EasyServerPages

User Interface Design Based on HTML5

Content

Basic Considerations....2
The Agent....2
Mass Data Access....4
General Setup....4
Translation....6
Connectivity and Asynchronously Calls....6
Recursion....7
ESP Document....7
- data-eezz-action....9
- data-easy-event....10
- data-easy-template....11

ESP Objects....12
- TTable....12
- TCell....13
- TDBTable (TTable)....14
- TBlackBoard....14

Example....15
- The HTML page....15
- The Application....16

Basic Considerations

ESP (EezzServerProtocol) implements a data exchange protocol. It adopts the idea of global status codes from HTTP, but uses a JSON structure, which could be consumed by all socket endpoints. This allows two or more services to interact and dissolves the current definition of client and server.

```
{ 'callback': { '<application-name.method>' : { '<parameter-name>':'<value>', … }, …  },
  'update'  : { '<browser-tag.attribute>' : '<referece-tag.attribute>', … }
  'return'  : { 'code': <number>, 'value': '<description>', 'args': [ '<value>', … ] },
  'version' : '<version string>'  }
```

Text 1: ESP Structure

The return codes are based on https://de.wikipedia.org/wiki/HTTP-Statuscode.

Beside the data, the protocol defines activities for the partner to execute (callback) and defines regions in the JSON structure to place the results into (update). It returns always a status code, a description value and an optional list of values for output.

An ESP package could visit several services in one roundtrip, allowing to keep track on each station. Each service would fetch the relevant information and add data to the structure.

Central issues was

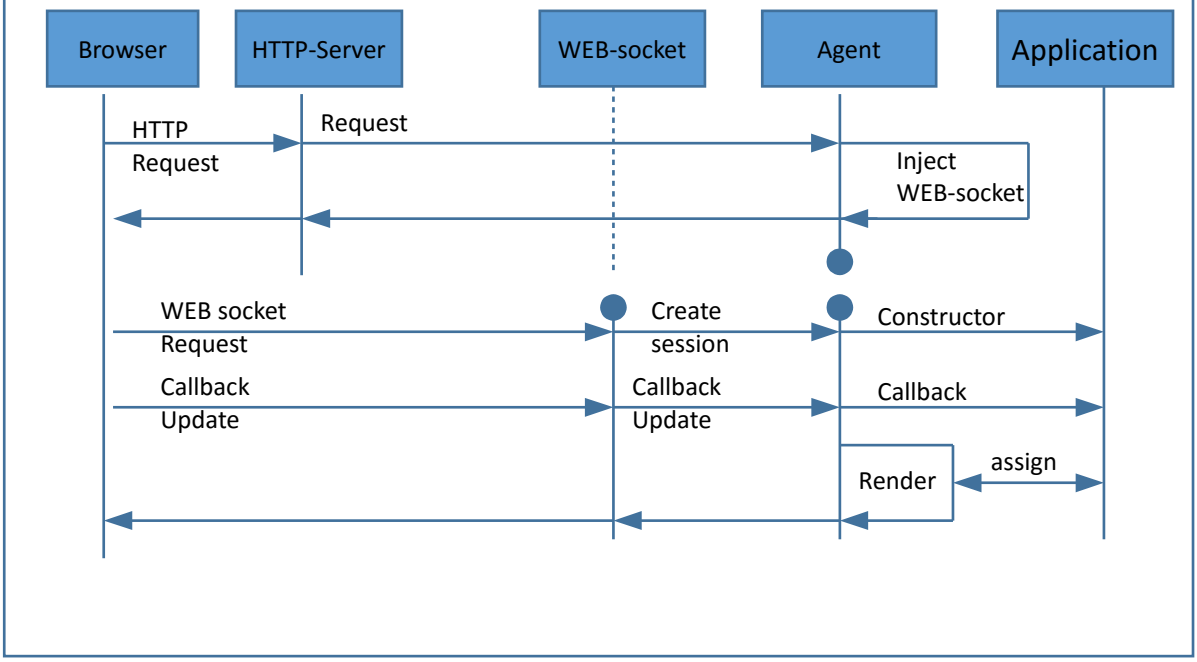
- Mass data awareness. There are methods for selecting virtual data slices and for scrolling and block navigation on huge data sets.
- Speed in execution. Using WEB-sockets allows flicker free and high speed transfer of updates to an existing page.
- Speed in development. Strict decoupling of user interface and application reduces the efforts in development and test.
- Reduce usage of java script for clear separation of responsibilities, better maintenance and less complexity.

The Agent

The agent has two main tasks:

1. Session management
2. Connecting back-end data selection to the user interface and rendering

The ESP project implements a HTTP server with an agent to analyze and compile HTML5 pages. A session is established in two steps. First locate the project opening a ESP-HTML via HTTP and then starting a session continuing with web-socket protocol.



Drawing 1: Time Flow

The agent session object is connected to the web socket session. If the user closes the browser, the web-socket server would send a shutdown request to the agent.

If a browser has a session, which remained after a network outage or a reset of the ESP server, the agent would be able to restore this session.

In a ESP-HTML page the agent extracts HTML-tags with the following attributes

- data-eezz-action: Executes agent methods, like assigning elements or dictionary access.
- data-eezz-event: Point the ESP protocol elements, which are used for data exchange.
- data-eezz-template: Tells the agent what and how to display variable elements.

The data-eezz-event point the ESP protocol elements, which are used for data exchange. The data-eezz-action calls the agent assign objects as elements, which could be used in this events. The assign has two formats, a constructor and a method call. The third entry shows a build in function to get and place the websocket protocol as java script into the page:

```
{'eezzAgent.assign' :'<path-to-application>/<class-name>' : { '<parameter-name>':'<value>', … }}
{'eezzAgent.assign' :'<application-name.method>' : { '<parameter-name>':'<value>', … }}
{'eezzAgent.assign' : 'eezzAgent.get_websocket':{}}
```

Text 1: Assign Structure

The first is used to load a module and call a constructor, the second to access the result of a request. The parameter in assignment could be used for initial access or to recover a session.

The event and the assign statement are equivalent to a SQL select and fetch statement respectively. Each application is first called with the constructor statement to load the module and initialize the

application. Subsequently the application is called the event callback method and has to provide the result in a call of the assign method. The environment expects an object of type TTable as return object.

The ESP agent uses the HTML page to extract templates. It fills the data given in a TTable object into these fragments and returns the result to the browser, which replaces or inserts the calculated data into the right place. The callbacks do_select and do_sort are automatically generated for rows and columns in a table respectively, if not superseded by user defined callback. The user could overwrite these methods to define own activities.

Any application could implement a service to push data to the front end. To do so the application places an update request to the blackboard and the agent would pick that up. This way it's possible to implement a huge set of services without disturbing the agent. At a certain point in time the agent collects all update requests to execute them in one single step, minimizing access to network.

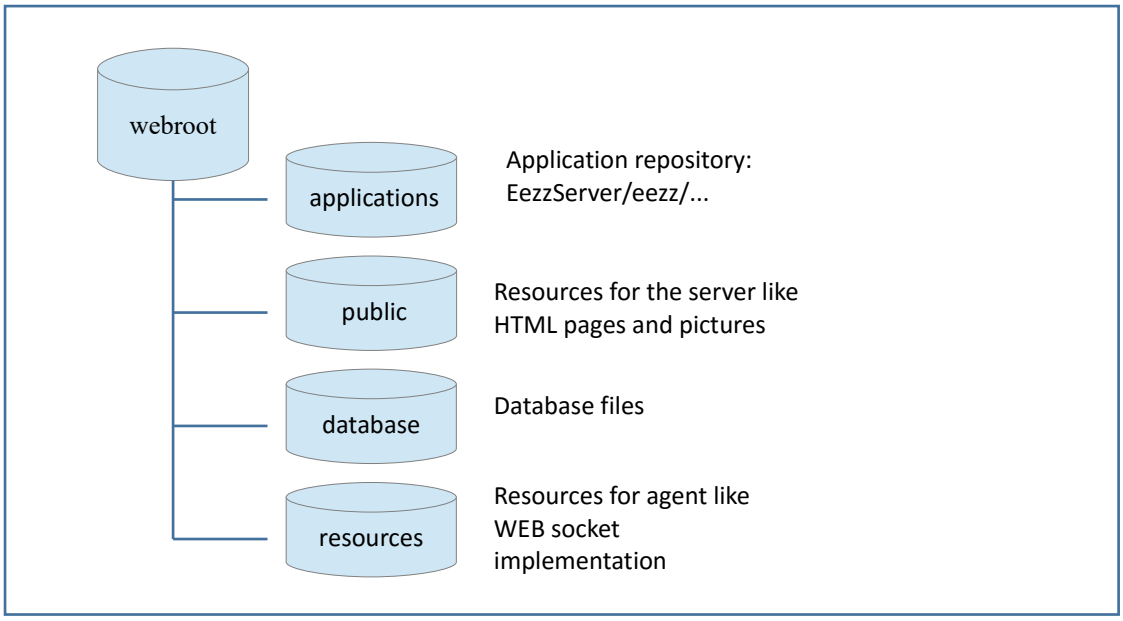
Mass Data Access

The TTable and TDBTable(TTable) are designed to handle bug data sets for output. The table defines a scroll threshold, which automatically switches to a block navigation on overflow.

The TDBTable works analog with database tables and calculates the desired number of result records for each navigation step.

General Setup

ESP (EasyServerPages) is based on HTML-5. On the server you would see the following hierarchy after setup:



Drawing 2: Structure on Disk

To start the service, you would need to install python3.3 https://www.python.org/downloads and launch the following command

```
> set PYTHONPATH=<path to webroot/applications/EasyServer/eezz>
> cd <path webroot/applications/EasyServer/eezz
> python34 server.py -webroot <path to webroot> -port 8000
```

Text 1: Starting the Server

The directory structure contains the following:

- The directory “public” contains the HTML files, available for the browser.
- The directory “resources” contains the translation and the java script for webscokets.
- The directory “database” contains sqlite3 database files
- The directory “applications” contains the server and user applications

You could start design a page without any Java-Script or any function logic, by using the following additional attributes keeping the usual HTML-5 layout:

ESP custom attribute	Description
data-eezz-action	Defines actions on document load
data-eezz-event	Defines actions on mouse and keyboard events
data-eezz-template	Defines layout for generated elements

Table 1: Custom HTML Attributes

ESP exports the following data types, which could be implemented in the application and being referenced by the user interface:

Exported Data Types	Description
TTable	Used for data exchange of customized tables
TDbTable	Used for data exchange of database tables
TCell	Used to customize type of a table cell or row
TBlackBoard	Used to organize for data exchange for async calls and services

Table 2: Data Types

Translation

The translation module of python compiles the HTML page to generate a file <page-name>.po. This file could be translated to a <page-name>.mo and deployed into the branch locales.

Text elements are marked using the following syntax:

```
<tag>_( " <text to translate> " )</tag>
```

Text 1: Mark elements for translation

Extract the test using gettext would create a <messages>.pot file, which could be translated manually to a <messages>.po file for each language. Such a file has to be compiled using msgfmt to a machine readable format <messages>.mo

```
> Tools\i18n\pygettext.py <esp-document>.html
> Tools\i18n\msgfmt.py <messages>.po
```

Text 2: Python Translation Elements

Example for a valid header of a <messages>.po file:

```
# Translation of index.html.
# Copyright (C) SAP
# Albert Zedlizt albert.zedlitz@gmail.com, 2013.
#
msgid ""
msgstr ""
"Project-Id-Version: 1.0\n"
"POT-Creation-Date: 2013-11-26 17:51+W. Europe Standard Time\n"
"PO-Revision-Date: 2013-11-26 17:51+W. Europe Standard Time\n"
```

Text 3: Header of a PO File

Special characters have to be supplied using the following syntax “#<char-spec>;”
To run this tool the HTML has to be formatted with indent 4 like python code.

Connectivity and Asynchronously Calls

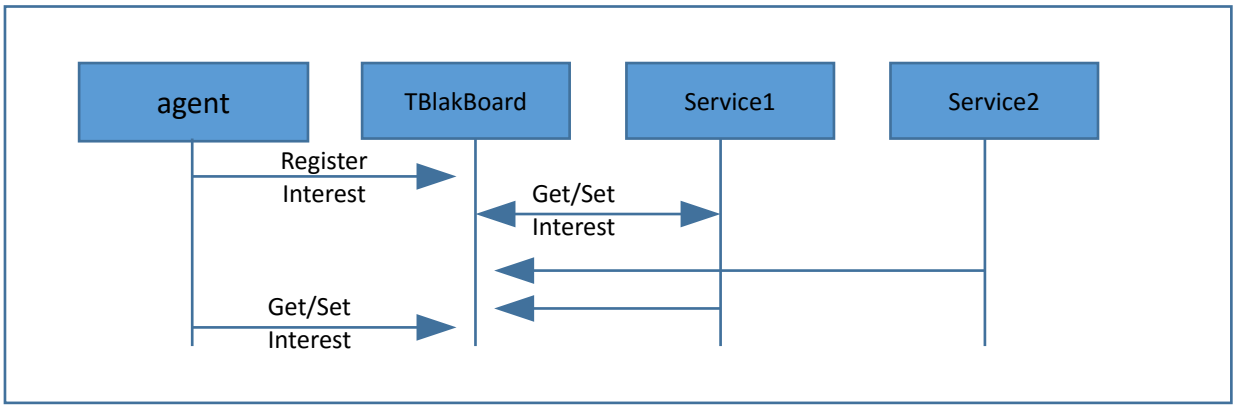
The agent might communicate with several services. The interaction should not effect the user interface in a visible delay of response.

The user should not wait for a long running process nor getting a flood of input if all services return in an asynchronously call.

The TBlackBoard class is used for data exchange for services and agents. An agent registers for a certain interest. A services would write into the TBlackBoad any time without waiting and the agent could access the data any time.

This way the agent could collect the data of several asynchronously calls in one single update request, minimizing the interaction to the user interface and hence network load. This concept would guarantee that services and agents are not blocking each other.

Two agents could connect via TBlackBoard for interactive data exchange implementing a common chat room as interest.



Drawing 3: Synchronize response in TBackBoard

Recursion

The Agent allows usage of recursion in the templates. The CSS style “class=eezzTreeNode” for a table indicates that that this structure could be reused for all row elements in this table. The CSS style “class=eezzNodeLeaf” creates unique ids for an element in such a structure, so that you could address this element.

An element in a tree is uniquely identified by its path. The user has to provide this path in a TCell, given for each row in a TTable object. The data supplier is called with this value and has to provide new elements for expanding a node element. Usually such a program takes the path given as parameter in a method call and expand this value for each new entry in the new node.

This concept allows a very compact definition of recursive structures for tree views.

ESP Document

The ESP document is based on HTML using customer attributes with name “data-*”. The attribute argument is a command in JSON format. Key and value have to be in single quotes. The following statement would be translated to the following equivalent and you could access this object in the following using aMyPage in data-easy-event callback:

```
01 <!DOCTYPE html>
02 <head>
03     <script data-eezz-action = "{'eezzAgent.assign':{'eezzAgent.get_websocket':{}}"/>
04 </head>
05 <html name = "aExample"
06     data-eezz-action = "{'eezzAgent.assign':{'EasyTest/module.easytest/TDirView':{}}}">
07     <body>
08         <table name = "aDirList" class="eezzTreeNode"
09           data-eezz-action = "{'eezzAgent.assign':{'aExample.getTreeRoot':{'path':'/'}}}
10           <tbody
11               data-eezz-event =
12                   "{'callback':{'aExample.getDirlist':{'path':'this.data-current-path'>
13               <tr data-eezz-template = "{'table-rows':[':']}">
14                   <td data-eezz-template="{'table-columns':['1:']}"></td></tr><tbody></table>
15     </body>
16 </html>
```

Illustration 1: ESP Document

In statement 03 we include the websocket java script interface, which is mandatory to run this page.

In statement 05 we specify the package and the class, which we could use as data source. The search path starts in the directory given in “webroot/applications”, with webroot given as start

parameter to the server. The statement in 05 creates an instance of this class. The line is translated to the following two statements:

```
from module.easytest import TDirView
aExample = TDirView
```

Text 1: Compile Line 02 in ESP Document

In the given example the class TdirView implements the method getTreeRoot, which is initially called with named parameter “path='/'”. The statement 08 attaches an instance of TTable, which is returned by getTreeRoot to the table tag.

The agent would now call the method getTreeRoot to fetch list of file in the path. For each row the agent generates a callback for, which is specifed in <tbody>. For elements of class eezzTreeNode the update is calculated by the agents tree management.

The value of this.data-current-path is given with the TCell, which has to be created for each row input by TDirView class:

```
xCell  = TCell(0, 'directory', {'data-current-path': os.path.join(<path>, <file>)}
xTree.append([<file>], xCell)
```

Illustration 2: Row type and attributes

The TCell has a value, a type and a set of parameter. The value data-current-path is used by the agent to calculate the path in line 12 of the ESP document. The type could be used to select an icon or a specific ouput format.

The statement in 14 shows, that its possible to restrict the output of columns either as range or list of column names.

data-eezz-action

The attribute is executed at document load and defines activities to load user applications and setup of the session. It defines the following activities:

Activities	Description

eezzAgent.assign	Assigns an object to an HTML tag. The syntax is as follows:
	`1. {'<path>/<package dir>/<module file>/<class>':{<arguments>}}`
	`2. {'<object>.<method>':{<arguments>}}`
	`3. {'eezzAgent.get_websocket':{}}`
	`4. {'eezzAgent.get_animation':{}}`
	The first assigns a package and a class to the ESP, the second uses the object created in this statement. Both method should return a TTable object.
	In 3. and 4. internal methods of eezzAgent are executed. The method get_websocket inserts the websocket java script interface into the HTML and should be used with tag <script>. The method get_animation would insert java-script for animation interface.

eezzAgent.dictionary	Assigns user dictionary to any HTML tag. This dictionary is visible for the child objects in the same hierarchy. The following call should return a python dict
	`{'<object>.<method>':{<arguments>}}`
	You prepare access to dictionary values with the following assignment in the HTML document:
	`<tag data-myValue = {key} ></tag>`
	It would be possible then to use this data-* attribute in update activity. In the text section of the HTML document you could use curly brackets to access dictionary values for output:
	`<tag> {key} </tag>`

eezzAgent.database	Generates a text database from an HTML-TABLE tag. A return value for event objects would access the HTML-TR tag to evaluate the output.
eezzAgent.async	Used to organize asynch calls. The syntax is the same as for eezzAgent.asign.

data-easy-event

The attribute implements mouse and keyboard events on the user interface. It defines the following activities:

Activities	Description
callback	Defines the callback to execute on user interaction. The TTable has its own callbacks, which are ready to use. The following syntax is used to define own callback: `{'<object>.<method>': { -empty- | 'arg-name':'arg-value', … }`
update	This entry contains all elements, which needs an update after callback `{'<destination >.<attr>': '*' | '<source>.<attr>', … }`
files	Defines files for download as array. You select files using the HTML-INPUT tag. With the source you access the name of this tag. The progress returns a text and a width for the given element. `{'source':<element>, 'type':<type>, 'progress':'<element>'}`
reader	Defines the processor for file downloads `{'<object>.<method>': { -empty- | 'arg-name':'arg-value', … }`
version	The version of the protocol

data-easy-template

This attribute is used for output generation output of TTable object

Activities	Description

table-rows	Defines the HTML-TR tag for row output.
	• In combination with HTML-THEAD output is done for table header
	• In combination with HTML-TBODY output is done for table body
	• In combination with HTML-SELECT output will generate OPTION tags
	
	The following entry is used to display all rows:
	`{ 'table-rows' : [ ':' ] }`
	
	The following changes the layout to tiles. In this case all columns of a row are collected into one tile. Values could be accessed within each tile using the column name.
	`{ 'table-rows' : {'tiles':4} }`

table-columns	Defines the HTML-TH or HTML-TD tag for column output. The following entry is used to display all columns. You would need the curly brackets in the inner HTML. The syntax is the same as for table-rows, but you could specify column by name or by range:
	`{ 'table-columns' : [ 'Column1-Name', 'Column2-Name', … ] }`
	`{ 'table-columns' : [ '1:' ] }`
	`{ 'table-columns' : [ '1:3' ] }`
	
	Use curly brackets to define the place, where to put the column value:
	`<td data-easy-template={ 'table-columns' : [ ' : ' ] }>{}</td>`

display	Sometime you want to make something visible depending on internal state. You could set the visibility of an entry according to an entry in the dictionary:
	`{ 'display' : {'<dictionary-key>', '<value>'}`
	
	A special key to evaluate the type of a row or column. This row or column is displayed only if the type given in TCell object matches. This could be used for formatting or selecting from a table a specific row:
	`{ 'display' : {'type', '<type of the table row>'}`
	
	A special key to evaluate the status of a column. This column is displayed, if the status of the given TCell object of a row matches. Status fields are automatically stored for asynchronously update.
	`{ 'display' : {'status', '<status of the table row>'}`

ESP Objects

TTable

Method	Description	
__init__	The TTable is used to connect the user application with the user interface. It maintains rows, header information and a dictionary, which could be accessed in both contexts.	
	`aColNames = list()`	List of column names. This is also the dimension of all rows.
	`aHeaderStr = str()`	Caption
	`aHeaderDict = dict()`	Dictionary used by TTable to push internal values like current selection and number of entries
	`visible_items = 100`	Visible items as the number of rows, which would be send to the browser
append	Appends a row into the table	
	`aRow`	List of values. If row dimension does not match column dimension, the append will fail. The list may contain following types • int • str • float • TCell • TTable
	`aCell = None`	TCell: Optional type specification for row
do_select	Callback for selecting row. This is used to prepare method get_selected_obj.	
get_selected_obj	Returns the selected table.	
	`index = -1`	Specifying -1 returns the current table. Otherwise the TTable in column[1] for the specified index. If no hierarchy is defined, this method returns the current table.
	`visible_items = None`	Reset the visible item count.
	`visible_block = None`	Reset the visible block. The visible block is used, if visible item count is exceeded by the number of entries. This could be used to switch navigation strategy from scroll to block.

get_rows, get_raw_rows	Returns a subset of rows of the table [offset: offset + visible_items] The get_rows returns formatted rows, the get_raw_rows the elements of a row as list of rows
get_selected_row	Returns the selected row as a list of values
get_selected_index	Returns the selected index
prinTable	

TCell

Method	Description	
__init__	The TCell is used to define the type of a cell. The cell types for the entire column could be set using TTable method set_column_type. TCell would overwrite this value	
	`aValue`	The display value
	`aType`	String as user defined type
	`aStatus`	Status for a given cell
	`aObject`	Any user defined object
getValue	Returns the TCell value	
getType	Returns the TCell type	
getStatus	Returns the TCell status	
getObject	Returns the TCell object	

TDBTable (TTable)

Method	Parameter	Description
__init__	Initializes a database view. The select statement is processed by the TDBTable object to optimize the data flow.	
	`database`	Path to a sqlite3 database
	`select`	Select statement in JSON structure
	`parameter = None`	Parameter for future use
execute	Execute the select statement using the visible row and visible block attributes. The aOrderInx is used to specify the sort order	
	`aOrderInx = [ASC|DESC]`	Sort order for the select

TBlackBoard

Method	Parameter	Description
__init__	TBlackBoard is used to exchange data between applications and agent.	
addMessage	Any application could advertise data	
addInterest	Any application could place an entry on blackboard, which is filled by addMessage	
getInterest	If an application fetches the information for a given entry, this entry is removed from blackboard	
delInterest	Any application should remove its interests from blackboard before it terminates.	

TTracer

Method	Parameter	Description

__init__	Initializes the traces using environment given by TBlackBoard.mRootPath	
writeException	Write after catching an exception	
	aLevel	The trace level hint for output.
	aJsnReturn	{'result': {'code':<int>, 'value':<message>}
setTraceLevel	Set the trace level	
	aLevel	The minimum level for output
write	Write a trace	
	aLevel	The trace level hint for output
	aReason	The header text for a reason of the trace
	aJsonReturn	{'result': {'code':<int>, 'value':<message>}
get_selected_sessions	Return a table with sessions. A session is a time interval between two starts. This method should be used in HTML	
get_selected_traces	Returns a table with all traces for a given session. This method should be used in HTML	

Example

The HTML page

Connect the application with the websocket interface

```
<script data-eezz-action="{'eezzAgent.assign':{'eezzAgent.get_websocket':{}}}"/>
```

Connect a class object to the tag. This tag could be accessed by name in other tags.

```
<html name="aExample"
    data-eezz-action="{'eezzAgent.assign':{'EasyTest/module.easytest/TDirView':{}}}">
```

```
from module.easytest import TdirView
aExample = TDirView()
```

Connect a table to a tag and specify output restrictions

```
<table name="aDirList"
       data-eezz-action =
           "{'eezzAgent.assign':
                {'aExample.get_selected_obj':{'index':'-1', 'visible_items':'40'}}}">
```

```
aDirList = aExample.get_selected_obj( index = -1, visible_items=40)
```

Specify the table header

```
<thead>
    <tr data-eezz-template="{'table-rows':[':']}">
        <th data-eezz-template="{'table-columns':['1:']}">{}</th></tr></thead>
```

```
for column in aDirList:
     '<th>{}</th>'.format( column )
```

Specify the table body

```
<tbody>
    <tr data-eezz-template="{'table-rows':[':']}">
        <td data-eezz-template="{'table-columns':['1:']}">[}</td></tr></tbody>
```

```
for row in  aDirList.rows:
   '<tr>'

   for column in row:
     '<td>{}</td>'.format( column )

   '</tr>'
```

The Application

The main program to test the application- You could test your application with the following command:

> python mytest.py

```
if __name__ == '__main__':
    aTest = TTest()
    aTest.printTable()
```

The class definition, which correspond to the assign statement. The table has two columns and 20 rows and the dictionary entry {table_caption:MyTable}

```
class TTest (TTable):
    # initialize a table
    def __init__(self):
        super().__init__(['Column1', 'Column2'], 'MyTable')
        for i in range(20):
            self.append(['value-{}.1'.format(i), 'value-{}.2'.format(i)])
```

The following method is called for each click in the table. The entry visible_items is used to define the size of the scrollbar. The entry visible_block is used, if the number of elements exceeds the visible_items to define a navigation block.

```
  def get_selected(self, index=-1, visible_items=None, visible_block=None):
         return TTable.get_selected(self,
                           index         = index,
                           visible_items = visible_items,
                           visible_block = visible_block)
```