Tetra WebApps integration with Telium Services

User Manual

ICO-OPE-02910 V3

Software Engineering

February 2016

www.ingenico.com

28/32, boulevard de Grenelle, 75015 Paris - France / (T) +33 (0)1 58 01 80 00 / (F) +33 (0)1 58 01 91 35

Ingenico – S.A. au capital de 53 086 309 € / 317 218 758 RCS PARIS

Contents

1 Document Information

1_1 Evolution Follow-up

Revision	Type of modification	Author(s)	Date
0.X	Drafts	Roland MARTINET Nicolas RAIBAUD	-
1.0	First Release	Roland MARTINET Nicolas RAIBAUD	2015/06/24
V1	Addons and fixes. “tetra.lookup” and “tetra.setup” are now promises Document synchronised with tetraJS v 1.1.0	Roland MARTINET Nicolas RAIBAUD	2015/07/15
V2	Added the tetra.weblet.notify() method	Roland MARTINET Nicolas RAIBAUD	2015/08/11
V3	Added Web application as Service Correct manageTransaction sample	Yannick MEYER	2015/11/03
V3.1	Added WP hide and show Added checkServices	Yannick MEYER	2016/02/05

1_2 Document Validity

	Name	Function	Signature	Date
Verified by	Yannick MENET			2016/02/05
Approved by	Karim ACHARI			2016/02/05

1_3 Objective

This document describes how Tetra web applications interface with Telium Tetra services.

1_4 Scope

This document explains the mechanisms behind the interface with Telium Tetra services, and introduces the tetraJS library that provides a high level abstraction API to simplify access to these services.

The document has been updated to match the tetraJS library version 1.2.0.

2 Tetra terminal software architecture

Tetra terminals feature three applicative frameworks that are able to interact with each other through a common software layer: the “Telium Tetra services”.

- The **Telium+ native framework** allows the execution on Tetra terminals of existing legacy Telium+ applications (migrated applications originally running on Telium2 terminals). Telium+ applications on Tetra terminals are executed over an emulation layer connected to Telium Tetra services.
- The **Telium Tetra native framework** allows the execution of a new type of Telium application built directly on top of the Telium Tetra services.
- The **Tetra web framework** allows the introduction of a new type of application: Tetra web applications (WebApps) exclusively built with web technologies (HTML, CSS, and JavaScript). The new “Tetra Web Platform” provides the execution environment to interpret and render WebApps and also ensures the intermediation between WebApps and Telium Tetra services.

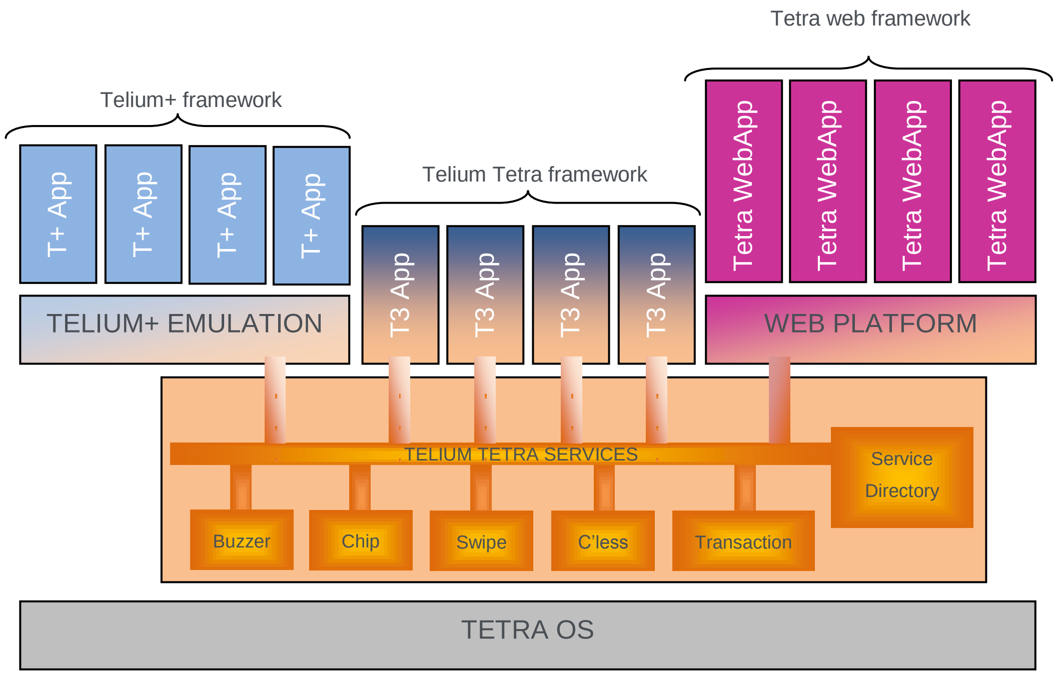

Fig 2.1 Tetra terminal software high level view

Telium Tetra services are the exclusive mean for applications running in various frameworks to:

- Access Tetra terminal physical peripherals such as the smartcard reader, the magnetic stripe reader, the contactless reader, the buzzer, … in a collaborative way
- Access high-level macro functionalities such as the transaction service.

2_1 Telium Tetra services

Applications that wish to use Telium Tetra services can discover the available services and connect to them by mean of a special service called the “Service Directory”. Once connected to a given service, the client application can then exchange messages directly with that service.

Each service exposes a set of one or more interfaces and events.

- Each interface implements one or more methods. Calling a method results in the exchange of two messages: a request and a response.
- Events are unsolicited messages sent by the service to any connected application, upon the realization of predetermined conditions.

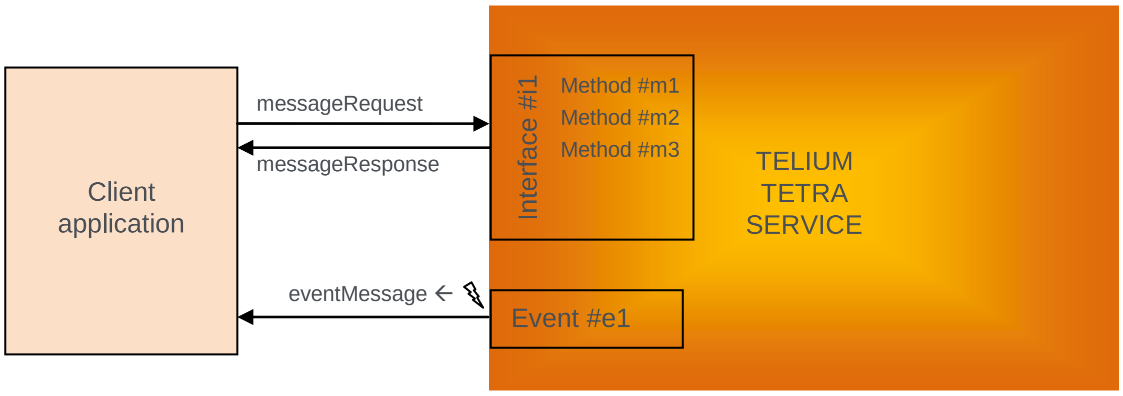

Fig 2.2 Client application interactions with a Telium Tetra service

To connect to a service exposing a given interface/event, an application needs to know:

- the service namespace in which the interface or the event is defined
- the service name which identifies a specific running instance of a service having a given namespace

In other words, the name of an interface or an event can exist within several namespaces. This is why to identify an interface or an event, both the service namespace and name have to be provided as input to the tetra.service() method.

In addition to this, to execute a method or be notified of an event, an application needs to specify:

- the “interface” and the “method” names or the “event” name

At runtime, a lookup method exposed by the “Service Directory” can be used to discover the list of all the running instances that implement a given service interface within a certain namespace.

The Web Platform provides WebApps with a set of Web Services to interact with Telium Tetra service interfaces and events. This also includes the interface to the Service Directory.

2_2 Getting access to Tetra services from Tetra WebApps

As suggested in ***Fig 2.1***, access to Telium Tetra services from Tetra WebApps is managed through the Web Platform. In effect, it is the Web Platform that connects to the services on behalf of the WebApps, and it exposes to the WebApps a set of “REST” web services that abstract the underlying mechanisms to access Tetra services.

The REST web services exposed by the Web Platform are listed for information in Appendix A. However, they are not meant to be called directly. Instead we recommend the use of the TetraJS library which provides a higher abstraction level and API (see section 3.).

The REST web services exposed by the Web Platform fall in one of two categories:

- web services intended for Web Platform configuration/setup
- web services intended for direct interface with Telium Tetra services

Calling a service interface methods and receiving event notifications from a service will involve different mechanisms and will be managed differently by the Web Platform.

2_2_1 Calling a service interface method

To call a service interface method, a WebApp will initiate an AJAX call to the Web Platform that will only return when the interface method on the native side will have been executed.
The Web Platform will interpret the AJAX request and convert it to a native call to the native service and perform the reverse operation with the answer from the native services.

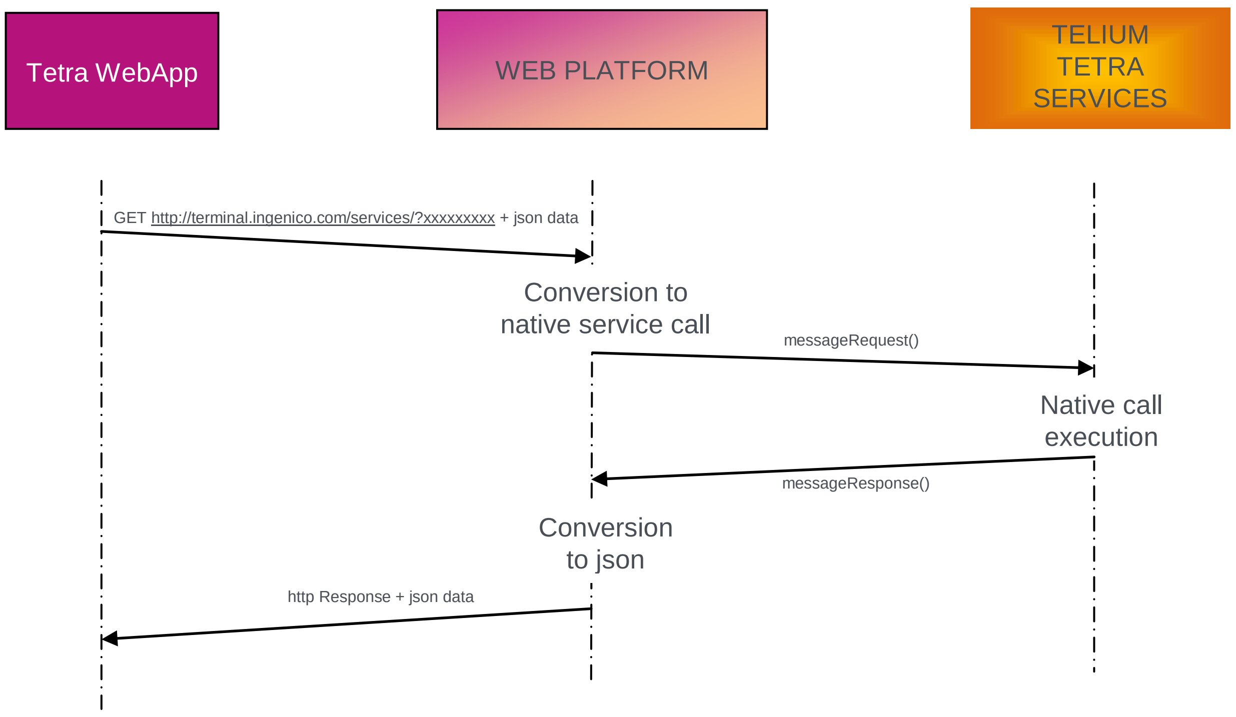
Tetra WebApp
WEB PLATFORM
TELIUM TETRA SERVICES
GET http://terminal.ingenico.com/services/?xxxxxxxxx + json data
Conversion to native service call
messageRequest()
Native call execution
messageResponse()
Conversion to json
http Response + json data

Fig 2.3 Calling a native Telium Tetra service interface method from a WebApp

2_2_2 Receiving a service event notification

To be notified of an event from a service, the WebApp needs to create a dedicated ***eventSource*** with an appropriate uri. The Web Platform will then register the eventSource and manage it as a SSE (Server-Sent Event).

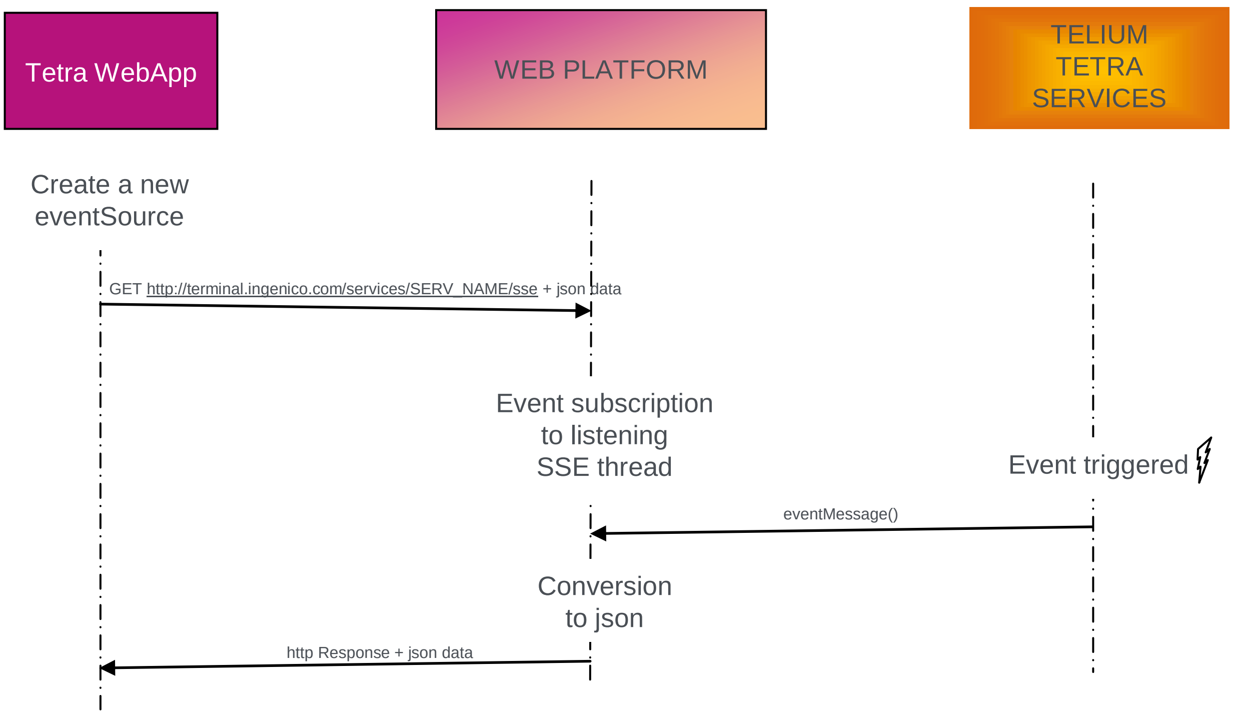
Tetra WebApp
WEB PLATFORM
TELIUM TETRA SERVICES
Create a new eventSource
GET http://terminal.ingenico.com/services/SERV_NAME/sse + json data
Event subscription to listening SSE thread
Event triggered
eventMessage()
Conversion to json
http Response + json data

Fig 2.4 Telium Tetra service event notification between a WebApp and a Tetra service

2_3 Telium Tetra services access rights and permissions

Application access to Telium Tetra services is filtered according to its access rights as defined by its signature.

Access to any interface method exposed by a Telium Tetra service is conditioned by a required level of access rights (Minimal / Vertical-Loyalty / Payment) and/or a required level of permission (Web Platform, Web User Component).

Regular WebApps will be signed “Web User Component” with either “Minimal” or “Vertical-Loyalty” access rights. This profile will limit the services/methods to which WebApps will be granted access.

The table at the beginning of section 3_7 lists the service interfaces that are exposed to WebApps (to date), together with the minimal Telium Tetra SDK version supporting the feature.

2_4 Using Telium Tetra services in collaborative mode

As presented previously, Telium Tetra services constitute a software layer common to all three frameworks present in Tetra terminals and are accessible to all applications running on these frameworks. All applications run in parallel and are likely to solicit the same Tetra services, some of which do not necessarily support simultaneous access from different applications. This mandates the applications to follow strict rules to coexist on the same terminal. WebApps **MUST** release services when they are not used so that other applications can access them.

As an additional protection, the Tetra Web Platform ensures that all services left open when a Web application terminates are automatically closed.

2_4_1 Releasing services that are not used

This rule particularly applies when a WebApp loses the display focus (a WebApp can detect this state by means of the JavaScript “visibility change” event).

- For services that cannot accept multiple simultaneous clients (i.e. not sharable by design), it is necessary that WebApps disconnect from the service when they lose focus.
- For services that can handle multiple clients and implement a mechanism to handle mutual exclusion, application must leave any critical section they could be in when they lose focus.

2_4_2 Releasing services when the application terminates

WebApps terminate in one of two cases:

- Termination is explicitly coded in the application’s JavaScript code
- Termination is triggered by a user action forcing the immediate application termination from the Web Desktop or Task Switcher

In the latter case, handling the release of services to which the application is still connected can be difficult to manage for the developer depending on the state of the application when the termination occurs. This is why it is recommended to use the TetraJS library where logic was added to automatically close all open services upon application termination, regardless of the termination cause.

2_5 Weblets vs WebApps

Tetra Web Platform is a framework intended for the execution of WebApps. WebApps can be splitted into one or more weblets. Basically, the simplest WebApp is made of a single weblet. Each weblet is allocated a webview for its rendering.

Web developers that wish to split their WebApps into functional blocks, having their own webview, can create multi weblet WebApps.

Weblet is the smallest html running unit managed at Web Platform level. This is the reason why a “weblet” object will be considered rather than a “WebApp” object in tetraJS.

2_6 Authenticated WAID

The WAID is the Web Application IDentifier. This number uniquely identifies the application. The WAID is attributed by the MarketPlace and is mandatory in the web application package.

The Web Platform provides a mechanism for a service to authenticate the WAID of the calling web application. This is implemented through a reserved message parameter that, when defined defined in the message description, is replaced by the Web Platform with the authenticated value of the WAID read from the signed package (rather than trusting the Web Application to provide the information).
Simply insert the reserved property “**authwaid**” in a message definition and the parameter will be filled by the Web Platform with the authenticated WAID value. This only applies to request messages, i.e. messages sent from a client to a service.
If the "`authwaid`" field is present in a JSON message, its value will be replaced by the real WAID. If it is not present – the new binary value will be implicitly added to the message.
Fields with the name "`authwaid`" in response messages are handled just like any other message parameter.

For example, consider the following message description

```
package ingenico.test;

message DoTestRequest
{
  required string testData = 1;
  optional string authwaid = 2;
}

Message DoTestResponse
{
  required int32 status = 3;
  required string authwaid = 2;
}

//Test interface
service Test
{
  rpc doTest (DoTestRequest) returns (DoTestResponse);
}
```

Request from a client web application to a service

Client	Service

JSON send: {
"testData": "Hello"
}
	JSON received:{
	"testData": "Hello"
	"authwaid": "01234567"
	}

JSON send: {
"testData": "Hello"
"authwaid": "FakeWAID"
}
	JSON received:{
	"testData": "Hello"
	"authwaid": "01234567"
	}

Response received by a client web application from a service

Service	Client

JSON send: {
  "status": 0
  "authwaid": "WAID?"
}
	JSON received:{
	  "status": 0
	  "authwaid": "WAID?"
	}

JSON send: {
  "status": 0
}
	Error: **no required field: "authwaid"**

3 The TetraJS library

TetraJS is a JavaScript library that provides a high level abstraction of the Telium Tetra services. The TetraJS library is intended for use in Tetra Web Applications.

The TetraJS library extends JavaScript with a new object called "tetra" attached to the JavaScript "window" object and represents the terminal in the Web Application.

TetraJS is designed to be compatible with AMD and CommonJS implementations.

TetraJS implements methods to support the call of service interfaces as well as the reception of service events, simplifying and somehow standardizing the interface with Telium Tetra services.

The main purpose behind the TetraJS library design is to abstract the details of the Tetra terminal internals (like URLs, SSEs, …) by providing web developers with an "easy to use" library that hides the details of interfacing with Telium Tetra services, and allowing them to focus exclusively on the WebApp logic and design.

A particular feature of interest is the management by the TetraJS library of the disconnection from Telium Tetra services when the WebApp is stopped by the user. When such an asynchronous stop occurs, it is very important that any service still open by the WebApp be properly disconnected so that it can be made available to other applications.

The tetraJS library is delivered in two versions (minified for the final production builds and non-minified for debug). It can be imported in the WebApp JavaScript sources as is.

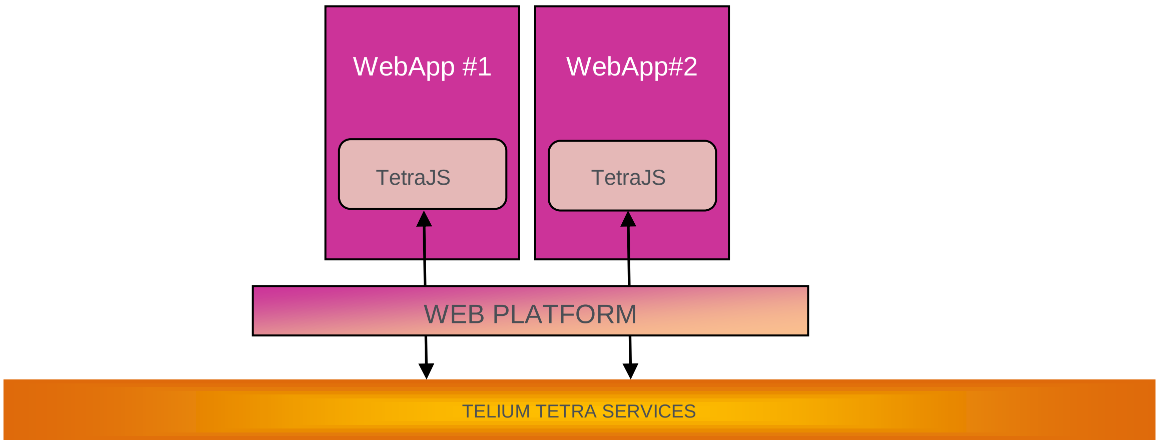

Fig 3.1 TetraJS library manages the WebApp interface with Telium Tetra services

To date, tetraJS library will expose five objects:

- **tetra:** this object represents the terminal in the JavaScript code.
- **tetra.version:** this object holds the Version-Release-Patch of the tetraJS library.
- **tetra.weblet:** this object represents the weblet running in the terminal.

- **<service> :** this object represents an instance of a running Telium Tetra service in the terminal. It is necessarily instantiated by mean of the tetra.service() method.
- **<waas> :** this object represents an instance of a running web application as a service. It is necessarily instantiated by mean of the tetra.waas() method.

Notice: the notation < xxxxx > represents the element (object, parameter, others…) whose name is specified between the '<' symbol and the '>' symbol.

The table below lists the methods and objects exposed or created by the library.

Object	Method	Description	TetraJS min version
tetra	.setup(…)	Configure the general settings of Tetra Web Platform	0.8.0
tetra	.lookup(…)	Lists all the services that expose a given interface name	0.8.0
tetra	.service(…)	Creates a new **<service>** object in JavaScript that will represent an instance of a Telium Tetra service	0.8.0
tetra	.hide()	Hides the Web Platform environment – deprecated - replaced with hideWP	0.8.0
tetra	.show()	Request the display focus	0.8.0
tetra	.hideWP()	Hides the Web Platform environment	1.4.1
tetra	.showWP()	Shows the Web Platform environment	1.4.1
tetra.weblet		Object that represents the weblet	0.8.0
tetra.weblet	.on(…)	Adds event listeners to react on general event occurrence	0.8.0
tetra.weblet	.off(…)	Removes event listeners	0.8.0
tetra.weblet	.show()	Requests the focus for the weblet	0.8.0
tetra.weblet	.hide()	Releases the weblet focus	0.8.0
tetra.weblet	.notify()	Displays a notification on the application icon at desktop level	1.2.0
<service>	.connect()	Connects the WebApp as a client to a new instance of the service	0.8.0
<service>	.disconnect()	Disconnects the WebApp from the service	0.8.0
<service>	.call(…)	Calls an interface method of a given service	0.8.0
<service>	.open()	Opens a SSE connection with the Web Platform to receive service events.	0.8.0
<service>	.on(…)	Adds event listeners on service event occurrence	0.8.0
<service>	.off(…)	Removes service event listeners	0.8.0
<service>	.close()	Closes the SSE connection	0.8.0
<service>	.destroy()	Destroys the **<service>** object	0.8.0
<service>	.reset()	Reset the queue of service methods	0.8.0
<waas>	.on()	Register a callback for the answer to a request	1.3.0
<waas>.on()	.sendResponse()	Send a response within the registered callback	1.3.0
<waas>	.start()	Start the WaaS service	1.3.0
<waas>	.sendEvent()	Send a message event to all client	1.3.0

Table 3.1 Methods and objects exposed by the tetraJS library

3_1 TetraJS general features

3_1_1 General concept of Tetra

TetraJS follows two main principles from JavaScript: the “chaining” and the “promises”.

The chaining approach allows executing a sequence of methods without having to repeat the object to which they apply. The resulting code is reduced and therefore easier to understand.

The promise approach, which has been standardized in ECMAScript 6, allows to process AJAX calls one after another (avoiding their execution in parallel and therefore the need for callbacks).

Hereafter an example of chaining mixing promises and other calls:

```
<service>
.reset()
.connect()
.call(…)
.success(…)
.error(…)
.disconnect()
```

3_1_2 Common methods parameters

setup(), lookup() , service(), connect(), call(), and disconnect() methods all have common optional parameters that can be modified to override the defaults settings of tetra or service.

```
{
   requestDelay:0,   // delay after previous request
   requestTimeout:0, // general ajax timeout
   then:’both’,      // overrides the “then” property
                     // Value can be:
                     // ‘resolved’: promise is executed if previous promise succeeded.
                     // ‘rejected’: promise is executed if previous promise failed.
                     // ‘both’: promise is executed whatever the previous promise result
   promise:false     // This parameter creates a new promise, same as calling .reset() before
                        calling the method.
   error:function(){},  // Method call on request error,
   success:function(){} // Method call on request success,
}
```

3_2 "tetra" Methods

3_2_1 Configure Gateway parameters: tetra.setup()

Use tetra.setup() method to get the current configuration or set a new configuration value.

To date, the only parameter handled by this method is gatewayTimeout which specifies the timeout applied by the terminal when connecting to a remote host.

tetra.setup(…)

Description:
This method allows to retrieve the current tetra library configuration settings and to change them individually.
The configuration settings apply to the whole tetra library.

		Parameter	Type	Description
`tetra.setup({[data:{ gatewayTimeout:<gatewayTimeout>, requestTimeout:<requestTimeout>, requestDelay:<requestDelay> ]},[<extraParameters>]});`	Request	*<gatewayTimeout>*	Number	[Optional] Timeout (in ms) applied for external requests.
		<requestTimeout>	Number	[Optional] Timeout (in ms) applied for internal requests (services on the local terminal)
		<requestDelay>	Number	[Optional] Delay between two subsequent requests.
		<extraParameters>	JSON Object	[Optional] Common promises properties
	Response	`{ gatewayTimeout:<new_timeout>, requestTimeout:<requestTimeout>, requestDelay:<requestDelay> }`		

Examples :

```
tetra.setup()
.success(callBack); // retrieves the current config

tetra.setup({data:{gatewayTimeout:10}})
.success(callBack); // Set gateway timeout to 10 seconds
```

ingenico GROUP

3_2_2 Lookup for a service: tetra.lookup(…)

The tetra.lookup(…) method allows to retrieve the list of all the Tetra Telium services that implement a given interface.

tetra.lookup(…)				
Description: Lookup for services exposing the interface provided as a parameter.				
`tetra.lookup(<interfaceName>,[<extraParameters>])`	Request	**Parameter**	**Type**	**Description**
		`<interfaceName>`	String	Concatenation of the service namespace with the interface name
		`<extraParameters>`	JSON Object	[Optional] Common promises properties

	Response	`{`
		`  [`
		`"services[0].id":"local.device.buzzer",`
		`"services[1].id":"pinpad.device.buzzer"`
		`  ]`
		`}`		

Table 3.2 tetraJS “lookup” method API

Example: Looking for the Buzzer interface

```
tetra.lookup("ingenico.device.buzzer.Buzzer")
.success(function(response){
    // log the service name
    console.log(response.services[0].id);
});
```

3_2_3 Create a service object: tetra.service(…)

The tetra.service(…) method creates an object representing a Tetra Telium service.

tetra.service(…)				
Description: Creates a JavaScript object representing a Telium Tetra service.				
`tetra.service({` `[namespace:<namespaceName>],` `service:<serviceName>,` `[requestDelay:0,` `requestTimeout:0]` `})`	Request	**Parameter**	**Type**	**Description**
		<namespaceName>	String	[Optional] Namespace as specified in the service documentation
		<serviceName>	String	The instance name of the service running in the terminal. If the service instance name is not known, it can be discovered first using the `tetra.lookup` method.
		requestDelay	Number	[Optional] Delay between consecutive requests.
		requestTimeout	Number	[Optional] Request timeout.
	Response	–		

Table 3.3 tetraJS "service" method API

Example: Create an object to hold the representation of the buzzer service local instance

```
var buzzer = tetra.service({
    namespace: ingenico.device.buzzer,
    service: local.device.buzzer
});
```

3_2_4 Configure Response and Request bytes formats

The format of 'bytes' data exchanged with a service can be configured using a "formats" object.

```
tetra.service({
     service: SERVICE_NAME
     namespace: NAMESPACE,
     formats: {
       'MESSAGE_NAME.PARAMETER_NAME':'FORMAT'
    }
})
```

Where “formats” can take one of the following values:

- hex_array (default) – array of hex string values, for example: "byte_field": ["69", "AB", "00"]
- hex_string – hexadecimal string, for example: "byte_field": "69AB00"
- int_array – array of integers, for example: "byte_field": [105, 171, 0]
- base64 – base64 string
- tlv – TLV tree, for example:

```
"byte_field": [
    {"tag" : "0x9F02",  "length" : 6,  "data" : "0002"},
    {"tag" : "0x5F2A", "length" : 2, "data" : "0978" }
]
```

3_2_5 Request the display focus: tetra.show()

Typically, this method must be used after a payment when the service originally called by a weblet returns.

Using this method the weblet can request the focus to the display layer manager in order to have the Web Platform environment shown on the foreground.

When the javascript visibility change event is triggered, the weblet can then display the result of the transaction.

`tetra.show()`
Description: This method brings the display layer of the Web Platform environment to the foreground.
`tetra.show()`

Table 3.4 tetraJS "show" method API

Example: The following code displays the Web Platform environment.

```
tetra.show();
```

3_2_6 Request to be hidden: tetra.hide() - deprecated

This method is kept present in the library but has no effect.

This method was typically used when a WebApp launches a payment transaction in order to bring the native Payment application UI to the foreground.
Following a change of the terminal architecture, the payment application can now get the focus automatically.

3_2_7 Hide the Web Platform environment: tetra.hideWP()

The tetra.hideWP() method allows weblets to hide the Web Platform display layer, thereby making the native Telium display layer (GOAL) visible.

! This method should not be used in a standard web application. It is present for specific use cases.

tetra.hideWP()
Description: This method hides the display layer of the Web Platform environment and shows the native Telium display environment (GOAL).
`tetra.hideWP()`

Table 3.5 tetraJS “hideWP” method API

Example: The following code hides the Web Platform display layer.

```
tetra.hideWP();
```

3_2_8 Show the Web Platform environment: tetra.showWP()

The tetra.showWP() method allows weblets to show the Web Platform display layer.
It will display the last weblet that had the focus regardless of wich web application calls this method.

! This method should not be used in a standard web application. It is present for specific use cases.

tetra.showWP()
Description: This method brings the display layer of the Web Platform environment to the foreground.
`tetra.showWP()`

Table 3.6 tetraJS "showWP" method API

Example: The following code displays the Web Platform environment.

```
tetra.showWP();
```

3_2_9 Check service started: tetra.checkServices

This method allows to easily check if a list of services is already registered in the system and can be called.
It can be used to wait for the dependency services to be all started.

This does necessarily mean that all functions of the service are ready to execute incoming requests but simply that the service itself is accessible. It is up to the service to provide the appropriate answer depending on its internal state.

tetra.checkServices()				
Description: This method checks a services list if they are already started.				
`tetra.checkServices([<interfaceName>], [<extraParameters>])`	Request	**Parameter**	**Type**	**Description**
		`<interfaceName>`	String array	Concatenation of the service namespace with the interface name

		`<extraParameters>`	JSON Object	[Optional]
				`{`
				`delay:<int>`
				`try:<int>`
				`then:<string>`
				`}`

	Response	`None,`
		`Depending on the result, success or error callback are called`		

`<extraParameters>` is a JSON object having the following structure:

```
{
    delay:<int> // The optional "delay" parameter specifies a delay in ms between each check
    try:<int>  //number of try for each service
    then:<string> // overrides the "then" common property
}
```

Table 3.7 tetraJS "checkServices" method API

Example: The following code displays the Web Platform environment.

```
tetra.checkServices(['ingenico.desktopenv.Settings','ingenico.transaction.Manager'],
        {delay: 1000, try: 3})
    .success(function () {console.log("success")})
    .error(function () {console.log("error")});
```

3_3 “tetra.weblet” methods

These methods apply to the tetra.weblet object that represent the running weblet.

3_3_1 Request the display focus: tetra.weblet.show()

The tetra.weblet.show() method allows a weblet to claim the display focus.
The weblet should monitor the :shown event to detect when it is effectively being given the focus).

`tetra.weblet.show()`				
Description: This method is used by weblets to request the display focus.				
`tetra.weblet.show()`	Request	**Parameter**	**Type**	**Description**
		-	-	-
	Response	-		

Table 3.6 tetraJS “weblet.show” method API

Example: The following code claims the focus on display for the weblet.

```
tetra.weblet.show();
```

3_3_2 Release the display focus: tetra.weblet.hide()

The tetra.weblet.hide() method allows a weblet to release the display focus.

tetra.weblet.hide()				
Description: This method is used by weblets to release the display focus.				
`tetra.weblet.hide()`	Request	**Parameter**	**Type**	**Description**
		-	-	-
	Response	-		

Table 3.7 tetraJS "weblet.hide" method API

Example: The following code removes the requesting weblet from the display.

```
tetra.weblet.hide();
```

3_3_3 Listen to weblet events: tetra.weblet.on(…)

The tetra.weblet.on() method allows a weblet to listen to Web Platform originated events as well as to custom events.

This method allows to synchronise treatments following the occurrence of an event.

Web Platform originated events

- **“close”** : weblet is about to be closed
- **“show”** : weblet has just recover the display focus
- **“hide”** : weblet has just lost the display focus
- **“wakeup”** : event called following the terminal wake up (typically a MOVE terminal resuming from sleep after some activity has been detected on the accelerometer, the keypad or the touchscreen)

Custom events

Could be any custom event defined by a string.
Such event can be triggered using the tetra.weblet.trigger(…) method.

tetra.weblet.on()				
Description: This method is to add events listeners.				

`tetra.weblet.on(`
`<eventName>,`
`<callback>,`
`[<context>])`
	Request	**Parameter**	**Type**	**Description**

		<eventName>	String	The event name: “close”,”show” or “hide”
		<callback>	Function	The function to be triggered upon the event occurence.
		<context>	Object	[Optional] Object attached to the listener if different from tetra.weblet (redefines “this” keyword)

Table 3.8 tetraJS “weblet.on” method API

Example: The following code will be called before application is closed.

```
tetra.weblet.on(“close”, function() {
    //TODO
});
```

3_3_4 Trigger a weblet event

This method allows to fire a custom event.

tetra.weblet.trigger(…)			
Description: This method allows to trigger custom events .			
tetra.weblet.trigger(<eventName>)	**Parameter**	**Type**	**Description**
	<eventName>	String Array RegExp	[Optional] Event name, list of events, or regular expression

Table 3.9 tetraJS "weblet.trigger" method API

Example: tetra.weblet.trigger(…) can take different parameter types as input as shown below:

```
// Trigger a specific event
tetra.weblet.trigger('event');

// Trigger a list of events
tetra.weblet.trigger(['event_1','event_2']);

// Trigger all events that start with string provided as RegExp expression
// For example trigger all the events that begin with "event:"
tetra.weblet.trigger(/^event:/);

// Trigger all events ( trigger any event that is currently listened).
tetra.weblet.trigger();
```

3_3_5 Remove weblet event listeners: tetra.weblet.off(…)

Stop listening to Web Platform or custom events.

`tetra.weblet.off(…)`			
Description: This method removes events listeners and event handlers.			
`tetra.weblet.off( [<eventName>], <callback>, <context>])`	**Parameter**	**Type**	**Description**
	`<eventName>`	String	[Optional] Event name, list of events, or regular expression
	`<callback>`	Function	[Optional] If provided, only remove the association of the given callback handler to the specified event.
	`<context>`	Object	[Optional] Object attached to the listener if different from tetra.weblet (redefines “this” keyword)

Table 3.10 tetraJS “weblet.off” method API

Example: tetra.weblet.off() can take different parameter types as input as shown below:

```
// Remove a specific event handler
tetra.weblet.off('event', handler,[context]);

// Remove a specific event
tetra.weblet.off('event');

// Remove a list of events
tetra.weblet.off(['event_1','event_2']);

// Remove all events that start with string provided as RegExp expression
// For example remove all the events that begin with “event:”
tetra.weblet.off(/^event:/);

// Remove all events
tetra.weblet.off();
```

3_3_6 Display a notification on the weblet icon on the Web desktop: tetra.weblet.notify(…)

A badge will be added on the weblet icon to notify that the weblet has something to push to the user.

tetra.weblet.notify(…)

Description:

This method allows a weblet to display a notification over its icon in Web Desktop

```
tetra.weblet.notify({
[badge:<badge>,
 count:<count>,
 save:<save>,
 id:<id>]
})
```

Parameter	Type	Description
`<badge>`	String	[Optional] Badge type you want to display For now "`default`" meaning use the default badge or "`none`" meaning no badge are supported.
`<count>`	Integer	[Optional] Number of displayed notifications. RFU (Do not comply with default badge)
`<save>`	Boolean	[Optional] At, "`true`" put in place the notification At "`false`", remove the notification Default is "`false`"
`<id>`	String	[Optional] If non present, targets the icon of the weblet object itself. Otherwise targets the weblet whose id is provided.

Table 3.11 tetraJS "weblet.notify" method API

Example: tetra.weblet.notify() can take different parameter types as input as shown below:

```
// Remove any badge notification applied on the weblet icon
tetra.weblet.notify({badge:"none"});

// Remove any badge notification applied on the icon of a given weblet
tetra.weblet.notify({badge:"none",id:"54B44421-WebAppName"});

// Set up a badge (here the default type) on the icon of the present weblet
tetra.weblet.notify({save:true});
```

3_4 "<service>" interface methods

3_4_1 Connect to a service: <service>.connect()

The ***<service>.connect()*** method allows a weblet to instantiate a Telium Tetra service and attach it to a javascript <service> object.

The <service> object must have been created first thanks to the tetra.service() method and exist in the javascript code.

<service>.connect()
Description: This method connects and attach a javascript <service> object to a real instance running in Telium Tetra.
`<service>.connect();`

Table 3.12 tetraJS "<service>.connect" method API

Example: The following code connects the weblet to the buzzer service as a client.

```
// First of all: create a buzzer service object
var buzzer = tetra.service({
    namespace: ingenico.device.buzzer,
    service: local.device.buzzer
});
//Then connects to the buzzer service
buzzer.connect();
```

3_4_2 Reset a service: <service>.reset()

The ***<service>.reset()*** method allows to flush a queue of chained promises.

The <service> object must have been created first using the tetra.service() method and exist in the javascript code.

<service>.reset()
Description: This method flushes a queue of chained promises attached to the <service> object.
`<service>.reset();`

Table 3.13 tetraJS "<service>.connect" method API

Example:

```
service
.connect()
.call(…)
.success(function() {
   return this
.reset()     // Allows to launch a new call after the previous
             //if successful
.call(…);              }
)
.disconnect();
```

3_4_3 Disconnect from a service: <service>.disconnect()

The <service>.disconnect() method allows a weblet to disconnect from the Tetra service instance attached to the <service> object.

After the call, the instance of the Tetra service is released. The object <service> still has an existence in the javascript code but is not attached any longer to any Tetra service.

<service>.disconnect()
Description: This method detach and disconnects and a javascript [service] object from its instance running in Telium Tetra.
`<service>.disconnect();`

Table 3.14 tetraJS "<service>.disconnect" method API

Example: the following code connects the weblet to the buzzer service as a client and then disconnects from it.

```
// First of all: create a buzzer service object
var buzzer = tetra.service({
    namespace: ingenico.device.buzzer,
    service: local.device.buzzer
});
//Then connects to the buzzer service
buzzer.connect();

//Finally dicconnects from the buzzer service
buzzer.disconnect();
```

3_4_4 Calling service interface methods: <service>.call(…)

The ***<service>.call(…)*** method allows a weblet to call an interface method exposed by that service.

<service>.call(…)				
Description: This method calls an interface methods exposed by the <service> object.				

`<service>.call(`
`<methodName>,`
`<extraParameters>`
`)`
	Request	**Parameter**	**Type**	**Description**

		`<methodName>`	String	Method name as specified in the documentation of Telium Tetra services. (See section 4.)

		`<extraParameters>`	JSON object	`{`
				`hide:<Boolean>`
				`expect:<CallbackName>`
				`data:<JSON object>`
				`}`
				See detailed description below …

	Response	`Usually a JSON object with variable contents depending on the called method.`
		`{`
		`...`
		`}`		

Table 3.15 tetraJS "<service>.call" method API

`<extraParameters>` is a JSON object having the following structure:

```
{
    // The optional "hide" parameter specifies whether the Web Platform
    // environment should be hidden (true) or not (false) during the
    // service call. Default value for hide is false.
    hide:true, // For example, this hides the Web Platform environment

    // The optional "expect" parameter provides a callback function that will evaluate the received
    // response against a predetermined set of expected values. By default there is not any predefined
    // callback. Callback function returns a Boolean: true for success and false for fail.
    expect:function(r){

                    // Evaluate the result of a card swipe operation
                    return  r.swipeResult!=4292967294 &&
                            r.swipeResult!=4293967295 &&
                            r.swipeResult!=4293967196;
                }

    // The optional "data" parameter is a JSON object that gathers the specific parameters related to
    the Tetra service method being called.
    data:{
            priority:1
        }
}
```

Example: the following code sounds forever a Beep at 100 Hz, and keeps Web Platform environment in the foreground.

```
buzzer.call(“Beep”,{
             data:{volume:1,
             frequency:  100,
             duration:-1
             }

});
```

3_4_5 Destroy a service

You can call ***<service>.destroy()*** in order to delete the <service> javascript object. If that <service> object is attached to Telium Tetra service instance, it disconnects first from the Telium Tetra service prior deleting the object.

<service>.destroy()
Description: This method destroys the javascript <service> object.
`<service>.destroy()`

Table 3.16 tetraJS “<service>.destroy” method API

Example:

```
// First of all: create a buzzer service object
var buzzer = tetra.service({
    namespace: ingenico.device.buzzer,
    service: local.device.buzzer
});


// Destroys the buzzer object previously instantiated
buzzer.destroy();
```

3_5 <service>" events

3_5_1 Opening a channel to receive Tetra service event notification

The ***<service>.open()*** method allows a weblet to open a communication channel in order to receive event notifications from a Telium Tetra service.

<service>.open()
Description: This method opens a communication channel to receive the event notifications from the Telium Tetra service attached to the <service> object.
`<service>.open()`

Table 3.17 tetraJS "<service>.open" method API

Example: The following code connects to the Smart Card Reader service and opens the communication channel to be notified of events.

```
tetra.service({
    namespace:"ingenico.device.chip",
    service:"local.device.chip0"
})
.connect()
.open()
```

3_5_2 Catching a Telium Tetra service event notification

The <service>.on() method allows a weblet to specify the handler to be called upon the occurrence of a given Telium Tetra service event.

`<service>.on(…)`				
Description: This method add events listeners related to Telium Tetra service events.				

`<service>.on(`
`<eventName>,`
`<callback>,`
`[<context>]`
`)`
	Request	**Parameter**	**Type**	**Description**

		`<eventName>`	String	The event name as described in Telium Tetra service doc (See section 3_7 .)
		`<callback>`	Function	Defines the callback handler to be called upon event occurrence. …
		`[<context>]`	Object	[Optional] Allows to redefine the context of "this" keyword in callback method, if necessary).By default this refers to the service object..

	Response	Usually a JSON object with variable contents depending on the triggered event.
		`{`
		`...`
		`}`		

Table 3.18 tetraJS "<service>.on" method API

Example: The following code will listen to the Telium Tetra Chip service event: `ChipDetectedEvent`.

```
<service>.on('ChipDetectedEvent',function(response){
    // Add your code here
});
```

3_5_3 Removing a Telium Tetra event listener

The <service>.off() method allows a weblet to stop listening to some Tetra Service event notification.

`<service>.off(…)`				
Description: This method remove listeners related to Telium Tetra service events.				
`<service>.off( <eventName>, [<callback>,<context>] )`	Request	**Parameter**	**Type**	**Description**
		`<eventName>`	String	The event name as described in Telium Tetra service doc (See section 3_7 .)
		`<callback>`	Function	[Optional] If provided, only remove the association of the given callback to the specified event.
		`[<context>]`	Object	[Optional] Allows to redefine the context of "this" keyword in callback method, if necessary).By default this refers to the service object..
	Response	`{ ... }`		

Table 3.19 tetraJS "<service>.off" method API

Example:

```
// Remove a specific event handler
service.off('event', handler,[context]);

// Remove a specific event
service.off('event');

// Remove a list of events
service.off(['event','event']);

// Remove all events that start with event using REGEX
service.off(/^event:/);


// Remove all events
service.off();
```

3_5_4 Closing communication between client and events source server

The <service>.close() method allows a weblet to close the communication channel used to listen to Telium Tetra service event notification.

When this method is called, any pending event listeners are stopped (off) prior to effectively closing the communication channel.

<service>.close()
Description: This method close the communication channel for Telium Tetra event listeners.
`<service>.close()`

Table 3.20 tetraJS "<service>.close" method API

Example:

```
<service>.close();
```

3_6 Error management

TetraJS allows error handling using various keywords and sequences. You'll find them in the table below:

METHOD	CODE

Using "then" method	`.call("Beep")`
	`.then(`
	`    function(response) { // Success 200`
	`        // do something …`
	`    },`
	`    function(e) { // Error 403,404,500 …`
	`        // do something …`
	`    }`
	`);`

Using two "then" methods	`.call("Beep")`
	`.then(function(response) { // Success 200`
	`    // do something …`
	`})`
	`.then(undefined, function(e) { // Error 403,404,500 …`
	`    // do something …`
	`});`

Using an alias	`.call("Beep")`
	`.success(function(response) { // Success 200`
	`    // do something …`
	`})`
	`.catch(function(e) { // Error 403,404,500 …`
	`    // do something …`
	`});`

Using another alias	`.call("Beep")`
	`.success(function(response) { // Success 200`
	`    // do something …`
	`})`
	`.error(function(e) { // Error 403,404,500 …`
	`    // do something …`
	`});`

Using callbacks	`call("Beep", {`
	`    success: function() { // Success 200`
	`        // do something …`
	`    },`
	`    error: function() { // Error 403,404,500 …`
	`        // do something …`
	`    }`
	`});`

Table 3.21 Handling errors with tetraJS

Errors have to be managed at two different levels:

- HTTP protocol level
- Service functionality level

3_6_1 Handling HTTP errors with tetraJS

Let’s illustrate HTTP error handling with the example below:

```
// First of all: create a buzzer service object
var buzzer = tetra.service({
    namespace: ingenico.device.buzzer,
    service: local.device.buzzer
});

// Then connects to the buzzer service
buzzer.connect();

// Sound a beep forever at 100 Hz
buzzer.call(“Beep”,
    {
       data:{
         volume:1,
         frequency:  100,
         duration:-1
    }
});

//Let’s assume the call failed for some reasons …
buzzer.then(undefined,
     function(error){

       // Write to the console the HTTP status code
       console.log(error.status); // For example 500

       // Write to the console the HTTP status text
       console.log(error.msg); // For example Internal Server Error

        // Write to the console the HTTP status text
       console.log(error.response); // For exemple: “JSON to protobuf conversion error
                                    // [platform errcode -1000000]”
    }
);

//Finally dicconnects from the buzzer service
buzzer.disconnect();
```

The table below lists the interpretation to be given to HTTP error codes for AJAX regular calls and for tetra service calls.

HTTP status	Meaning for general request URI (AJAX calls)	Meaning for Telium Tetra services (tetra service calls)
200: OK	The request has been successfully executed.	The request has been successfully executed. Any data/status returned by the service is valid and can be exploited.
400: BAD REQUEST	The request could not be interpreted by the server due to malformed syntax. The request shouldn't be renewed without any modifications.	The request includes some unexpected content (wrong parameter).
403: FORBIDDEN	The server understood the request but is refusing to fulfill it because it does not match with the whitelist.	Web Application is not granted the access to this resource because of insufficient access rights.
404: NOT FOUND	The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent. For example, the resource couldn't be found in the WebApp package or on the remote server.	The service interface could not be found or is unavailable at the moment.
405: METHOD NOT ALLOWED	The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response shall include an “Allow” header containing a list of valid methods for the requested resource.	
415: UNSUPPORTED MEDIA	The server is refusing to service the request because the format of the requested resource does not have a MIME type allowed by Web Platform. Usually this code will be issued when external data returned by a remote server break the rules. In the response, a text page sent by Web Platform reminds the list of the only supported MIME types for external data.	N/A
500: INTERNAL SERVER ERROR		Internal Web Platform error.
501: NOT IMPLEMENTED	The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.	The specified interface method does not exist or is not supported by the service.
502: BAD GATEWAY	The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.	N/A
503: SERVICE UNAVAILABLE	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.	The service couldn't be accessed because all its instances are busy.
504: GATEWAY TIMEOUT		

Table 3.22 HTTP status

3_6_2 Handling Telium Tetra service errors with tetraJS

Let's assume

```
var swipeService = tetra.service({ // Include swipe service
    service: 'local.device.swipe0',
    namespace: 'ingenico.device.swipe'
});

swipeService
.reset() // Reset service
.call('GetResult', { // Call GetResult
   requetDelay: 0,
   expect: function (r) { // Test that response is correct
   return r.swipeResult != 4292967294 && r.swipeResult != 4293967295
   }
})
.success(function (r) {
   console.log('Card swiped');

   swipeService
            .reset()      // Reset service
            .call("Stop") // Stop service
            .disconnect(); // Disconnect from service and do transaction
           .success(doTransaction)
    })
    .error(function (e) {
        detectSwipe(); // Polling swipe
    });
```

3_7 Web application as Service – WaaS

A web application can provide service to other web or native applications. To achieve this using tetraJS:

- First create a WaaS object containing the global description
- Next, register a callback for each message to be answered by the WaaS. Permissions and bytes array format should be provided at the same time,
- Last, start the WaaS service

In tetraJS, Web application as a Service is accessed using the object tetra.waas.

3_7_1 Creating a new Web application as Service

The tetra.waas(…) method creates an object representing a web application as service.

tetra.waas(…)				
Description: Creates a JavaScript object representing a web application as service.				
`tetra.waas(interfaceName)`	Request	**Parameter**	**Type**	**Description**
		`<interfaceName>`	String	Interface name proposed by the WaaS instance

3_7_2 Registering a callback to answer a request

The <waas>.on() method allows a weblet to register a callback to process the call to a method on the WaaS service.

`<waas>.on(…)`				
Description: This method add a callback that will treat a call to the rpc.				

`<waas>.on(`
`<methodName>,`
`<methodConfig>,`
`<callback>,`
`)`
	Request	**Parameter**	**Type**	**Description**

		`<methodName>`	String	Method name corresponding to the one defined in the service message description

		`<methodConfig>`	Object	[Optional]
				JSON object allowing to redefine permission and format for this specific method
				`{`
				`permission:[<string>]`
				`format:{<message.field>:`
				`"format"}`
				`}`
				More details below

		`<callback>`	Function	Function that will be called when a client will request the methodName.

`<methodConfig>` is a JSON object having the following structure:

```
    {
        // The optional “permission” parameter specifies permission to accept client call to this
methodName

        // VERTICAL corresponds to a native or web application with VERTICAL permission
        // PAYMENT corresponds to a native or web application with PAYMENT permission
        // ALL_TELIUM_PLATFORM corresponds to a native Telium component provided by the Telium SDK
        // ALL_WEB_PLATFORM corresponds to a native WebPlatform component provided by the WebPlatform addon

        // by default, service will accept any of those clients VERTICAL, PAYMENT, ALL_TELIUM_PLATFORM,
        ALL_WEB_PLATFORM.
        permission:['VERTICAL', 'PAYMENT’], // For example, will accept any vertical or payment application

        // The optional “format” parameter provides a mean to configure a bytes field in the message.
        format: {
        'MESSAGE_NAME.PARAMETER_NAME':’FORMAT’
    }
```

For more details on possible format values, refer to paragraph *3_2_4 Configure Response and Request bytes formats*.

3_7_3 Starting the WaaS service

The <waas>.start() method starts the WaaS service. Only after this point will the weblet answer to registered rpc method.

`<waas>.start()`				
Description: This method start the WaaS service.				
`start()`	Request	**Parameter**	**Type**	**Description**

3_7_4 Send the response to the client

The <waas>.on() callback context (this) provides a sendResponse() method that will send the answer back to the client.

`sendResponse()`				
Description: This method sends the response to the client.				
`this.sendResponse(<dataResponse>, <callback>, <error>)`	Response	**Parameter**	**Type**	**Description**
		`<dataResponse>`	Object	[Optional] JSON object containing the response. If not provided an empty JSON response is sent
		`callback`	Function	[Optional] Callback function called in case of success
		`error`	Function	[Optional] Callback function called in case of error Request could be in error when JSON data is not correct

3_7_5 Send event to all client

The <waas>.sendEvent() method sends aan event message to all connected clients.

sendResponse()				
Description: This method send a message to the client.				
`this.sendResponse(<eventName>, <dataResponse>, <callback>, <error>)`	Response	**Parameter**	**Type**	**Description**
		`<eventName>`	String	eventName is the message name like it is described
		`<dataResponse>`	Object	[Optional] JSON object containing the response. If not provided an empty JSON response is sent
		`callback`	Function	[Optional] Callback function called in case of success
		`error`	Function	[Optional] Callback function called in case of error Request could be in error when JSON data is not correct

4 Telium Tetra service methods and events

The following table lists the Telium Tetra services exposed to weblets together with the minimal version of the Telium SDK in which they were published.

Telium Tetra service	Interface name / Event name	Interface methods	Possible service instances	Minimal Telium SDK
Buzzer	ingenico.device.buzzer.Buzzer	Beep	local.device.buzzer	10.8.0
Smart Card Reader (SCR)	ingenico.device.chip.Chip	start	local.device.chip0	10.8.1
		getResult		10.8.1
		startRemove		10.8.1
		getRemoveResult		10.8.1
		stop		10.8.1
		isCardPresent		10.8.1
	ingenico.device.chip.ChipEmv	emvPowerOn		10.8.1
		emvPowerOff		10.8.1
		emvGetAtr		10.8.1
		emvReset		10.8.1
		emvApdu		10.8.1
	ingenico.device.chip.ChipDetectedEvent			10.8.1
	ingenico.device.chip.ChipRemovedEvent			10.8.1
Magnetic Stripe Reader (MSR)	ingenico.device.swipe.Swipe	start	local.device.swipe0	10.8.1
		getResult		10.8.1
		getData		10.8.1
		stop		10.8.1
	ingenico.device.swipe.SwipeDetectedEvent			10.8.1
Contactless Card Reader (NFC)	ingenico.device.contactless.ContactlessCard	startDetection	local.device.contacless0	10.8.1
		getDetectionResult		10.8.1
		startRemoval		10.8.1
		getRemovalResult		10.8.1
		Stop		10.8.1
		getUid		10.8.1
		fieldOff		10.8.1
		apdu		10.8.1
	ingenico.device.contactless.ClessDetectedEvent			10.8.1
	ingenico.device.contactless.ClessRemovedEvent			10.8.1
Transaction	Ingenico.transaction.Manager	manageTransaction	local.transaction.engine	10.8.0

The service interfaces are documented in what follows in a format that matches the TetraJS library ".call" API.

4_1 Buzzer

4_1_1 Buzzer.Beep(…)

Description
This method allows emitting a sound using the buzzer of the terminal.

Service namespace	Interface	Method
`ingenico.device.buzzer`	`Buzzer`	`Beep`

List of possible service instances
`local.device.buzzer`

TetraJS `.call`	

.call parameters	`data:{`
	`  'volume':<a>,`
	`  'frequency':<b>,`
	`  'duration':<c>`
	`}`
	
	with:
	`<a>` sound volume in % (0 – 100)
	`<b>` sound frequency in Hz (40 – 3000)
	`<c>` sound duration in 10 ms

TetraJS `.success`	
HTTP status	`200 = OK`
Buzzer.Beep(…) status	–

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_1_2 Buzzer sample code

TetraJS based weblet sample

The code below triggers a sound of 1000 Hz, at 50% of volume during 2 seconds

```
var tetra = require('../tetra.js');
var buzzerCfg = { 'frequency':1000,
                  'volume':50,
                  'duration':200
                };

var buzzer = tetra.service({
                             service: 'local.device.buzzer',
                             namespace: 'ingenico.device.buzzer'
                          });
buzzer
.connect()
.call('Beep',{data:buzzerCfg})
.disconnect();
```

4_2 Smart Card Reader (SCR)

4_2_1 Chip.start(…)

Description
This method launches the detection of a card insertion in a given smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`start`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'timeout':<a> //optional`
	`}`
	
	with:
	`<a>` timeout in ms

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.start(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_2 Chip.getResult()

Description
This method gets the result of a card insertion in the smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`getResult`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.getResult(…) status	`data:{`
	`  'return':<a>,`
	`  'chipResult':<b>`
	`}`
	
	with:
	`<a>` service call status
	`<b>`

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_3 Chip.startRemove(…)

Description
This method launches the detection of a card removal from a given smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`startRemove`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'timeout':<a> //optional`
	`}`
	
	with:
	`<a>` timeout in ms

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.startRemove(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_4 Chip.getRemoveResult()

Description
This method gets the result of a card removal from a given smart card reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`getRemovalResult`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.getResult(…) status	`data:{`
	`  'return':<a>,`
	`  'chipResult':<b>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>`

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_5 Chip.stop(…)

Description
This method stops the detection of a card insertion in a given smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`stop`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.stop(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_6 Chip.isCardPresent(…)

Description
This method tells if a card is currently inserted in the smart card reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`Chip`	`isCardPresent`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Chip.stop(…) status	`data:{`
	`  'return':<a>,`
	`  'chipPresent:<b>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>` 0: chip card is not present
	1: chip card is present

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_7 ChipEmv.emvPowerOn()

Description
This method powers up a card already inserted in a given smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`ChipEmv`	`emvPowerOn`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ChipEmv.emvPowerOn() status	`data:{`
	`  ‘return’:<a>`
	`  }`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_8 ChipEmv.emvPowerOff()

Description
This method powers down a card already inserted in a given smartcard reader.

Service namespace	Interface	Method
`ingenico.device.chip`	`ChipEmv`	`emvPowerOff`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`
ChipEmv.emvPowerOff() status	`data:{ 'return':<a> }` with: `<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_9 ChipEmv.emvReset()

Description
This method performs a reset on a card already inserted.

Service namespace	Interface	Method
`ingenico.device.chip`	`ChipEmv`	`emvReset`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ChipEmv.emvReset() status	`data:{`
	`  'return':<a>`
	`  }`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_10 ChipEmv.emvGetAtr()

Description
This method performs a reset on a card already inserted.

Service namespace	Interface	Method
`ingenico.device.chip`	`ChipEmv`	`emvGetAtr`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ChipEmv.emvPowerAtr() status	`data:{`
	`   'return':<a>,`
	`   'atr':<b>`
	`   }`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>` Array object enclosing ATR bytes

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_11 ChipEmv.emvApdu(…)

Description
This method sends an apdu to a card already inserted

Service namespace	Interface	Method
`ingenico.device.chip`	`ChipEmv`	`emvApdu`

List of possible service instances
`local.device.chip0`

TetraJS `.call`	

.call parameters	`data:{`
	`   'apducommand':<a>`
	`   }`
	
	with:
	`<a>` apdu command (array of up to 384 bytes)

TetraJS `.success`	
HTTP status	`200 = OK`

ChipEmv.emvApdu(…) status	`data:{`
	`   'return':<a>,`
	`   'apduResponse':<b>`
	`   }`
	
	with:
	`<a>` service call status (See list of returned status in Appendix B)
	`<b>` response provided by the card (array of up to 384 bytes including the response and the status bytes)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_2_12 Smart Card Reader sample codes

TetraJS based weblet sample

The code below will detect the card insertion and send an apdu command using event method

```
var chipService = tetra.service({ // Instantiate chip Service
            service: 'local.device.chip0',
            namespace: 'ingenico.device.chip',
            formats: {
                'EmvApduResponse.apduResponse': 'hex_string' // Request apduResponse property
from  EmvApduResponse message response to be in hexa string format
            }
        });

function sendCommand() {

    chipService
        .reset()
        .call('EmvPowerOn') // start sending apdu command
        .then(function (r) {
            log("Emv Power on started");
        }, function (e) {
            console.log(e);
        })
        .call('EmvApdu', { // Call EmvApdu
                expect: function (response) { // Service response should be equal to 9000 in
order to be correct
                    var apduResponse = response.apduResponse;
                    return apduResponse && apduResponse === "9000";
                },
                requestDelay: 250,
                data: {"apducommand": ["00", "A4", "00", "00", "02", "00", "02"]}
        })
        .then(function (r) {
                console.log(r);
          }, function (e) {
                console.log(e);
        })
       .call('StartRemove', {data: {timeout: 10000}}) // Call StartRemove
       .then(function (r) {
            console.log('Please eject your card');
       });
}
```

ingenico GROUP

```
/*** START LISTENNING ***/

chipService
.reset() // Reset service
.disconnect() // Disconnect from service if connected
.connect() // Connect to service
.close()  // Close SSE if opened
.open()  // Open SSE
.on('ChipDetectedEvent', function (r) {  // Listen to ChipDetectedEvent
   console.log('Card inserted');

   sendCommand();

})
.on('ChipRemovedEvent', function (r) { // Listen to ChipRemovedEvent
    console.log('Card ejected');

    chipService
              .reset() // Reset service
              .close(); // Close SSE
})
.call('Start', {data: {timeout: 10000}})  // Start detection first
.then(function (r) {
   console.log('Please insert your card');
}, function (e) {
   console.log(e.msg);
});
```

TetraJS based weblet sample

The code below will detect the card insertion and send an apdu command using polling method

```
var chipService = tetra.service({ // Instantiate chip Service
            service: 'local.device.chip0',
            namespace: 'ingenico.device.chip',
            formats: {
                'EmvApduResponse.apduResponse': 'hex_string' // Request apduResponse property
from  EmvApduResponse message response to be in hexa string format
            }
        });

function sendCommand() {

    chipService
        .reset()
        .call('EmvPowerOn') // start sending apdu command
        .then(function (r) {
            log("Emv Power on started");
        }, function (e) {
            console.log(e);
        })
        .call('EmvApdu', { // Call EmvApdu
                expect: function (response) { // Service response should be equal to 9000 in
order to be correct
                    var apduResponse = response.apduResponse;
                    return apduResponse && apduResponse === "9000";
                },
                requestDelay: 250,
                data: {"apducommand": ["00", "A4", "00", "00", "02", "00", "02"]}
         })
         .then(function (r) {
                console.log(r);
          }, function (e) {
                console.log(e);
        })
        .call("StartRemove", {success: detectEjection});

}

function detectInsertion() {

   console.log('Please insert your card');

   chipService
        .reset()
        .call('GetResult', { // Call Get result method
            expect: function (r) {
                return r.chipResult === 0;  // Except that chipResult is correct
            }
        })
        .success(sendCommand)
        .error(function (e) {
            detectInsertion(); // Polling detect insertion
        });
}

function detectEjection() {
   console.log('Please eject your card');

   chipService
       .reset() // Reset service
       .call('GetRemoveResult', { // Call GetRemoveResult
            expect: function (r) {
                return r.chipResult === 0; // Except that chipResult is correct
            }
       })
       .success(function (r) {
           console.log('Card ejected');

           chipService
               .reset() // Reset service
               .call("Stop") // Stop detection
               .disconnect(); // Disconnect from service

       })
       .error(function (e) {
                detectEjection(); // Polling ejection
       });
}

/***  START POLLING MODE ***/
chipService
    .reset() // Reset service
    .disconnect() // Disconnect from service if connected
    .connect() // Connect to service
    .call("Start",  {data:  {timeout:  10000},success:  detectInsertion});    //  Start  card
detection
```

4_3 Magnetic Stripe Reader (MSR)

4_3_1 Swipe.start(…)

Description
This method launches the detection of a card swipe for a given magnetic stripe reader.

Service namespace	Interface	Method
`ingenico.device.swipe`	`Swipe`	`start`

List of possible service instances
`local.device.swipe0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'timeout':<a> //optional`
	`}`
	
	with:
	`<a>` timeout in ms

TetraJS `.success`	
HTTP status	`200 = OK`

Swipe.start(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_3_2 Swipe.getResult()

Description
This method gets the result of a card swipe in the smartcard reader.

Service namespace	Interface	Method
`ingenico.device.swipe`	`Swipe`	`getResult`

List of possible service instances
`local.device.swipe0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Swipe.getResult() status	`data:{`
	`  'return':<a>,`
	`  'swipeResult':<b>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>`
	`STOPPED:` there's currently no card swipe detection in progress.
	`STARTED_NO_CARD:` detection is in progress but no card has been yet swiped.
	`STARTED_CARD:` a card was swiped and detection is not any longer in progress.

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_3_3 Swipe.getData()

Description
This method gets the data resulting from a card swipe in a given magnetic stripe reader.

Service namespace	Interface	Method
`ingenico.device.swipe`	`Swipe`	`getData`

List of possible service instances
`local.device.swipe0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

Swipe.getData() status	`data:{`
	`  'return':<a>,`
	`  'status1':<b>`
	`  'raw1':<c>`
	`  'iso1':<d>`
	`  'status2':<e>`
	`  'raw2':<f>`
	`  'iso2':<g>`
	`  'status3':<h>`
	`  'raw3':<i>`
	`  'iso3':<j>`
	`}`
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`ERR_UNAUTHORIZED:` service hasn't been opened or card hasn't been yet swiped
	`ERR_FAILED:` unavailable data or swipe resource access error
	`SUCCESS:` a card was successfully swiped
	`<b>` status of track#1
	`<c>` raw data from track#1
	`<d>` iso data from track#1
	`<e>` status of track#2
	`<f>` raw data from track#2
	`<g>` iso data from track#2
	`<h>` status of track#3
	`<i>` raw data from track#3
	`<j>` iso data from track#3

TetraJS .error	
HTTP Status	Refer to Table 3.22

4_3_4 Swipe.stop()

Description
This method stops card swipe detection for a given magnetic stripe reader

Service namespace	Interface	Method
`ingenico.device.swipe`	`Swipe`	`stop`

List of possible service instances
`local.device.swipe0`

TetraJS .call	
.call parameters	-

TetraJS .success	
HTTP status	`200 = OK`

Swipe.stop() status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`ERR_UNAUTHORIZED:` if swipe detection is already stopped
	`ERR_FAILED:` if stopping swipe detection failed
	`SUCCESS:` if swipe detection was correctly stopped.

TetraJS .error	
HTTP Status	Refer to Table 3.22

4_3_5 SwipeDetectedEvent()

Description
This event should be listened and handled by the weblet in order to react upon the swipe of a card.

Service namespace	Event
`ingenico.device.swipe`	`SwipeDetectedEvent`

List of possible service instances
`local.device.swipe0`

TetraJS `.on .off`	
.on .off parameters	`SwipeDetectedEvent,function(response){}`

4_3_6 Magnetic Stripe Reader sample code

TetraJS based weblet sample

The code below manages a card swipe using the polling method

```
var tetra = require('../tetra.js'), // Include Tetra Library
        swipeService = tetra.service({ // Include swipe service
            service: 'local.device.swipe0',
            namespace: 'ingenico.device.swipe'
        });

function detectSwipe() {

   console.log('Please swipe your card');

   swipeService
       .reset() // Reset service
       .call('GetResult', { // Call GetResult
                expect: function (r) { // Test that response is correct
                    return  r.swipeResult  !=  4292967294  &&  r.swipeResult  !=  4293967295  &&
r.swipeResult != 4293967196;
                }
       })
       .success(function (r) {
           console.log('Card swiped');

                swipeService
                    .reset() // Reset service
                    .call("Stop") // Stop service
                  .disconnect(); // Disconnect from service
```

```
      })
      .error(function (e) {
                detectSwipe(); // Polling swipe
      });
}

/*** START POLLING ***/
swipeService
   .reset() // Reset service
   .disconnect() // Disconnect from service
   .connect() // Connect to service
   .call("Start", {data: {timeout: 10000},success: detectSwipe}); // Call Start detection and
detect swipe
```

TetraJS based weblet sample

The code below manages a card swipe using the event method

```
var tetra = require('../tetra.js'), // Include Tetra Library
        swipeService = tetra.service({ // Include swipe service
            service: 'local.device.swipe0',
            namespace: 'ingenico.device.swipe'
        });

/*** START EVENT MODE ***/
swipeService
    .reset() // Reset service
    .disconnect() // Disconnect from service
    .connect() // Connect to service
    .close() // Close service
    .open() // Open service
    .on('SwipeDetectedEvent', function (r) { // Listen to SwipeDetectedEvent
        console.log('Card swiped');

        swipeService
            .reset() // Reset service
            .call("Stop") // Stop service
            .disconnect(); // Disconnect from service
     })
     .call('Start',{data:{timeout:10000}}) // call Start detection
     .then(function (r) {
          console.log('Please swipe your card');
     }, function (e) {
          console.log('Error occured');
     });
```

4_4 Contactless Card Reader (NFC)

4_4_1 ContactlessCard.startDetection(…)

Description
This method launches the detection of a contactless card for a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`startDetection`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'typesOfCard':<a> //optional`
	`  'timeout':<b>`
	`}`
	
	with:
	`<a>` default = 3 (ISO type A and B),
	`<b>` timeout in ms, if 0 is given the detection is attempted for a very small time

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.startDetection(…) status	`data:{`
	`  ‘return’:<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_2 ContactlessCard.getDetectionResult()

Description
This method gets the result following the tap of a contactless card for a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`getDetectionResult`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.getDetectionResult(…) status	`data:{`
	`  'return':<a>,`
	`  'type':<b>,`
	`  'model':<c>,`
	`  'uid':<d>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>` Card type detected.
	`<c>` Card model detected.
	`<d>` Card identifier.

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_3 ContactlessCard.startRemoval(…)

Description
This method launches the detection of a card removal from a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`startRemoval`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'timeout':<a> //optional`
	`}`
	
	with:
	`<a>` timeout in ms

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.startRemoval(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_4 ContactlessCard.getRemovalResult()

Description
This method gets the result of a card removal from a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`getRemovalResult`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.getRemovalResult() status	`data:{`
	 `'return':<a>`
	 `}`
	
	 with:
	 `<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_5 ContactlessCard.stop()

Description
This method stops the detection of a contactless card for a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`stop`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.stop(…) status	`data:{`
	`  'return':<a>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_6 ContactlessCard.getUid()

Description
This method gives the identifier of the card that was tapped on a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`getUid`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.getUid() status	`data:{`
	`  'return':<a>,`
	`  'uid':<b>`
	`}`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>` if successful array of bytes representing the card id

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_7 ContactlessCard.fieldOff()

Description
This method switches off the field on a given NFC reader.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`fieldOff`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	
.call parameters	-

TetraJS `.success`	
HTTP status	`200 = OK`
ContactlessCard.fieldOff() status	`data:{ 'return':<a> }` with: `<a>` service call status (check Telium Tetra status codes in appendix B)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_8 ContactlessCard.apdu(…)

Description
This method sends an apdu command to a contactless card.

Service namespace	Interface	Method
`ingenico.device.contactless`	`ContactlessCard`	`apdu`

List of possible service instances
`local.device.contactless0`

TetraJS `.call`	

.call parameters	`data:{`
	`  'command':<a>`
	`  }`
	
	with:
	`<a>` apdu command (array of bytes, up to 384 bytes)

TetraJS `.success`	
HTTP status	`200 = OK`

ContactlessCard.apdu() status	`data:{`
	`  'return':<a>,`
	`  'response':<b>`
	`  }`
	
	with:
	`<a>` service call status (check Telium Tetra status codes in appendix B)
	`<b>` response provided by the card (up to 384 bytes including the response and the status bytes)

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_4_9 Contactless Card Reader (NFC) sample code

TetraJS based weblet sample

The code below send an apdu command to the card using polling method

```
var tetra = require('../tetra.js'), // include tetra library

contactLessService = tetra.service({ // Instantiate service
      service: 'local.device.contactless0',
                          namespace: 'ingenico.device.contactless'
                       });

function getCardInformations() {

   var aidCommand = ["00", "A4", "04", "00", "07", "A0", "00", "00", "00", "04", "10", "10", "00"];
   var PPSEreponse = [];

   contactLessService
       .reset() // Reset service
       .call('GetUid', {requestDelay: 0}) // Call GetUid method
       .success(function (r) {
           console.log(r);
       })
       .call('Apdu', {
           data: { // Select PPSE
              command: ["00", //CLA
                        "A4", //INS  select file
                        "04", "00",// P1 P2
                        "0E",
                        "32", "50", "41", "59", "2E", "53", "59", "53", "2E", "44", "44", "46", "30", "31", "00"]
                }
            })
            .success(function (r) {
                PPSEreponse = r.response;
            })
            .call('Apdu', {  // Select AID
                data: {
                    command: aidCommand
                }
            })
            .success(function (r) {


                if (
                    (PPSEreponse[27] == "A0") &&
                    (PPSEreponse[28] == "00") &&
                    (PPSEreponse[29] == "00") &&
                    (PPSEreponse[30] == "00") &&
                    (PPSEreponse[31] == "03")
                ) {
                    aidCommand[9] = "03";
                    aidCommand[10] = PPSEreponse[32];
                    aidCommand[11] = PPSEreponse[33];
                } else if (
                    (PPSEreponse[27] == "A0") &&
                    (PPSEreponse[28] == "00") &&
                    (PPSEreponse[29] == "00") &&
                    (PPSEreponse[30] == "00") &&
                    (PPSEreponse[31] == "04")
                ) { // MC

                    aidCommand[10] = PPSEreponse[32];
                    aidCommand[11] = PPSEreponse[33];
                } else {
                    return;
                }

                contactLessService
                    .reset() // Reset service
                    .call('Apdu', {
                        data: { // Send APDU to the card
                            command: aidCommand
                        }, requestDelay: 0
                    })
                    .success(function (r) {
                        console.log(r);
                    })
       });
}

function detectContact() {
   console.log('Please tap your card');

   contactLessService
      .reset() // Reset service
      .call('GetDetectionResult', {requestDelay: 0}) // Call GetDetectionResult
      .success(function (r) {
           console.log("Card detected");
           return getCardInformations(); // Get card informations after swiped
       })
       .error(function (e) {
           detectContact(); // Polling contactless
       });
}

/*** START POLLING MODE ***/
contactLessService
    .reset() // Reset service
     .disconnect() // Disconnect from service
     .connect() // Connect to service
     .call("StartDetection", {data: {timeout: 10000}, success: detectContact}); // Call
StartDetection method
```

TetraJS based weblet sample

The code below send an apdu command to the card using event method

```
var tetra = require('../tetra.js'), // include tetra library
    contactLessService = tetra.service({ // Instantiate service
       service: 'local.device.contactless0',
       namespace: 'ingenico.device.contactless'
    });

function getCardInformations() {

   var aidCommand = ["00", "A4", "04", "00", "07", "A0", "00", "00", "00", "04", "10", "10", 
"00"];
   var PPSEreponse = [];

   contactLessService
       .reset() // Reset service
       .call('GetUid', {requestDelay: 0}) // Call GetUid method
       .success(function (r) {
           console.log(r);
       })
       .call('Apdu', {
           data: { // Select PPSE
              command: ["00", //CLA
                        "A4", //INS  select file
                        "04", "00",// P1 P2
                        "0E",
                        "32",  "50",  "41",  "59",  "2E",  "53",  "59",  "53",  "2E",  "44",  "44", 
"46", "30", "31", "00"]
                }
            })
            .success(function (r) {
                PPSEreponse = r.response;
            })
            .call('Apdu', {  // Select AID
                data: {
                    command: aidCommand
                }
            })
            .success(function (r) {


                if (
                    (PPSEreponse[27] == "A0") &&
                    (PPSEreponse[28] == "00") &&
                    (PPSEreponse[29] == "00") &&
                    (PPSEreponse[30] == "00") &&
                    (PPSEreponse[31] == "03")
                ) {
                    aidCommand[9] = "03";
                    aidCommand[10] = PPSEreponse[32];
                    aidCommand[11] = PPSEreponse[33];
                } else if (
                    (PPSEreponse[27] == "A0") &&
                    (PPSEreponse[28] == "00") &&
                    (PPSEreponse[29] == "00") &&
                    (PPSEreponse[30] == "00") &&
                    (PPSEreponse[31] == "04")
                ) { // MC

                    aidCommand[10] = PPSEreponse[32];
                    aidCommand[11] = PPSEreponse[33];
                } else {
                    return;
                }

                contactLessService
                    .reset() // Reset service
                    .call('Apdu', {
                        data: { // Send APDU to the card
                            command: aidCommand
                        }, requestDelay: 0
                    })
                    .success(function (r) {
                        console.log(r);
                    })
        });
}


contactLessService
    .reset() // Reset service
    .disconnect() // Disconnect from service
    .connect() // Connect to service
    .close() // Close service
    .open() // Open service
    .on('ClessDetectedEvent', function (r)  { // Listen to ClessDetectedEvent
         console.log('Card detected');

         return getCardInformations();
    })
    .call('StartDetection', {data: {timeout: 10000}}) // Call start detection method
    .then(function (r) {
       console.log('Please approach your card');
    }, function (e) {
        console.log(e)
    });
```

4_5 Transaction

4_5_1 Manager.manageTransaction(…)

Description
This method launches a transaction relying on the payment applications available in the terminal.

Service namespace	Interface	Method
`ingenico.transaction`	`Manager`	`manageTransaction`

List of possible service instances
`local.transaction.engine`

TetraJS `.call`	
.call parameters	

```
data:{
  transaction:
  {
    'currency':<a>,
    'value':<i>,
    'type':<j>
  },

  'mean'::<k>
}
```

<a> Currency

```
{
   'code':<b>,
   'numCode':<c>,
   'minorUnit':<d>,
   'minorUnitSeparator':<e>, // default = “,”
   'thousandSeparator':<f>,  // default = ““
   'position':<g>,
   'symbol':<h>
}
```

`<b>:`	Alphabetic code of the currency (ISO 4217)
`<c>:`	Numeric code of the currency (ISO 4217)
`<d>:`	Number of digits of the minor currency unit (ISO 4217)
`<e>:`	Character displayed between the major and minor units.
`<f>:`	Character displayed after the thousand and million digits.
`<g>:`	Position of the unit regarding the value. `[default = CURRENCY_AFTER_AMOUNT];`
`<h>:`	Symbol to be displayed instead of the alphabetic code (in UTF-8) `[default = ""];`

<i> Amount expressed in units of the currency

<j> Transaction type: by default “`Payment`”

	`<k>` Payment mean `UNKNOWN:` (0) Unspecified payment mean `MAG_STRIPE_1:` (1) Magnetic Stripe ISO1 `MAG_STRIPE_2:` (2) Magnetic Stripe ISO2 `MAG_STRIPE_3:` (3) Magnetic Stripe ISO3 `CHIP_CARD:` (4) SmartCard `MANUAL:` (5) Manual Entry `CONTACTLESS:` (6) Contactless `CHECK:` (7) Check

TetraJS `.success`	
HTTP status	`200 = OK`

Manager.manageTransaction() status	`data:{`
	`  'return':<a>,`
	`  'transactionDetails':<b>`
	`  }`
	
	with:
	`<a>` service call status
	`<b>` transaction report, enhances the returned status

TetraJS `.error`	
HTTP Status	Refer to Table 3.22

4_5_2 Transaction sample code

TetraJS based weblet sample

The code below launches a payment transaction of 2 €

```
tetra
.service({
    service:  'local.transaction.engine',
    namespace: 'ingenico.transaction'
})
.reset()
.connect()
.call('ManageTransaction',{
    hide:true,
    data: {
        transaction: {
            currency: {
                code: 'EUR',
                numCode: 978,
                minorUnit: 2,
                minorUnitSeparator: ",",
                thousandSeparator: "",
                position: "CURRENCY_BEFORE_AMOUNT",
                symbol: "&amp;euro;"
            },
            value: "200",
            transactionType: "Payment"
        },
    }
})
.success(function (e) {
    console.log('All is finished')
  })
.error(function (e) {
    console.log('ERROR: ' + e.response.transactionDetails);
  })
.disconnect()
```

APPENDIX A: Tetra Web Services REST API

A1] Terminal/Web Platform configuration, capabilities

Configure platform

Description	Configure platform
Scope	System
Method	POST
URI format	http://terminal.ingenico.com/setup

Request	JSON:
	{
	"*CONFIG_ID*": *VALUE*,
	…
	}
	CONFIG_ID – configuration ID
	VALUE – configuration value
	
	Example: {"gatewayTimeout": 600}

Response	
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors

Configure platform (alternative version)

Description	Configure platform
Scope	System
Method	POST
URI format	http://terminal.ingenico.com/setup?*CONFIG_ID*= *VALUE&..* Example: http://terminal.ingenico.com/setup?gatewayTimeout=600
Request	
Response	
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors

Get configuration

Description	Configure platform
Scope	Public
Method	GET
URI format	http://terminal.ingenico.com/setup
Request	

Response	JSON:
	{
	  "*CONFIG_ID*": *VALUE*,
	…
	}
	  CONFIG_ID – configuration ID
	  VALUE – configuration value
	
	Example: {"gatewayTimeout": 600}

HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors

A2] Interaction with Telium Tetra services

Lookup for a service interface

Description	Demand Service Directory to get a list of services proposing requested interface
Scope	Public
Method	GET
URI format	http://terminal.ingenico.com/service?interface=*INTERFACE_NAME* *INTERFACE_NAME* – name of interface (“service” in proto file)
Request	
Response	{ "services": [{ "id": *SERVICE_NAME*, "descr": *DESCRIPTION* }, ...] } *SERV_NAME* – name of services *DESCRIPTION* – service description
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 501 Not Implemented – specified interface method does not exist or is not supported by service 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

Lookup for a service event

Description	Demand Service Directory to get a list of services proposing requested event
Scope	Public
Method	GET
URI format	http://terminal.ingenico.com/service?event=*EVENT_NAME* *EVENT_NAME* – name of interface (“service” in proto file)
Request	

Response	{
	"services": [
	{ "id": *SERVICE_NAME*,
	"descr": *DESCRIPTION*
	},
	...
	]
	}
	
	SERV_NAME – name of services
	DESCRIPTION – service description

HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 501 Not Implemented – specified event does not exist or is not supported by service 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

Connect to a service

Description	Connect to a service of the platform.
Scope	Public
Method	GET
URI format	http://terminal.ingenico.com/service/*SERV_NAME?format_BYTES_FIELD_FULL_NAME*=*BYTES_FORMAT*... *SERV_NAME* – name of the platform service JSON representation of the binary protobuf data (type “bytes”) could be configured within the request as format sequence if not default. Request binary data could be represented in JSON with one of allowed form (*hex_string*, *hex_array*, *int_array*, *oct*, *hex*, *tlv* or *base64*; see description below). *BYTES_PARAM_FULL_NAME* – full name of the bytes-field (package_name.message_name.field_name) *BYTES_FORMAT* – JSON binary representation format. Possible values: • ***hex_string*** – hexadecimal string Example: "byte_field": "69AB00" • ***hex_array*** (default) – array of hex string values. Example: "byte_field": ["69", "AB", "00"] • ***int_array*** – array of integers. Example: "byte_field": [105, 171, 0] • ***tlv*** – TLV tree JSON object (an array of tag/length/value structures). Example: "byte_field": [{"tag" : "0x9F02", "length" : 6, "data" : [0,0,0,0,0,2]}, {"tag" : "0x5F2A", "length" : 2, "data" : [9,120] }] • ***base64*** – base64 string. Example: "aasA"
Request	
Response	
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

Make a synchronous call (RPC message names)

Description	Make a service remote procedure call

Scope	Public
Method	POST
URI format	http://terminal.ingenico.com/service/*SERV_NAME*?request=*REQUEST_MESSAGE_NAME*&response=*RESPONSE_MESSAGE_NAME* *SERV_NAME* – name of the platform service *REQUEST_MESSAGE_NAME* – request message full ID. *RESPONSE_MESSAGE_NAME* – response message full ID.

Request	{
	JSON_OBJECTS
	}
	
	JSON_OBJECTS – objects complying with service API
	
	Request binary data could be represented in JSON with one of allowed form (*hex_string*, *hex_array*, *int_array*, *oct*, *hex*, *tlv* or *base64*).

Response	{
	JSON_OBJECTS
	}
	
	JSON_OBJECTS – objects complying with service API
	
	JSON representation of the binary protobuf data (type “bytes”) could be configured within the **connect** request if not default.

HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 501 Not Implemented – specified interface method does not exist or is not supported by service 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

Disconnect from a service

Description	Disconnect from a service of the platform.
Scope	Public
Method	DELETE
URI format	http://terminal.ingenico.com/service/*SERV_NAME* *SERV_NAME* – name of the platform service
Request	
Response	
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

Service events

SSE technology is used to receive event messages from a service. SSE provides an efficient implementation of XHR (XmlHttpRequest) streaming; the actual delivery of the messages is done over a single, long-lived HTTP connection.

Description	Subscribe to service events from javasscript using: ***new EventSource(uri);***
Scope	Public
Method	GET
URI format	http://terminal.ingenico.com/service/*SERV_NAME*/sse *SERV_NAME* – name of the platform service
Request	
Response	Message stream: event: *EVENT_MESSAGE_NAME*\n data: { \n data: *JSON_OBJECTS* \n data: } \n\n *JSON_OBJECTS* – objects complying with service API
HTTP Statuses	200 OK 400 Bad Request – REST syntax error 405 Method Not Allowed – REST error (wrong method) 500 Internal Server Error – ESS exceptions or platform runtime errors 503 Service Unavailable – targeted service is not available in the machine 504 Gateway Timeout – service communication timeout (10s by default)

APPENDIX B: Telium Tetra status codes

Notice: the following symbols are not part of the tetra library. Error codes are given for reference to be used as constants in your code.

```
SUCCESS                  =  0,        ///< Success

/* Errors service */
ERR_NOT_FOUND            = -1000000,  ///< The element searched was not found
ERR_TIME_OUT             = -1000001,  ///< Time out detected
ERR_UNKNOWN_MESSAGE      = -1000003,  ///< The message or method is unknown
ERR_LINK_BUSY            = -1000004,  ///< The link is busy processing a call
ERR_NOT_RESERVED         = -1000005,  ///< The service is not reserved
ERR_RESERVED             = -1000006,  ///< The service is reserved
ERR_FAILED               = -1000100,  ///< Unrecoverable occured
ERR_ACCESS_DENIED        = -1000101,  ///< The access is denied
ERR_LINK_BROKEN          = -1000102,  ///< The link was brutally broken
ERR_LINK_FAILED          = -1000103,  ///< The message exchanges on the link failed
ERR_OPEN_FAILED          = -1000110,  ///< The file cannot be opened
ERR_PARSE_FAILED         = -1000111,  ///< The file cannot be parsed
ERR_PARSING_DESCRIPTOR   = -1000113,  ///< The name contains illegal characters or
                                    is not correct
ERR_BAD_NAME             = -1000114,  ///< The name contains illegal characters or
is not correct
ERR_BAD_ALIAS            = -1000115,  ///< The alias contains illegal characters or
is not correct
ERR_FULL_BUFFER          = -1000116,  ///< The reception buffer is full
ERR_REQUEST_TIME_OUT     = -1000117,  ///< The send resquest is timed out
ERR_RESPONSE_TIME_OUT    = -1000118,  ///< The response awaited is timed out
ERR_DECODING_FAILED      = -1000119,  ///< The reponse cannot be decoded
ERR_ENCODING_FAILED      = -1000120,  ///< The request cannot be encoded
ERR_READ_FAILED          = -1000121,  ///< Read failed
ERR_WRITE_FAILED         = -1000122,  ///< Write failed

/* Errors payment */
ERR_TOO_MANY_CARDS       = -2000001,  ///< Too many cards have been detected (Cards
errors)
ERR_CANCELLED            = -2000002,  ///< The processing has been cancelled (Cards
errors)
ERR_INVALID_CARD         = -2000003,  ///< The detected card is not valid (Cards
errors)
ERR_NOT_ISO              = -2000004,  ///< The non card ISO has been detected but
cannot be used with the contactless driver (Cards errors)
ERR_CARD_REMOVED         = -2000005,  ///< Card removed

/** Errors errno */
ERR_ERRNO_BASE           = -3000000,  ///< Errno base
ERR_EPERM                = -3000001,  ///< Operation not permitted
ERR_ENOENT               = -3000002,  ///< No such file or directory
ERR_ESRCH                = -3000003,  ///< No such process
ERR_EINTR                = -3000004,  ///< Interrupted system call
```

```
ERR_EIO                   = -3000005,  ///< I/O error
ERR_ENXIO                 = -3000006,  ///< No such device or address
ERR_E2BIG                 = -3000007,  ///< Argument list too long
ERR_ENOEXEC               = -3000008,  ///< Exec format error
ERR_EBADF                 = -3000009,  ///< Bad file number
ERR_ECHILD                = -3000010,  ///< No child processes
ERR_EAGAIN                = -3000011,  ///< Try again
ERR_ENOMEM                = -3000012,  ///< Out of memory
ERR_EACCES                = -3000013,  ///< Permission denied
ERR_EFAULT                = -3000014,  ///< Bad address
ERR_ENOTBLK               = -3000015,  ///< Block device required
ERR_EBUSY                 = -3000016,  ///< Device or resource busy
ERR_EEXIST                = -3000017,  ///< File exists
ERR_EXDEV                 = -3000018,  ///< Cross-device link
ERR_ENODEV                = -3000019,  ///< No such device
ERR_ENOTDIR               = -3000020,  ///< Not a directory
ERR_EISDIR                = -3000021,  ///< Is a directory
ERR_EINVAL                = -3000022,  ///< Invalid argument
ERR_ENFILE                = -3000023,  ///< File table overflow
ERR_EMFILE                = -3000024,  ///< Too many open files
ERR_ENOTTY                = -3000025,  ///< Not a typewriter
ERR_ETXTBSY               = -3000026,  ///< Text file busy
ERR_EFBIG                 = -3000027,  ///< File too large
ERR_ENOSPC                = -3000028,  ///< No space left on device
ERR_ESPIPE                = -3000029,  ///< Illegal seek
ERR_EROFS                 = -3000030,  ///< Read-only file system
ERR_EMLINK                = -3000031,  ///< Too many links
ERR_EPIPE                 = -3000032,  ///< Broken pipe
ERR_EDOM                  = -3000033,  ///< Math argument out of domain of func
ERR_ERANGE                = -3000034,  ///< Math result not representable
ERR_EDEADLK               = -3000035,  ///< Resource deadlock would occur
ERR_ENAMETOOLONG          = -3000036,  ///< File name too long
ERR_ENOLCK                = -3000037,  ///< No record locks available
ERR_ENOSYS                = -3000038,  ///< Function not implemented
ERR_ENOTEMPTY             = -3000039,  ///< Directory not empty
ERR_ELOOP                 = -3000040,  ///< Too many symbolic links encountered
ERR_ENOMSG                = -3000042,  ///< No message of desired type
ERR_EIDRM                 = -3000043,  ///< Identifier removed
ERR_ECHRNG                = -3000044,  ///< Channel number out of range
ERR_EL2NSYNC              = -3000045,  ///< Level 2 not synchronized
ERR_EL3HLT                = -3000046,  ///< Level 3 halted
ERR_EL3RST                = -3000047,  ///< Level 3 reset
ERR_ELNRNG                = -3000048,  ///< Link number out of range
ERR_EUNATCH               = -3000049,  ///< Protocol driver not attached
ERR_ENOCSI                = -3000050,  ///< No CSI structure available
ERR_EL2HLT                = -3000051,  ///< Level 2 halted
ERR_EBADE                 = -3000052,  ///< Invalid exchange
ERR_EBADR                 = -3000053,  ///< Invalid request descriptor
ERR_EXFULL                = -3000054,  ///< Exchange full
ERR_ENOANO                = -3000055,  ///< No anode
ERR_EBADRQC               = -3000056,  ///< Invalid request code
ERR_EBADSLT               = -3000057,  ///< Invalid slot
ERR_EBFONT                = -3000059,  ///< Bad font file format
```

```
ERR_ENOSTR               = -3000060,  ///< Device not a stream
ERR_ENODATA              = -3000061,  ///< No data available
ERR_ETIME                = -3000062,  ///< Timer expired
ERR_ENOSR                = -3000063,  ///< Out of streams resources
ERR_ENONET               = -3000064,  ///< Machine is not on the network
ERR_ENOPKG               = -3000065,  ///< Package not installed
ERR_EREMOTE              = -3000066,  ///< Object is remote
ERR_ENOLINK              = -3000067,  ///< Link has been severed
ERR_EADV                 = -3000068,  ///< Advertise error
ERR_ESRMNT               = -3000069,  ///< Srmount error
ERR_ECOMM                = -3000070,  ///< Communication error on send
ERR_EPROTO               = -3000071,  ///< Protocol error
ERR_EMULTIHOP            = -3000072,  ///< Multihop attempted
ERR_EDOTDOT              = -3000073,  ///< RFS specific error
ERR_EBADMSG              = -3000074,  ///< Not a data message
ERR_EOVERFLOW            = -3000075,  ///< Value too large for defined data type
ERR_ENOTUNIQ             = -3000076,  ///< Name not unique on network
ERR_EBADFD               = -3000077,  ///< File descriptor in bad state
ERR_EREMCHG              = -3000078,  ///< Remote address changed
ERR_ELIBACC              = -3000079,  ///< Can not access a needed shared library
ERR_ELIBBAD              = -3000080,  ///< Accessing a corrupted shared library
ERR_ELIBSCN              = -3000081,  ///< .lib section in a.out corrupted
ERR_ELIBMAX              = -3000082,  ///< Attempting to link in too many shared
libraries
ERR_ELIBEXEC             = -3000083,  ///< Cannot exec a shared library directly
ERR_EILSEQ               = -3000084,  ///< Illegal byte sequence
ERR_ERESTART             = -3000085,  ///< Interrupted system call should be
restarted
ERR_ESTRPIPE             = -3000086,  ///< Streams pipe error
ERR_EUSERS               = -3000087,  ///< Too many users
ERR_ENOTSOCK             = -3000088,  ///< Socket operation on non-socket
ERR_EDESTADDRREQ         = -3000089,  ///< Destination address required
ERR_EMSGSIZE             = -3000090,  ///< Message too long
ERR_EPROTOTYPE           = -3000091,  ///< Protocol wrong type for socket
ERR_ENOPROTOOPT          = -3000092,  ///< Protocol not available
ERR_EPROTONOSUPPORT      = -3000093,  ///< Protocol not supported
ERR_ESOCKTNOSUPPORT      = -3000094,  ///< Socket type not supported
ERR_EOPNOTSUPP           = -3000095,  ///< Operation not supported on transport
endpoint
ERR_EPFNOSUPPORT         = -3000096,  ///< Protocol family not supported
ERR_EAFNOSUPPORT         = -3000097,  ///< Address family not supported by protocol
ERR_EADDRINUSE           = -3000098,  ///< Address already in use
ERR_EADDRNOTAVAIL        = -3000099,  ///< Cannot assign requested address
ERR_ENETDOWN             = -3000100,  ///< Network is down
ERR_ENETUNREACH          = -3000101,  ///< Network is unreachable
ERR_ENETRESET            = -3000102,  ///< Network dropped connection because of
reset
ERR_ECONNABORTED         = -3000103,  ///< Software caused connection abort
ERR_ECONNRESET           = -3000104,  ///< Connection reset by peer
ERR_ENOBUFS              = -3000105,  ///< No buffer space available
ERR_EISCONN              = -3000106,  ///< Transport endpoint is already connected
ERR_ENOTCONN             = -3000107,  ///< Transport endpoint is not connected
```

```
ERR_ESHUTDOWN            =  -3000108,   ///<  Cannot  send  after  transport  endpoint
shutdown
ERR_ETOOMANYREFS         = -3000109,  ///< Too many references: cannot splice
ERR_ETIMEDOUT            = -3000110,  ///< Connection timed out
ERR_ECONNREFUSED         = -3000111,  ///< Connection refused
ERR_EHOSTDOWN            = -3000112,  ///< Host is down
ERR_EHOSTUNREACH         = -3000113,  ///< No route to host
ERR_EALREADY             = -3000114,  ///< Operation already in progress
ERR_EINPROGRESS          = -3000115,  ///< Operation now in progress
ERR_ESTALE               = -3000116,  ///< Stale NFS file handle
ERR_EUCLEAN              = -3000117,  ///< Structure needs cleaning
ERR_ENOTNAM              = -3000118,  ///< Not a XENIX named type file
ERR_ENAVAIL              = -3000119,  ///< No XENIX semaphores available
ERR_EISNAM               = -3000120,  ///< Is a named type file
ERR_EREMOTEIO            = -3000121,  ///< Remote I/O error
ERR_EDQUOT               = -3000122,  ///< Quota exceeded
ERR_ENOMEDIUM            = -3000123,  ///< No medium found
ERR_EMEDIUMTYPE          = -3000124,  ///< Wrong medium type
ERR_ECANCELED            = -3000125,  ///< Operation Canceled
ERR_ENOKEY               = -3000126,  ///< Required key not available
ERR_EKEYEXPIRED          = -3000127,  ///< Key has expired
ERR_EKEYREVOKED          = -3000128,  ///< Key has been revoked
ERR_EKEYREJECTED         = -3000129,  ///< Key was rejected by service
ERR_EOWNERDEAD           = -3000130,  ///< Owner died
```