to a specific component of the MicroBrowser.

Each section contains a table that lists the possible values of pre-defined variables `err.code.high` and `err.code.low` (the **Error Code** column). Descriptions (the **Description** column) correspond to the text strings that may appear in the system log and/or be part of the `err.description` variable value.

Note: for more information on error handling and error handling variables see "Error handling" on page 31 and "Error handling variables" on page 123.

The following list should help you to find a particular error:

(-321XX) CFGM errors

These errors are related to the operation of the Configuration Manager.

Error Code	Description
-32101	Unable to read timestamp value
-32102	Unable to prepare config request
-32103	Unable to parse server response
-32104	Unable to open file
-32105	Unable to write file
-32106	Unable to rename file
-32107	Out of memory
-32108	Unsupported TACDenial size
-32109	Unsupported TACOnline size
-32110	Unsupported TACDefault size
-32111	Unsupported TDOL size
-32112	Unsupported DDOL size
-32113	Unsupported AID size
-32114	Unsupported TRMD size
-32115	Unsupported RID size
-32116	Unsupported Key data size
-32117	Unsupported key exponent size
-32118	Unable to configure
-32119	Error in response
-32120	Unable to read file
-32121	Can't delete a non-present hotcard
-32122	Unable to set timestamp
-32123	Incorrect IIN range
-32124	Incorrect RSA Key of ICC config

(-322XX) FPM errors

Form Processing Module errors.

Error Code	Description
-32201	FPM unknown error
-32202	Invalid input type
-32203	Runtime error
-32204	Unable to generate TML post ID
-32206	Bad data entered
-32208	Unable to extract account number from PAN
-32209	Unable to read DUKPT key id
-32210	Unable to init FPM
-32211	Unable to send log to host
-32212	Peripheral error
-32213	SSA error

Error Code	Description
-32214	PID error
-32215	Screen re-drawing error
-32251	Wrong equal check
-32252	Incorrect, not equal
-32253	Wrong not equal check
-32254	Incorrect, equal
-32255	Wrong min check
-32256	Too short
-32257	Wrong max check
-32258	Too long
-32259	Value assign error
-32260	SH1 calculation error
-32261	Cast SH1 hash to Base64 error
-32262	Invalid date
-32263	Wrong %s check
-32264	Too big
-32265	Too small
-32266	Unknown parser
-32267	Card muted
-32268	Card error
-32269	Card removed
-32270	Invalid TML call sequence OR EMV card has been changed
-32271	Canceled by user
-32272	FPM timeout
-32273	Internal ssa error
-32274	No pinpad
-32275	PIN entry type is not supported

(-323XX) TML errors

These errors are related to TML parser operation.

Error Code	Description
-32300	Attribute redefined
-32301	Error getting string
-32302	String too long
-32303	Error getting int
-32304	Unexpected end of BTML
-32305	Element not closed
-32306	Screen too big
-32307	Attr/content violates TML scheme
-32308	Invalid attribute value
-32309	Invalid attribute

Error Code	Description
-32310	Required attribute missed
-32311	Fatal variable library error
-32312	Can not convert attribute value
-32313	Variable not found
-32314	Unknown element or element is not allowed here
-32315	Bad content
-32316	Cells in table overlaps
-32317	Screen filename too long
-32318	Can't update screen file
-32319	Runtime error parsing BTML
-32320	Invalid format specified
-32322	Invalid variable type
-32323	Logger error
-32324	Log record template is too long
-32325	Length parsing error
-32327	Expected end of BTML after reading last closing tag
-32330	Runtime error when parsing CSS
-32331	CSS file read error
-32332	HTTP error retrieving CSS
-32333	No/invalid CSS declaration found
-32334	CSS selector/declaration empty
-32335	CSS property/value unrecognized

(-324XX) HTTP_CLIENT errors

These errors are related to HTTP Client operation.

Error Code	Description
-32400	Internal error: not enough memory
-32401	Can't connect to OEGW
-32402	Resource identified by URI is unavailable
-32403	Unable to send data to OEGW
-32404	Unable to receive data from OEGW
-32405	Unsupported content type
-32406	Unable to update TML screen cache
-32407	Unable to store post
-32408	Unable to find offline post
-32409	Not enough free DFS space
-32410	BHTTP parser fails
-32411	Offline posts limit reached
-32412	Unable to create HTTP cache directory structure
-32413	Unable to install embedded TML application
-32414	Invalid TML received

Error Code	Description
-32415	Uri too long
-32416	Internal error: varlib failure
-32417	Unable to follow HTTP redirect
-32418	Cache cleanup error
-32419	Error processing cache updates
-32420	Internal file open error
-32421	Unable to prepare configuration data update request
-32422	Unable to update HTTP cache
-32423	Unable to update configuration data
-32424	Fatal HTTP cache error. All data is lost
-32425	Offline post processing error
-32426	Cache limit reached
-32427	Invalid ImageLib received
-32428	Duplicated PostID. Post with such ID already exists
-32429	Unable to compress offline posts storage
-32430	Offline posts storage corrupted
-32431	Duplicated screen or ID/IML image name
-32432	Screen file too long
-32433	Fatal: Unsupported cache metadata version

(-325XX) INITM errors

These errors are related to the functioning of the Initialization Manager.

Error Code	Description
-32501	Internal error.
-32502	Unable to operate with TML vars.
-32503	Unable to find transport channel.
-32504	Unable to connect to OE InitService.
-32505	Communication error during initialization.
-32506	Communication error or ITID/Security Key are invalid. Terminal can be locked on OEGW
-32507	Unable to close OE InitService connection.
-32508	Certificate is not confirmed.
-32509	Could not save configuration data.

(-326XX) MAIN_APP errors

These errors are related to the functioning of the main application.

Error Code	Description
-32600	OEBR initialization error.
-32601	Open peripheral channels failed.
-32602	Graphic system initialization failed.
-32603	OEBR not initialized.
-32604	File open error.
-32605	File create error.
-32606	File write error.
-32610	VARLIB error.
-32611	Indicators error.
-32612	RNM error.
-32613	FPM error.
-32614	Virtual keyboard widget error.
-32615	User Logger error.
-32616	HTTP client error.
-32617	Initialization module error.
-32618	Network media error.
-32619	OEBR update error.
-32625	Signature capturing not supported.
-32626	No next screen defined.
-32627	Page can not be opened.
-32628	Wrong URI.
-32629	<setvar> failed.
-32630	No baddata defined for tform screen.
-32631	Checkbox is associated with non-string variable.
-32632	Invalid econn attribute.
-32633	<strtemplate> failed.
-32634	Battery status check error.
-32635	Peripheral inquiry error.
-32636	Terminal screen backlight error.
-32637	Signature capture widget error.
-32638	Cannot calibrate the touchscreen. Try to repeat it more carefully.
-32639	Touchscreen not supported.
-32640	Memory allocation error.
-32641	Invalid TML-screen type.
-32642	Econn processing error.
-32643	OEBR timer processing error.
-32644	Unknown calc-function.
-32645	Wrong URI

(-327XX) CARD_PARSER errors

These errors are related to the functioning of the card parsers.

Error Code	Description
-32701	Check baddata
-32702	Key pressed
-32704	ICC timeout
-32705	Invalid card (track 1)
-32706	Invalid card (track 2)
-32707	Card track 1 is too big
-32708	Card track 2 is too big
-32710	Cannot get start date from card
-32711	Invalid start date format
-32712	Card is expired
-32713	Card is not effective yet
-32714	Invalid expiry date format
-32715	Cannot get service code from card
-32716	Invalid issue number position
-32717	Cannot get issue number
-32718	Unknown issue number
-32719	Unable to locate card number delimiter
-32720	Invalid card number (IIN part)
-32721	Card is inactive (IIN blocked)
-32722	Invalid card scheme
-32723	Card scheme is inactive
-32724	Unable to find card format by it's number
-32725	Invalid card number
-32726	Invalid card number (PUK part)
-32727	Invalid card number (LUHN part)
-32728	Cannot update configuration data element
-32729	Card scheme floor limit exceeded
-32734	Cardholder name have an invalid characters
-32735	Failed to find configuration data
-32736	Failed to find risk management policy for the card scheme
-32737	Offline transaction is not allowed
-32738	Internal error
-32739	TTL description is incorrect for card
-32740	IIN range expired
-32741	Discretionary data missing
-32742	Card found in Hot Card List
-32743	Hot Card List file missed. Please connect to OEGW to solve problem
-32744	Hot Card List file corrupted. Please connect to OEGW to solve problem
-32745	IIN table file missed. Please connect to OEGW to solve problem

Error Code	Description
-32746	IIN table file corrupted. Please connect to OEGW to solve problem
-32747	TTL table file missed. Please connect to OEGW to solve problem
-32748	TTL table file corrupted. Please connect to OEGW to solve problem
-32749	Card Scheme table file missed. Please connect to OEGW to solve problem
-32750	Card Scheme table file corrupted. Please connect to OEGW to solve problem
-32751	Risk Managment Verdict table file missed. Please connect to OEGW to solve problem
-32752	Risk Managment Verdict table file corrupted. Please connect to OEGW to solve problem
-32753	Risk Managment file missed. Please connect to OEGW to solve problem
-32754	Risk Managment file corrupted. Please connect to OEGW to solve problem
-32755	Invalid card number (incorrect PAN length)
-32756	Unable to get ISO tracks
-32757	Risk Managment failed to write a reject reason

(-328XX) ICC_EMV_PARSER errors

These errors are ICC EMV parser-related.

Error Code	Description
-32801	Check baddata
-32802	Key pressed
-32804	ICC timeout
-32805	Failed to add application selection criteria
-32806	Unable to start EMV payment module
-32807	Card is expired
-32808	Card is not effective yet
-32809	Card is blocked
-32810	Card is removed
-32811	Card is changed
-32812	Card is muted
-32813	Application is blocked
-32814	Application selection failed (no candidates)
-32815	Application selection failed
-32816	Cardholder confirmation during application selection is not supported
-32817	Application selection failed
-32818	Unable to create EMV transaction context
-32819	Unable to prepare EMV transaction
-32820	Data authentication failed
-32821	Cannot find CA RSA key

Error Code	Description
-32822	Internal buffer overflow
-32827	Wrong authorization code length
-32828	Wrong authorization response code length
-32829	Fall back to magnetic stripe
-32830	Cannot supply power to ICC
-32831	Application table file missed. Please connect to OEGW to solve problem
-32832	Card provider table file missed. Please connect to OEGW to solve problem
-32833	RSA key table file missed. Please connect to OEGW to solve problem
-32834	Application table file corrupted. Please connect to OEGW to solve problem
-32835	Card provider table file corrupted. Please connect to OEGW to solve problem
-32836	RSA key table file corrupted. Please connect to OEGW to solve problem
-32837	Cannot load configuration data for card provider
-32838	Cannot load configuration data for the application
-32839	Internal error
-32840	Smart card insertion detection failed
-32841	Unknown tag
-32842	inconsistent TML application
-32843	Reject transaction
-32844	unable to verify cardholder
-32845	Not Authorized
-32846	Service Not Allowed
-32847	PIN Limit Exceeded
-32848	Issuer Authentication
-32849	transaction validation failed
-32850	transaction action analysis failed
-32851	transaction completion failed
-32852	Failed to del application selection criteria
-32853	Issuer script is too long
-32854	Memory allocation error
-32855	VARLIB error
-32856	Set verdict error
-32857	CVM value error
-32858	SMC-remove error
-32859	AMG selection error
-32860	AMG transaction validation error
-32861	AMG action analyze error
-32862	AMG transaction completion error
-32863	Card holder verification error
-32864	EMV initialization error

Error Code	Description
-32865	EMV Select Data error
-32866	PAN consistency check failed
-32867	Invalid transaction type
-32868	Logical error within the card
-32869	Unable to get requested EMV data
-32870	Unable to store provided EMV data
-32871	The card has been removed and re-inserted

(-329XX) RNM errors

Error Code	Description
-32901	memory allocation error
-32902	initialization error
-32903	invalid object position
-32904	line width is not enough for object
-32905	wrong TML variable type associated with checkbox
-32906	VARLIB error
-32907	text width calculation error
-32908	object rendering error
-32909	valref read error
-32910	file operation error
-32911	canvas operation error
-32912	wrong variable type associated with date input
-32913	HTTP client retrieve resource error
-32914	screen binary data is corrupted - invalid CRC
-32915	screen loader error
-32916	error of setvar processing
-32917	Error of log processing
-32918	only 1bpp or 8bpp pictures are supported
-32919	8bpp pictures are not allowed for this screen
-32920	invalid image size
-32921	printer is out of paper
-32922	printing error
-32924	HTTP client fails to retrieve images
-32925	error of indicators rendering
-32926	HMI error
-32927	error of graphics context set
-32928	error of font set
-32929	error of text positioning
-32930	baddata initialization fails
-32931	'on focus' and 'main' images must have same sizes and color depths
-32932	logrec loading error

Error Code	Description
-32933	strtemplate loading error
-32934	table has not enough space for all columns
-32935	color text in monochrome screen is not allowed
-32936	TML variable associated with image has different from opaque type
-32937	list control value is not a string
-32938	runtime error.
-32939	Table headers and/or footers are too big in height. No place for rows. Header/footer freezing mode is switched OFF.

(-330XX) COMMON errors

Error Code	Description
-33000	Font file read error
-33001	Memory allocation error during font load
-33004	Font set operation error
-33005	Font define operation error
-33006	Font addition error

(-331XX) VARLIB errors

These errors are related to the processing of TML variables.

Error Code	Description
-33100	Memory allocation error.
-33101	File reading error.
-33102	File writing error.
-33103	Prohibited cast.
-33104	Incorrect scope.
-33106	Variable absent.
-33107	Overlapping is forbidden.
-33108	Operation is not permitted.
-33112	Invalid value assigned.
-33113	Invalid formatter.
-33114	Value formatting error.
-33115	Variant processing error.
-33116	Invalid date/time value, Month > 12 or Month < 1.
-33117	Invalid date/time value, Wrong day number.
-33118	Invalid date/time value, Hour > 24.
-33119	Invalid date/time value, Minutes > 59.
-33120	Invalid date/time value, Seconds > 59.
-33121	Invalid date/time value, Milliseconds > 999.
-33122	Invalid date/time value, minutes and seconds can't be bigger then 00 at midnight (24:00:00).

Error Code	Description
-33123	Invalid date/time value, time-zone.
-33124	psyDateTimeGet error.
-33125	Load URI map error.
-33126	Setvar operand processing error.
-33127	Add variable to scope error.
-33128	Setvar error, Indirect data casting is forbidden in binary setvar.
-33129	Setvar error, Binary setvar for opaque variables is forbidden.
-33130	Setvar error, Wrong binary setvar of date. Date variable plus/minus integer constant or variable is only allowed.
-33132	Variant error. Any cast is forbidden if both operands are TML variables.
-33133	Variant error. Only string operands are allowed if both operands are constants.
-33134	URI map error.
-33135	Too many "unique_id" requests on the same screen
-33137	Directory init error.
-33138	Getvar error.
-33139	Incorrect variable type.
-33141	User Log scope loading error.
-33143	Error processing string template.
-33144	Runtime error.
-33145	ITID isn't changed because of off-line posts presence.
-33146	Error trying to manually connect to the OEGW by setting `oebr.connection.state` to `connected`

(-332XX) ISTP errors

Error Code	Description
-33200	Unable to load/save TML variable.
-33201	Unable to send data via channel.
-33202	Unable to receive data via channel.
-33203	Terminal credentials are undefined. Terminal might not been initialised.
-33204	Memory allocation failure.
-33205	Unable to uncompress incoming message
-33206	Packet with unsupported ISTP version received
-33207	ISTP packet is longer than 64k

(-333XX) SSL_TRANSPORT errors

Error Code	Descrotion
-33301	Unable to initialize ssl transport
-33302	Unable to load key material
-33306	Socket connection failed
-33307	Unable to create socket
-33308	Unable to establish session
-33312	SSL read error
-33313	SSL write error
-33314	The other peers want to close connection
-33315	Invalid SSL certificate
-33316	SSL certificate expired
-33317	SSL certificate commonName does not match connect target
-33318	SSL memory allocation error
-33319	SSL run-time error
-33320	SSL cache error

(-334XX) NET_MEDIA errors

Error Code	Description
-33401	Unable to operate with TML variables
-33402	Unable to resolve host address
-33403	Invalid netmask
-33404	Invalid TML var state

(-335XX) PERIPHERAL errors

Error Code	Description
-33501	Peripheral busy
-33502	Peripheral not available
-33503	Peripheral key pressed
-33503	connection aborted by user
-33504	Peripheral bad data
-33505	Peripheral internal error
-33506	Peripheral timeout
-33506	connection timed out
-33507	Memory error
-33508	Unknown audio sound type '%s'

(-336XX) IMAGER errors

Error Code	Description
-33601	Check baddata
-33602	Key pressed
-33603	Imager invalid output
-33604	Imager timeout
-33605	Imager TML var read error
-33606	Imager TML var write error
-33608	Imager not open
-33609	Imager config read failed
-33610	Invalid imager command code: %s
-33611	Invalid imager command in sequence: %s
-33651	Command process error
-33652	Wrong IMAGER out data size
-33653	Wrong IMAGER out data format
-33654	Unable to open IMAGER port. Due to multiplexing the port conflicts with COM1
-33699	Imager general error

(-337XX) PAM errors

Error Code	Description
-33701	Check baddata
-33702	PAM TML open error
-33703	PAM TML var read error
-33704	PAM TML var write error
-33706	PAM amdSend failed

(-338XX) VKBRD errors

These errors are related to the Virtual Keyboard operation.

Error Code	Description
-33801	Memory allocation error.
-33802	Configuration load error.
-33803	Peripheral error.
-33804	File open error.
-33805	File read error.
-33806	HMI-function error.
-33807	Keyboard widget draw error.
-33808	Varlib error.
-33809	Language is not supported.

Error Code	Description
-33810	Runtime error.

(-339XX) IND errors

Error Code	Description
-33901	Memory allocation error.
-33902	File open error.
-33903	File read error.
-33904	Skin picture has wrong BPP depth.
-33905	Skin loading error.
-33906	Indicator draw error.
-33907	Runtime error.

(-340XX) LOG errors

Error Code	Description
-34001	Memory error
-34002	Variable read error
-34003	Variable write error
-34004	Parser error
-34005	File error
-34006	Varlib error
-34007	Wrong layout
-34008	Invalid logger
-34009	Directory init error
-34010	No parent logger
-34011	Config loading error
-34012	Record size limit exceed
-34013	Logger is locked
-34014	Access error
-34015	No records found in log
-34016	HTTP client error
-34017	Internal logger error
-34018	Critical DFS free space-logs have been reduced by half

(-341XX) HCL errors

These errors are related to the Hot Card List operation.

Error Code	Description
-34101	Incorrect pan
-34102	Hot Card List file missed
-34103	Hot Card List file corrupted
-34104	Hot Card List internal error

(-342XX) OEMSG errors

Error Code	Description
-34201	Out of space
-34202	Buffer overlap
-34203	Buffer gap
-34204	Integrity fail
-34205	Invalid processor

(-343XX) GCL_PGCOMM errors

Currently, Incendo Online MicroBrowser does not produce any GCL_PGCOMM errors