Integración SuperCapa Versiones 5.x

Versión SC 5.216. Manual Práctico 24/10/14

Tabla de contenido

1 Introducción

Este manual pretende ser una guía práctica para la integración de SuperCapa en cualquier página HTML donde se requiera un número 900 y C2C.

El objetivo principal de SC.js o SuperCapa, es el de gestionar los números 900 y C2C (Click to call) de una forma centralizada, en base a campañas.

La API JavaScript de SuperCapa provee los métodos necesarios para establecer qué número 900 y C2C usar allí donde se incluya la misma.

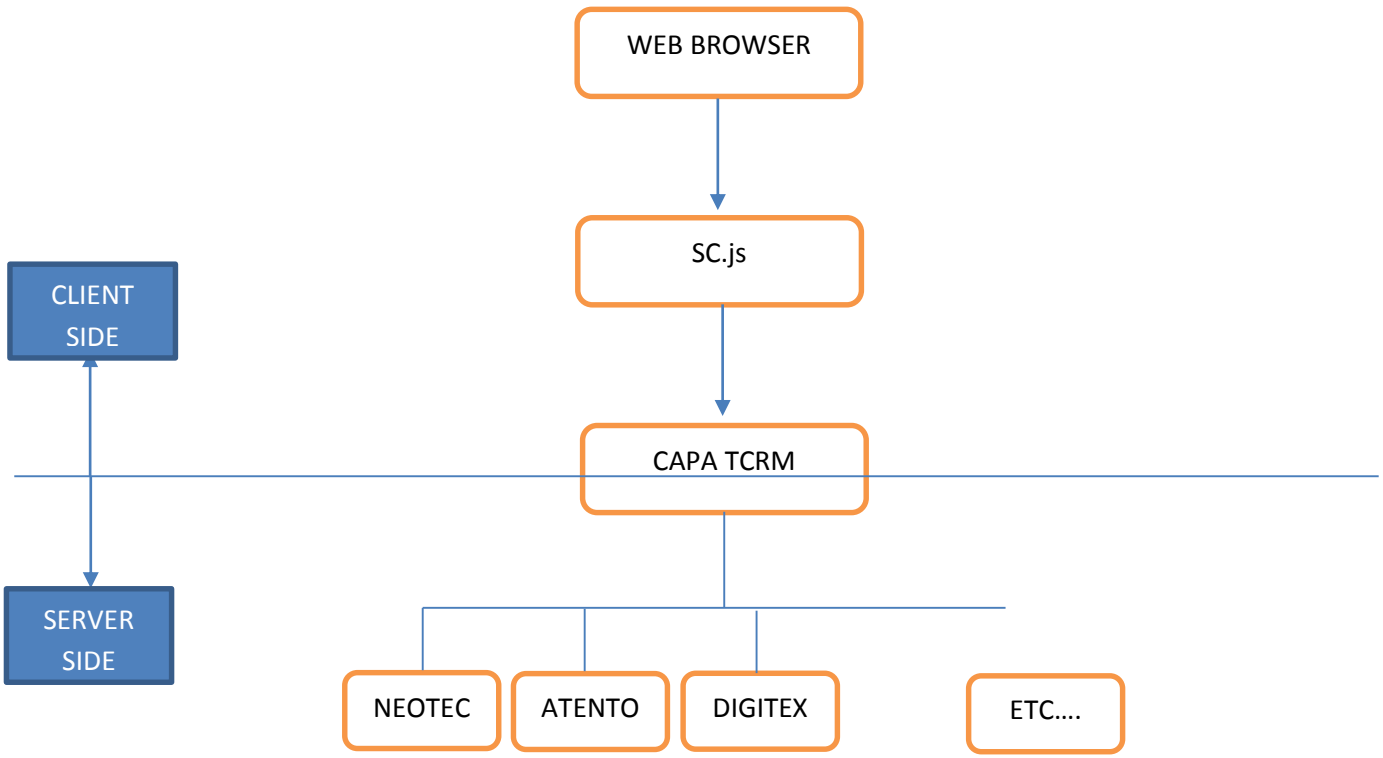
2 Proceso de integración rápida

El proceso de integración de la SuperCapa podría ser el siguiente:

1. Incluir los ficheros JS definidos en el punto 4.1 (Ficheros JavaScript)
2. Añadir los parámetros de prueba a la url para poder hacer pruebas con los distintos C2C. Punto 4.6 (Parámetros de prueba, validaciones y depuración)
3. Asociar el método lanzar a algún botón. Punto 5.1 (Funcionalidades)
4. En consola (depurador del navegador) habilitar el modo de depuración "SC.startDebug();"
    a. Si todo ha ido correcto, debería abrirse el C2C y mostrar por consola los pasos que se van siguiendo.

* Recordar que no todos los C2C aceptan todas las funcionalidades, punto 5.3 (Tabla Resumen)

3 Esquema de funcionamiento



4 Funcionamiento normal

4.1 Ficheros JavaScript (descargar html anexo)

Los ficheros JavaScript necesarios para el correcto funcionamiento de SuperCapa son los siguientes:

```
1. <script type="text/javascript">var s = s || {};var s_account = "";function
   s_gi(s_account) { return s || {}; }function tl(a,b,c) { return s || {}; }</script>

2. <script type="text/javascript" src="//www.movistar.es/estaticos/landings/js/jquery-
   1.10.2.js"></script>

   <!-- SC JS -->
3. <script type="text/javascript" src="//www.movistar.es/atcliente/c2c/venta-asistida-
   externo/polaris.venta.asistida.externo/polaris.venta.asistida.externo.nocache.js">
   </script>

4. <script type="text/javascript" src="//www.movistar.es/estaticos/js/SC.js"></script>

   <body>
5. <script>(function(a,b,c,d){
   a='//tags.tiqcdn.com/utag/telefonica/col10/prod/utag.js';
   b=document;c='script';d=b.createElement(c);d.src=a;d.type='text/java'+c;d.async=true;
   a=b.getElementsByTagName(c)[0];a.parentNode.insertBefore(d,a);})();</script>
```

1. var s & function tl
2. Versión 1.10.2 de jQuery o superior
3. JavaScript “Venta-Asistida-Externo”
4. JavaScript propio de SuperCapa
5. Añadir la huella de tealium lo más cerca de la etiqueta de apertura del body

4.2 Campañas

En resumen la SC puede definirse como un “gestor de campañas” en el sentido que asocia una plataforma C2C y un 900 a determinada campaña que se definen de forma unívoca con una dupla (parámetro = valor).

El término campaña es muy importante ya que es la base de la asignación de plataformas y números 900.

El funcionamiento consiste en crear un “repositorio” de campañas activas que se descargarán a través del JSON de SC y que se asignará en el momento de carga de la página.

Además es importante entender la prioridad que establece la SC a la hora de asignar la campaña.

1. **Parámetros en la URL**
   Lo primero que la SC intentará hacer es buscar una campaña (param=valor) en parámetros de la URL.
2. **SC.establecerCampaniaAlternativa(param, valor)**
   Si en la url no existen parámetros que identifiquen una campaña, entonces cargará la campaña que se le indique mediante este método.
   Es importante entender que el primero siempre tendrá prioridad, es decir, si por parámetros se indica una campaña, ignorará lo que se defina en este método.

Estas campañas se proporcionarán desde Telefónica.

Se recomienda definir siempre esté método para asegurar que aunque no existan parámetros en la url, al menos una campaña se cargue correctamente.

4.3 Ejecución C2C

La ejecución de los C2C se realizará a través del método lanzar del objeto SC.

```
/**
* Lanza el C2C
*
* @param tipología<String> OBLIGATORIO Tipo de C2C a lanzar [cml | cml | completo ]
* @param campos<Object> Map conteniendo pares clave valor de los valores a enviar al C2C
* @param invoAdicional<Object> Map conteniendo pares clave valor de información adicional
*
* @returns true si la ejecución fue correcta, false en otro caso.
*/
SC.lanzar(tipologia, campos, infoAdicional);
```

La ejecución del método lanzar, lleva implícita la ejecución de los siguientes métodos en el orden indicado:

1. SC.preAbrir()
2. – Ejecución del C2C –
3. SC.postAbrir()
4. SC.callback()

4.4 Tras la inclusión de SC.js

Tras el momento de la inclusión del fichero JavaScript “www.movistar.es/estaticos/js/SC.js” tienen lugar los siguientes eventos de forma automática:

1. **Creación del objeto SC**. Tras incluir SC.js se crea de forma automática el objeto SC. Este objeto encapsula todos los métodos de la API de SuperCapa. Se trata de una variable global accesible desde cualquier ámbito.
2. **Carga JSON**. Tras la creación se realiza la carga del fichero JSON correspondiente conteniendo la información necesaria para SuperCapa.
3. **SC.onReady().** Tras los pasos 1 y 2 anteriores, se ejecutará la función onReady, pensada para sincronizar la finalización de carga de SuperCapa con los métodos definidos en la propia API.

4.5 HTML de ejemplo

En la siguiente url: http://www.movistar.es/promociones/supercapa se puede ver una implementación de prueba de SuperCapa, no obstante, a continuación se detallan los principales elementos necesarios:

```
<head>

<!-- SCODE-->
<script type="text/javascript"
src="//www.movistar.es/estaticos/js/tealium/s_code.js"></script>
```

```
<script type="text/javascript">
	window.s.products = "Valor proporcionado";
	window.s.pageName = "";
</script>

</head>


<body>

<!-- Tealium-->
<script>(function(a,b,c,d){
a='//tags.tiqcdn.com/utag/telefonica/col10/prod/utag.js';
b=document;c='script';d=b.createElement(c);d.src=a;d.type='text/java'+c;d.async=true;
a=b.getElementsByTagName(c)[0];a.parentNode.insertBefore(d,a);})();</script>
<!-- ./Tealium -->



<a href="javascript:;"
   onclick="SC.lanzar('cml',{nombre:v1, apellido:v2, telefono:v3, fecha:v4, hora:v5});">
   Boton C2C CML</a>

<a href="javascript:;" onclick="SC.lanzar('cmn',{nombre:v1, telefono:v2});">Boton C2C CMN</a>

<a href="javascript:;" onclick="SC.lanzar('completo');">Boton Completo</a>



<!-- jQuery -->
<script type="text/javascript" src="//www.movistar.es/estaticos/landings/js/jquery-
1.10.2.js"></script>

<!-- SC JS -->
<script type="text/javascript" src="//www.movistar.es/atcliente/c2c/venta-asistida-
externo/polaris.venta.asistida.externo/polaris.venta.asistida.externo.nocache.js"></script>
<script type="text/javascript" src="//www.movistar.es/estaticos/js/SC.js"></script>



<!-- Personalizacion SuperCapa -->
<script type="text/javascript">

	SC.onReady (function(){

		//Si no hay Campania por URL
		SC.establecerCampaniaAlternativa('sctest','test1');

		//Se indica que no se desea mostrar modal
		SC.setUsoModal(false);

		//Función a ejecutar antes de abrir el C2C
		SC.setPreAbrir (function(){

			/** El código aquí **/
			return true;
		});

		//Función a ejecutar despues de abrir el C2C
```

```
            SC.setPostAbrir (function(){

                  /** El código aquí **/

            });

            //Función a ejecutar tras la ejecución de los distintos estados del C2C
            SC.callback (function(idlead,idEstado,info){

                  if (info){
                        /** El código aquí **/
                  }

            });

            //Compruebo la validez de la configuracion
            SC.validarConfiguracion(true);

      });

</script><!-- ./Personalizacion SuperCapa -->

</body>
```

4.6 Parámetros de prueba, validaciones y depuración

Con el objetivo de agilizar el proceso de integración de SuperCapa, se han definido unas 'campañas' de pega que permiten probar los distintos C2C definidos en SuperCapa. Estos parámetros son:

1. sctest=test2
2. sctest=test3

Al usarlos en la url donde se encuentre integrada supercapa, esta configurará dichos C2C para poder hacer pruebas con las distintas variantes de C2C.

Por otro lado, se pueden habilitar los mensajes de SuperCapa a través de alguna consola de depuración o desarrollo HTML (Herramientas como firebug o los propios depuradores de google chrome o Firefox). Estos mensajes dan información de errores de ejecución o funcionamiento que pueden ayudar a la integración.

```
/**
 * Inicializa la depuración
 */
SC.startDebug();


/**
 * Detiene la depuración
 */
SC.stopDebug();
```

Por otro lado, se ha habilitado un método que permite comprobar si los elementos más básicos de SuperCapa se encuentran definidos:

```
/**
 * Valida si los elementos obligatorios para SuperCapa se encuentran
 * correctamente editados
 *
 * @param mostrar:boolean, si true, muestra por consola los errores
 *
 * @returns true si la configuracion es correcta, false en otro caso
 */
SC.validarConfiguracion(mostrar)
```

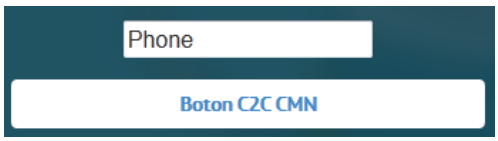
5 C2C (Clic to call)

Un C2C (click to call) es una petición a una plataforma de atención a través de SuperCapa. Dicha petición debe contener los campos necesarios para poder alcanzar en última instancia al usuario final que será contactado.

5.1 Funcionalidades

5.1.1 CMN (Call me now)

La funcionalidad 'cmn' de un C2C permite especificar a SuperCapa que se desea recibir contestación por parte de la plataforma de **forma inmediata**.



Campos obligatorios:

telefono : Teléfono en el cual desea ser contactado el usuario final.

Campos opcionales:

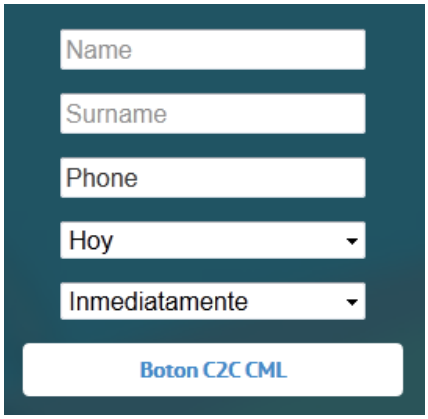
nombre: Nombre del usuario final.

Implementación:

```
1. var campos = {};
2. campos.telefono = "999666999";
3. campos.nombre = "Juanito";
4. SC.lanzar('cmn', campos);
```

5.1.2 CML (Call me later)

La funcionalidad 'cml' de un C2C permite especificar a SuperCapa que se desea recibir contestación por parte de la plataforma no de forma inmediata, sino **especificando cuando** se desea ser atendido.



Campos obligatorios:

telefono: Teléfono en el cual desea ser contactado el usuario final.

Campos opcionales:

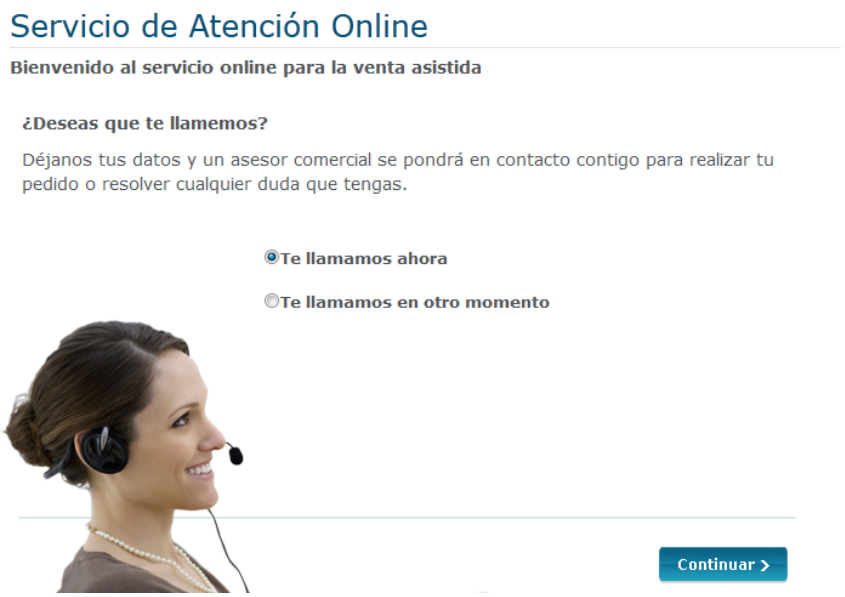
nombre: Nombre del usuario final
apellido: Apellido del usuario final
fecha: Fecha en la que desea ser atendido: [‘Hoy’ | ‘Mañana’ | ‘Pasado Mañana’] (Valor por defecto ‘Hoy’).
hora: Hora en la que desea ser atendido ['Inmediatamente' | '09:00 a 12:00' | '12:00 a 15:00' | '15:00 a 18:00' | '18:00 a 21:00'] (Valor por defecto ‘Inmediatamente’)

Implementacion:

```
1. var campos = {};
2. campos.nombre = "Alberto";
3. campos.apellido = "Cañabate";
4. campos.telefono = "666666666";
5. campos.fecha = "Mañana";
6. campos.hora = "09:00 a 12:00";
7. SC.lanzar('cml', campos);
```

5.1.3 Completo

La funcionalidad ‘completo’ de un C2C no requiere de ninguna información. La información se irá solicitando en las distintas ventanas que se mostrarán tras su ejecución:



Campos obligatorios

Ninguno

Campos opcionales

Ninguno

Implementación

```
1. SC.lanzar('completo');
```

5.2 C2C Definidos

5.2.1 Capa TCRM

Funcionalidades aceptadas: completo
Estados: Consultar punto 9.1 (Flujo Capa)

5.2.2 CapaExpres TCRM

Funcionalidades aceptadas: cmn, cml
Estados: Consultar punto 9.2 (Flujos Capa Expres)

5.3 Tabla Resumen

En la siguiente tabla se muestran los distintos C2C definidos y las funcionalidades aceptadas por los mismos.

Tabla de compatibilidades	CMN	CML	Completo
Capa TCRM	No	No	Si
CapaExpres TCRM	Si	Si	No

5.4 Inactividad de C2C

El C2C seleccionado por SuperCapa puede tener dos estados principalmente, activo e inactivo. Cuando un C2C se encuentra inactivo, este no podrá atender al usuario final.

5.4.1 Nivel JavaScript

Para determinar si un C2C es activo a nivel JavaScript se puede usar el siguiente método

```
/**
 * Determina si el C2C se encuentra activo
 *
 * @returns true si es activo, false en otro caso
 */
SC.isC2CActivo();
```

5.4.2 Nivel HTML

Para determinar si el C2C se encuentra activo a nivel HTML se pueden usar las siguientes clases

```
<div class="sc-visible-c2cActivo">
      Este elemento será visible únicamente cuando el C2C esté activo
</div>

<div class="sc-oculto-c2cActivo">
      Este elemento estará oculto únicamente cuando el C2C esté activo
</div>
```

5.5 Información de C2C

- Integración por CAPA TCRM: esto incluye a capa TCRM y capaExpres TCRM ya que esta segunda sólo se diferencia de la primera en la navegación. Realmente su funcionalidad es la misma. Y podría definirse como un “repartidor” de plataformas que se llaman bajo la misma implementación. Esta CAPA es un desarrollo propio de Telefónica y permite entre otras muchas cosas realizar desbordamientos en base a situaciones de disponibilidad reales de los call center. La información que se obtiene en cada estado es:
    - infoEstado (**propio de SC**)
    - idLead (**propio de SC**)
    - alias: identificador de Capa (**propio de CAPA**)
    - phase: identifica el estado (**propio de CAPA**) (Coincidirá con infoEstado)
    - reason: motivo por el cual se llegó a dicho estado (**propio de CAPA**)
    - sessionid: formado por sessionid de CAPA e idLead de SC (**propio de CAPA**)
    - idu: identificador asociado a la llamada. Solo tendrá valor en fases posteriores a la realización de la llamada (**propio de CAPA**)

- workplace: contiene el nombre del centro. Solo tendrá valor en fases posteriores a la realización de la llamada (**propio de CAPA**)

Esta información se recogerá en la función definida a realizar en cada cambio de estado, por ejemplo:

```
SC.callback (function(idlead,infoEstado,info){
	var idLead = idlead;
	var estado = infoEstado;
	if (info){
		var alias = info.alias;
		var phase = info.phase;
		var reason = info.reason;
		var sessionid = info.sessionid;
		var idu = info.idu;
		var provider = info.provider;
		var workplace = info.workplace;

	}

});
```

Hay que recordar que ciertos valores como el “idu”, “provider” o “workplace” no están disponible hasta que la llamada es efectiva (véase diagrama de estados de Capa o CapaExpress, esto corresponde al estado “CLIENT ANSWER”).

6 Número de teléfono 900

SuperCapa gestionará un número de teléfono gratuito al que podrá llamar el usuario final. Se puede obtener dicho número de dos formas, html y JavaScript

6.1 Nivel HTML

Tras la carga de SuperCapa se rellenará el contenido de los elementos html con clase ‘sc-telefono’ añadiendo a los mismos el teléfono gratuito de la forma que se muestra a continuación

```
<div>
	<div class="sc-telefono">900 900 900</div>
	<a class="sc-telefono" href="tel:900900900">900 900 900</a>
</div>
```

6.2 Nivel JavaScript

Se puede obtener el número 900 usando el siguiente método:

```
/**
 * Devuele el telefono gratuito
 *
 * @returns String con el teléfono o null si no lo hay o es inactivo
 */
SC.getTelefono();
```

6.3 Inactividad del teléfono

Los teléfonos gratuitos pueden encontrarse inactivos. La inactividad de los mismos puede determinarse a nivel JavaScript o HTML:

```
<div class="sc-visible-900Activo">
      Este elemento será visible únicamente cuando el 900 esté activo.
</div>

<div class="sc-visible-900Activo">
      Este elemento estará oculto únicamente cuando el 900 esté activo
</div>
```

```
/**
 * Determina si el teléfono es activo
 *
 * @returns true si es activo, false en otro caso
 */
SC.isTelefonoActivo();
```

7 Huella de Tealium y variables de tagueado

La huella de tealium permite a la SuperCapa taguear los eventos que suceden así como enviar la información necesaria a Google Analytics.

La huella de talium consta de los siguientes elementos necesarios:

7.1 s_code.js

Por un lado necesita de la inclusión del siguiente fichero JavaScript:

```
<script type="text/javascript" src="//www.movistar.es/estaticos/js/tealium/landings-externas/s_code.js"></script>
```

7.2 utag.js

Se trata de un trozo de código JavaScript que se debe añadir después de la inclusión de s_code.js y al principio de la página, justo tras la apertura de la etiqueta <body>.

```
<!-- TEALIUM -->
<script type="text/javascript">
(function(a,b,c,d){
a='//tags.tiqcdn.com/utag/telefonica/col10/prod/utag.js';
b=document;c='script';d=b.createElement(c);d.src=a;d.type='text/java'+c;d.async=true;
a=b.getElementsByTagName(c)[0];a.parentNode.insertBefore(d,a);
})();
</script>
<!-- FIN TEALIUM -->
```

Para evitar errores, **se debe evitar la doble inclusión de este código en cualquier página**

7.3 Definir e inicializar la variable s.products

Es necesario definir la variable s.products y establecer su valor de forma correcta. La forma correcta de hacer esto es la siguiente:

```
window.s["products"] = "oferat....";
```

El valor de s.products será proporcionado por parte de Telefonica.

Esta línea de código debe ejecutarse lo antes posible dentro de la página y **tras la inserción de s_code.js**

7.4 Definir e inicializar la variable s.pageName

Es necesario definir la variable s.pageName y establecer su valor de forma correcta. La forma correcta de definir la variable s.pageName es la siguiente:

```
window.s["pageName"] = "…";
```

El valor de la variable s.pageName sigue el siguiente patrón:

<microsites>:<nombre-proveedor>:<opcional>:<valor s.products>

Ejemplo de implementación:

```
window.s["pageName"] = "microsites:digital-media:mobile:" + window.s.products;
```

Esta línea de código debe ejecutarse lo antes posible dentro de la página y **tras la inserción de s_code.js**

8 Métodos SuperCapa

8.1 onReady(func)

```
/**
 * Establece la funcionalidad a ejecutar tras la carga completa de SuperCapa
 */
SC.onReady(func);
```

8.2 lanzar(tipologia, campos, infoAdicional);

```
/**
 * Lanza el C2C
 *
 * @param tipologia:String Tipo de C2C a lanzar [cml | cml | ... ]
 * @param campos:Map, Map conteniendo pares (clave valor) de los valores a enviar al C2C
 * @param invoAdicional:Map Map conteniendo pares (clave valor) de información
 * adicional a enviar al C2C
 *
 * @returns true si la ejecución fue correcta, false en otro caso.
 */
SC.lanzar(tipologia, campos, infoAdicional);
```

8.3 callback(*func(idLead, infoEstado, info){}*)

```
/**
 * Establece la función que quedará a la escucha de los distintos
 * eventos que ocurran en SuperCapa. Cuando ocurran se ejecutará
 *
 * @param func:Function
 */
SC.callback(func);
```

8.4 getIdC2C()

```
/**
 * Devuelve el identificador de C2C
 *
 * @returns string el id del C2C si existe, null en otro caso
 */
SC.getIdC2C();
```

8.5 getTelefono()

```
/**
 * Devuele el telefono gratuito
 *
```

```
 *  @returns String con el telefono o null si no lo hay
 */
SC.getTelefono()
```

8.6 getLastIdLead()

```
/**
 * Devuelve el último idLead generado
 *
 * @returns {string} el idLead generado o null si no hay ninguno
 */
SC.getLastIdLead();
```

8.7 getIdCampania()

```
/**
 * Devuelve el identificador de la campania
 *
 * @returns string el id de la campania si existe, null en otro caso
 */
SC.getIdCampania()
```

8.8 getUrlParamName()

```
/**
 * Devuelve el parametro de la url que determina la campania
 *
 * @returns string con el nombre del parametro o null si error
 */
SC.getUrlParamName();
```

8.9 getFuncionalidadC2C()

```
/**
 * Devuelve un array con las funcionalidades aceptadas por el C2C seleccionado
 *
 * @returns Array<String> conteniendo las funcionalidades que acepta el c2c
 */
SC.getFuncionalidadC2C();
```

8.10 getVersion()

```
/**
 * Devuelve la versión de SuperCapa
 */
```

```
SC.getVersion();
```

8.11 establecerCampaniaAlternativa(parametro, valor)

```
/**
 * Establece la campaña a usar en base a su parámetro y a un id,
 * siempre que no haya previamente definida una campaña
 *
 * @param parametro:String Parametro de la campania
 * @param valor:String Valor del parametro de la campania
 *
 * @returns true si correcto, false en otro caso
 */
SC.establecerCampaniaAlternativa(parametro, valor);
```

8.12 establecerConfiguracionAlternativa(idC2C, telefono, parametroC2C)

```
/**
 * Establece una configuración alternativa en caso de que no haya campaña
 * predefinida ni campaña alternativa
 *
 * @param idC2C:String identificador de C2C
 * @param telefono:String número teléfono a mostrar
 * @param parametroC2C:String parametro a pasar al C2C si es necesario
 *
 * @returns true si correcto, false en otro caso
 */
SC.establecerConfiguracionAlternativa(idC2C, telefono, parametroC2C);
```

8.13 setPostAbrir(func)

```
/**
 * Establece funcionalidad adicional al C2C despues de su ejecución
 *
 * @param func:function Función a ejecutar despues de lanzar el C2C
 */
SC.setPostAbrir(func);
```

8.14 setUsoModal(usaModal)

```
/**
 * Determina si usar los modales própios de SuperCapa o no.
 *
 * @param usaModal:boolean si true, se suarán los modales, si false no.
 */
SC.setUsoModal(usaModal);
```

8.15 isC2CActivo()

```
/**
 * Determina si el C2C se encuentra activo
 *
 * @returns true si es activo, false en otro caso
 */
SC.isC2CActivo();
```

8.16 isTelefonoActivo()

```
/**
 * Determina si el telefono es activo
 *
 * @returns true si es activo, false en otro caso
 */
SC.isTelefonoActivo();
```

8.17 isCargaFinalizada()

```
/**
 * Determina si ya se cargó la SuperCapa
 */
SC.isCargaFinalizada();
```

8.18 isModalActivo()

```
/**
 * True si se van a usar los modales propios, false en otro caso
 */
SC.isModalActivo();
```

8.19 validarConfiguracion(mostrar)

```
/**
 * Valida si los elementos obligatorios para SuperCapa se encuentran
 * correctamente editados
 *
 * @param mostrar:boolean, si true, muestra por consola los errores
 *
 * @returns true si la configuracion es correcta, false en otro caso
 */
SC.validarConfiguracion(mostrar)
```

8.20 stopDebug()

```
/**
 * Detiene la depuración
 */
SC.stopDebug();
```

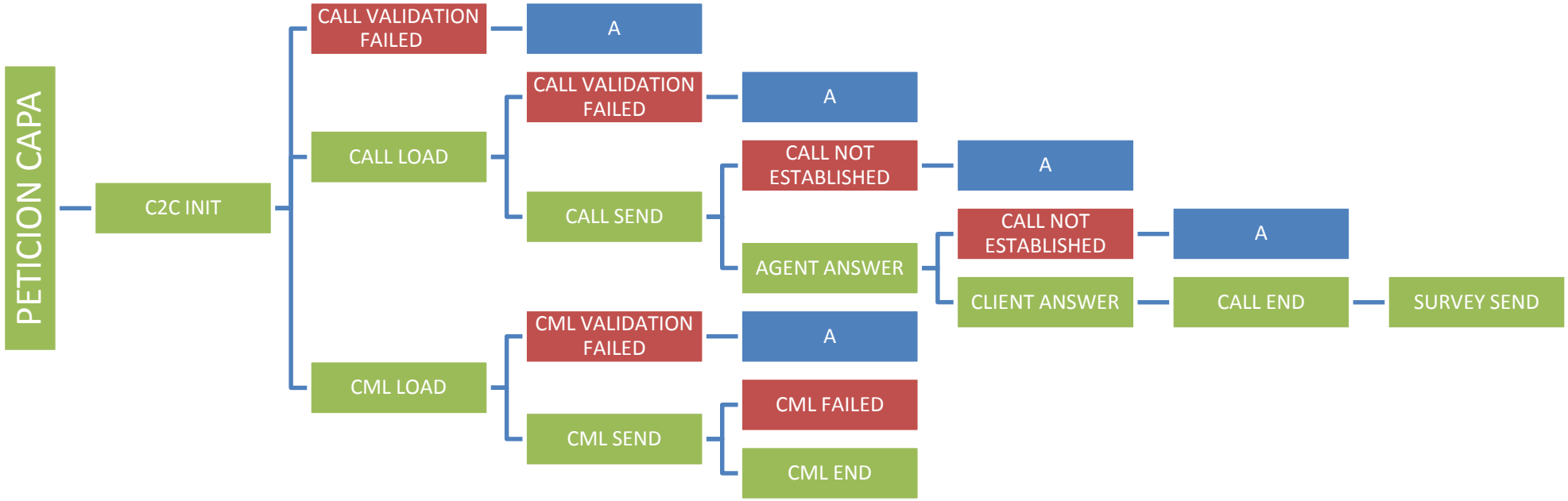
8.21 startDebug()

```
/**
 * Inicializa la depuración
 */
SC.startDebug();
```

9 Anexos

9.1 Flujo Capa

9.1.1 Flujo Normal



A = CML LOAD

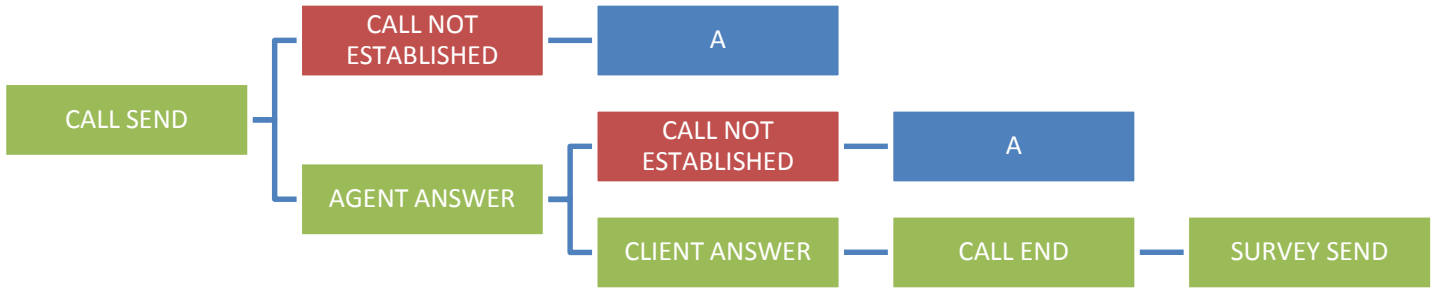
El estado A es realmente el estado CML LOAD. Cuando ocurre un error 'controlado' el flujo pasa al estado CML LOAD.

9.1.2 Estados Genéricos

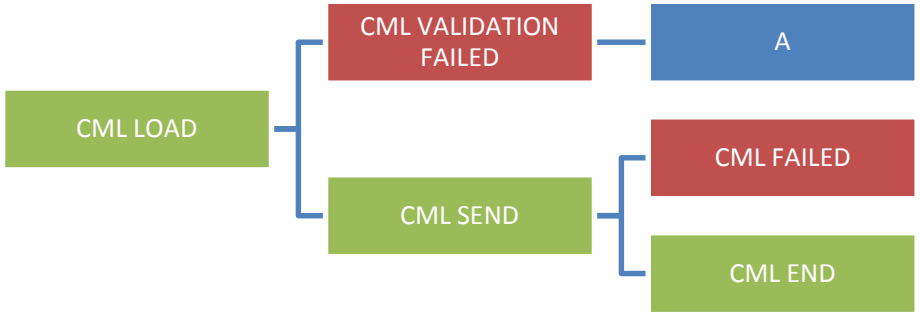
Los estados que se pueden despertar en cualquier momento del flujo normal son: 'GENERIC FAILURE' y 'C2C FEND'

9.2 Flujos Capa Expres

9.2.1 CMN



9.2.2 CML



9.2.3 Estados Genéricos

Los estados que se pueden despertar en cualquier momento del flujo normal son: 'GENERIC FAILURE' y 'C2C FEND'