

P-HAS-5100-FAR

R-HAS-6100-FAR

**Bases de datos: REST y base de datos
genérica**

Contenido

1 – Bases de datos.....	3
GET	4
PUT	8
POST	9
DELETE.....	10
2 - Bases de datos genéricas	10
3 - Acceso a memoria de auditoría	11
Jornadas	11
Jornada.....	14
Documentos por jornada	14
Documento	16
Items	16
Item.....	18
4 – Archivos.....	19
5 – Autorizaciones y restricciones de archivos	20
6 - Información general.....	21
7 - Información de registradora	21
8 - Reportes.....	23
9 - Listas de precios.....	23
10 - Display.....	23

1 - Bases de datos

La arquitectura REST (Transferencia de Estado Representacional) es utilizada para la comunicación con la base de datos de la caja registradora fiscal R-HAS-6100-FAR.

No se pretende explicar en este documento los principios de REST, sino proveer una breve reseña práctica sobre la interacción con la base de datos utilizando estos principios. Para ello, se utilizará el CURL como programa de comunicación entre la PC y la base de datos de la caja, por lo que los ejemplos harán referencia a la sintaxis de CURL.

Se mostrarán cuadros sinópticos de las operaciones REST y sus errores, usando como ejemplo la tabla de PLU.

En las tablas, "myhost" hace referencia a la dirección IP del equipo en el que se encuentra la base de datos.

	http://myhost:8082/plu
GET	Devuelve la base de datos completa, paginada. Devuelve 200 (OK) y un XML/JSON con el resultado, incluyendo los links a los próximos registros. Admite los siguientes parámetros en el URL: offset: a partir de qué registro leer (usado para paginar) limit: cuántos registros reportar por cada solicitud. campo: campo de la base de datos (nombre) y el valor a hacer coincidir. Puede haber más de un valor. <i>Ejemplo:</i> http://myhost:8082/plu?precio=0
PUT	404 (NOT FOUND): no se puede dar de alta usando PUT.
POST	Se espera un XML/JSON con el registro. Devuelve: 201 (CREATED) si se pudo dar el alta. En ese caso en el header de HTTP vuelve un campo Location con el URL del registro creado. 400 (BAD REQUEST) si hay un error leyendo el registro de entrada 500 (INTERNAL SERVER ERROR) si hay un error durante el alta. En esos dos casos vuelve un XML/JSON con el error.
DELETE	Borra todos los artículos. Devuelve: 200 (OK): si se pudo borrar.

	http://myhost:8082/plu/1
GET	Busca la PLU cuyo código sea 1. Devuelve: 200 (OK): la encontró, sigue el registro en XML/JSON de la PLU en cuestión. 404 (NOT FOUND): no la encontró. 500 (INTERNAL SERVER ERROR): no la pudo leer. Sigue un XML/JSON con la descripción del error.
PUT	Modifica la PLU 1 con el registro XML/JSON que se está pasando. El registro debe existir. Primero se lee, luego se actualiza con los campos pasados, y finalmente se escribe. Devuelve: 200 (OK): se pudo modificar. 404 (NOT FOUND): no se encontró el registro. 400 (BAD REQUEST): hubo un error leyendo el registro de entrada 500 (INTERNAL SERVER ERROR): hubo un error durante la modificación. En esos dos casos vuelve un XML/JSON con el error.
POST	404 (NOT FOUND): no se puede dar de alta usando POST y una clave.
DELETE	Borra la PLU 1 de la tabla. Devuelve: 200 (OK): anduvo bien. 404 (NOT FOUND): si no se pudo borrar porque no se encontró o por alguna otra razón.

Ahora vamos a algunos ejemplos usando CURL. Por conveniencia, como dijimos, "myhost" equivale a la dirección de IP de la caja. En caso de tener una conexión mediante proxy, se debe usar **--noproxy myhost**, para evitar pasar por el proxy.

GET

Para leer la PLU 1 se puede usar la siguiente línea de comandos:

```
curl http://myhost:8082/plu/1 --header "Accept: application/json" --write-out "\\nRetcode:
%{http_code}"
```

La opción **--header** le envía el header especificado a continuación como parte de la solicitud. En este caso, se le pasa la opción Accept con el formato que uno quiere (podría haber pasado text/xml). Esto es necesario porque, a diferencia de antes, no hay un archivo real que se le esté pasando con la información de la clave que se quiere ver, de manera que el formato hay que decirlo de otra manera. La última opción, **--write-out**, sirve para imprimir el código de retorno en la pantalla. Entonces, si la PLU 1 existe, devuelve, por ejemplo:

```
{
  "PLU":
  {
    "Codigo": "1",
    "Descripcion": "Verdulería",
    "Grupo": 1,
    "Indicadores":
    [
    ],
    "Precio": 12,
    "IVA": 0,
    "ImpuestoInterno": 0
  }
}
Retcode: 200
```

Es decir, el registro y al final el código de retorno: 200 (OK), que no forma parte del JSON de salida sino que lo imprime la opción **--write-out** que le pasamos a curl.

Si no existiera el registro, volvería:

```
Retcode: 404
```

Ahora, para leer todos los artículos:

```
curl http://myhost:8082/plu --header "Accept: application/json" --write-out
"\\nRetcode: %{http_code}"
```

vuelve:

```
{
  "OK":
  {
    "data":
    [
      {
        "class": "PLU",
        "Codigo": "1",
        "Descripcion": "Verdulería",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 12,
        "IVA": 0,
        "ImpuestoInterno": 0,
        "links":
        [

```

```

        {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/PLU/1"
        }
    ],
    },
    {
        "Codigo": "2",
        "Descripcion": "Panadería",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 0,
        "IVA": 0,
        "ImpuestoInterno": 0,
        "links":
        [
            {
                "class": "link",
                "rel": "self",
                "href": "http://myhost:8082/PLU/2"
            }
        ]
    },
    {
        "Codigo": "3",
        "Descripcion": "Carnicería",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 0,
        "IVA": 0,
        "ImpuestoInterno": 0,
        "links":
        [
            {
                "class": "link",
                "rel": "self",
                "href": "http://myhost:8082/PLU/3"
            }
        ]
    }
],
"links":
[
    {
        "class": "link",
        "rel": "next",
        "href": "http://myhost:8082/PLU?offset=3&limit=3"
    },
    {
        "rel": "first",
        "href": "http://myhost:8082/PLU?offset=0&limit=3"
    },
    {
        "rel": "last",
        "href": "http://myhost:8082/PLU?offset=39&limit=2"
    }
]
}
}
Retcode: 200

```

Vuelve 200 (OK) y una lista en principio con tres registros (PLUs 1, 2 y 3); cada uno incluye un link a sí mismo. Al final hay links para seguir recorriendo los otros registros. Para seguir el próximo link propuesto usando CURL hay que especificar las variables usando **--data**. Cuando aparece al menos un **--data**, CURL asume que es un POST, así que hay que decir ahora sí explícitamente que es un GET (opción **--request**):

```
curl --request GET http://myhost:8082/PLU --header "Accept: application/json"
--data offset=3 --data limit=3 --write-out "\nRetcode: %{http_code}"
```

y devuelve:

```
{
  "OK":
  {
    "data":
    [
      {
        "class": "PLU",
        "Codigo": "4",
        "Descripcion": "Limpieza",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 0,
        "IVA": 0,
        "ImpuestoInterno": 0,
        "links":
        [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/PLU/4"
          }
        ]
      },
      {
        "Codigo": "5",
        "Descripcion": "Lácteos",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 0,
        "IVA": 0,
        "ImpuestoInterno": 0,
        "links":
        [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/PLU/5"
          }
        ]
      },
      {
        "Codigo": "6",
        "Descripcion": "Bebidas",
        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 0,
        "IVA": 0,
        "ImpuestoInterno": 0,

```

```

        "links":
        [
            {
                "class": "link",
                "rel": "self",
                "href": "http://myhost:8082/PLU/6"
            }
        ]
    },
    "links":
    [
        {
            "class": "link",
            "rel": "previous",
            "href": "http://myhost:8082/PLU?offset=0&limit=3"
        },
        {
            "rel": "next",
            "href": "http://myhost:8082/PLU?offset=6&limit=3"
        },
        {
            "rel": "first",
            "href": "http://10.0.7.87:8082/PLU?offset=0&limit=3"
        },
        {
            "rel": "last",
            "href": "http://myhost:8082/PLU?offset=39&limit=2"
        }
    ]
}
Retcode: 200

```

Es decir, las PLUs que siguen (4, 5 y 6) y nuevos links para seguir navegando.

Se puede utilizar parámetros también para filtrar un GET por ciertos campos. Por ejemplo:

```

curl --request GET http://myhost:8082/PLU --header "Accept: application/json"
--data precio=12 --write-out "\nRetcode: %{http_code}"

```

Devuelve el único registro que tiene precio 12.

```

{
    "OK":
    {
        "data":
        [
            {
                "class": "PLU",
                "Codigo": "1",
                "Descripcion": "Frutas y verduras",
                "Grupo": 1,
                "Indicadores":
                [
                ],
                "Precio": 12,
                "IVA": 0,
                "ImpuestoInterno": 0,
                "links":
                [
                    {
                        "class": "link",
                        "rel": "self",
                        "href": "http://10.0.7.87:8082/PLU/1"
                    }
                ]
            }
        ]
    }
}

```

```

    }
  ]
},
"links":
[
  {
    "class": "link",
    "rel": "first",
    "href": "http://10.0.7.87:8082/PLU?offset=0&limit=1&Precio=12"
  },
  {
    "rel": "last",
    "href": "http://10.0.7.87:8082/PLU?offset=0&limit=1&Precio=12"
  }
]
}
}
Retcode: 200

```

Se sabe que es el único porque, además de no haber otras coincidencias (si hubiera dos o tres deberían haber aparecido), no hay "next" ni "previous". En caso de no haber coincidencias, vuelve 404 (NOT FOUND). Esta prestación es tentativa, porque sólo admite consultas por uno o más campos que coincidan (no menor, no mayor, sólo igual) y que coincidan exactamente ("12.00" no coincide si el precio es "12").

PUT

Ahora vamos a modificar la PLU 1. Armamos un registro que tenga sólo la descripción y le ponemos plu1.json:

```

{
  "PLU":
  {
    "Descripcion" : "Frutas y verduras"
  }
}

```

(antes decía "Verdulería")

Para el PUT, usamos la siguiente línea de comandos:

```

curl --request PUT http://myhost:8082/plu/1 --header "Content-Type: application/json" --write-out "\nRetcode: %{http_code}" --data-binary @plu1.json

```

Lo nuevo que aparece en realidad ya era conocido del otro protocolo, que es poner **--data-binary** y el nombre del archivo a poner como parte de la solicitud, precedido de una arroba. El request ahora es PUT (opción **--request**). Esto arroja:

```

Retcode: 200

```

Y podemos corroborar usando un GET de /plu/1 que el PUT funcionó bien:

```

curl http://myhost:8082/plu/1 --header "Accept: application/json" --write-out "\nRetcode: %{http_code}"

```

```

{
  "PLU":
  {
    "Codigo": "1",
    "Descripcion": "Frutas y verduras",

```

```

        "Grupo": 1,
        "Indicadores":
        [
        ],
        "Precio": 12,
        "IVA": 0,
        "ImpuestoInterno": 0
    }
}
Retcode: 200

```

Se puede ver que tiene la descripción cambiada y el resto intacto.

¿Qué sucede si el registro que se le envía tiene algún error? Volvería una respuesta como la siguiente:

```

{
  "Error":
  {
    "Code": "DATAFILE_INVALID_RECORD",
    "Description": "Registro inválido",
    "Context": "No object given"
  }
}
Retcode: 400

```

Vuelve 400 (BAD REQUEST) y un JSON o un XML con el error que disparó el problema.

POST

Para el POST (Alta) vamos a hacer una prueba reutilizando el mismo archivo que en el PUT, sólo que vamos a agregarle la clave. La clave debe ir en el registro y no en el URL, que debe ser sólo el nombre de la tabla donde se da el alta. Ahora plu1.json se ve así:

```

{
  "PLU":
  {
    "Codigo": 1,
    "Descripcion" : "Frutas y verduras"
  }
}

```

y la línea de comandos de POST cambia el **--request** y deja sólo el recurso **"/plu"**:

```

curl --request POST http://myhost:8082/plu --header "Content-Type:
application/json" --write-out "\\nRetcode: %{http_code}" --data-binary
@plu1.json

```

Si se realiza un alta de un artículo que ya existe, para ver la diferencia con PUT, En principio el POST devuelve:

```

Retcode: 201

```

Que significa que el nuevo registro fue creado. Para ver el nuevo registro, se puede ver la solicitud completa usando la opción **--include** del curl:

```

curl --include --request POST http://myhost:8082/plu --header "Content-Type:
application/json" --data-binary @scripts/plu1.json

```

```

HTTP/1.0 201 CREATED
HSL REST Server
Date Wed, 10 Aug 2016 17:51:53 GMT
Connection: close
Access-Control-Allow-Origin: *

```

Location: http://myhost:8082/PLU/1

Ahí puede verse en el dump de la solicitud el **201 CREATED**, y el campo Location con el link al recurso. Si hacemos un GET sobre esa dirección, puede ahora observarse:

```
{
  "PLU":
  {
    "Codigo": "1",
    "Descripcion": "Frutas y verduras",
    "Grupo": 0,
    "Indicadores":
    [
    ],
    "Precio": 0,
    "IVA": 0,
    "ImpuestoInterno": 0
  }
}
Retcode: 200
```

El campo "Descripcion" tiene lo que le dijimos que tenga, pero el resto tiene valores por defecto. Esto significa que el POST ha rescrito el registro original, a diferencia de cuando se hace un PUT.

DELETE

La primera operación que vamos a probar es borrar la PLU 1. Para esto basta con:

```
curl --request DELETE http://myhost:8082/plu/1 --header "Accept:
application/json" --write-out "\nRetcode: %{http_code}"
```

Es como un GET pero la solicitud es DELETE. El resultado es 200 (OK) cuando el registro ha sido borrado. Si se vuelve a ejecutar el mismo comando, el resultado es 404 (NOT FOUND), confirmando que la PLU ha sido efectivamente eliminada.

La última prueba la dejamos para el final, porque luego sólo queda el vacío, la nada:

```
curl --request DELETE http://myhost:8082/plu --header "Accept:
application/json" --write-out "\nRetcode: %{http_code}"
```

Es decir, se pide un DELETE a /plu, no a /plu/1. Esto borra todos los artículos, y devuelve 200 (OK) incondicionalmente.

2 - Bases de datos genéricas

Una de las opciones que ofrece la caja registradora fiscal R-HAS-6100-FAR es la utilización de bases de datos genéricas, es decir cuyo formato se amolde a las necesidades del usuario.

Las bases de datos genéricas son principalmente de uso potencial para aplicaciones Javascript. Si una aplicación web que interactúa con la caja y su base de datos tuviera que guardar información, no tendría donde hacerlo si no fuera por las bases de datos genéricas.

Esta prestación permite crear bases de datos y manejarlas como si fueran bases de datos preexistentes, con la única limitación de que todos sus campos son strings.

Para la creación de una base de datos, se debe acceder desde REST al recurso *userdata*.

POST y PUT sirven para dar de alta nuevas bases de datos. En el caso de que ya exista, se borran todos los datos y se sobrescribe con el formato de la nueva definición. DELETE se utiliza para borrar definiciones, mientras que GET para realizar consultas.

Por ejemplo, si se realiza un PUT del siguiente JSON en IP:8082/userdata/test

```
{
  "Data": {
    "Fields": [
      {
        "className": "Campo",
        "Nombre": "Identificador",
        "Longitud": 10,
        "Clave": true
      },
      {
        "Nombre": "Largo",
        "Longitud": 30
      },
      {
        "Nombre": "Corto",
        "Longitud": 5
      }
    ]
  }
}
```

Se crea una nueva base vacía con tres campos de distinta longitud, y el primero es la clave ("Clave: true"). A partir de ahí se puede utilizar desde IP:8082/test como se usan las otras bases en REST. Nótese que, una vez que la base está creada, se puede acceder a ella directamente a través del nombre que se dio (en este caso "test"), mediante la IP y el puerto correspondiente.

3 - Acceso a memoria de auditoría

Las operaciones anteriores se referían a la base de datos; aquí están las operaciones de acceso a memoria de auditoría. A diferencia de la base de datos, los accesos a auditoría son siempre de lectura (GET y nunca POST, PUT o DELETE). Los ejemplos que siguen están con el curl siguiendo las convenciones del mail anterior, pero bien podrían haberse hecho directamente con el navegador, ya que siempre es GET. Las posibilidades son:

Jornadas

URL: http://myhost:8082/jornadas

Devuelve todas las jornadas que hay en este momento en memoria de auditoría. Por ejemplo:

```
curl http://myhost:8082/jornadas --header "Accept: application/json" --
write-out "\nRetcode: %{http_code}"
```

Devuelve:

```
{
  "OK":
  {
    "data":
    [
```

```

{
  "class": "Jornada",
  "Z": 1,
  "Fecha": "160707",
  "Bajada": true,
  "Borrable": true,
  "links":
  [
    {
      "class": "link",
      "rel": "self",
      "href": "http://myhost:8082/Jornadas/1"
    },
    {
      "rel": "docs",
      "href": "http://myhost:8082/Jornadas/1/Documentos"
    }
  ]
},
{
  "Z": 2,
  "Fecha": "160304",
  "Bajada": true,
  "Borrable": true,
  "links":
  [
    {
      "class": "link",
      "rel": "self",
      "href": "http://myhost:8082/Jornadas/2"
    },
    {
      "rel": "docs",
      "href": "http://myhost:8082/Jornadas/2/Documentos"
    }
  ]
},
{
  "Z": 3,
  "Fecha": "160304",
  "Bajada": true,
  "Borrable": true,
  "links":
  [
    {
      "class": "link",
      "rel": "self",
      "href": "http://myhost:8082/Jornadas/3"
    },
    {
      "rel": "docs",
      "href": "http://myhost:8082/Jornadas/3/Documentos"
    }
  ]
}
],
"links":
[
  {

```

```

        "class": "link",
        "rel": "next",
        "href": "http://myhost:8082/Jornadas?offset=4&limit=3"
    },
    {
        "rel": "last",
        "href": "http://myhost:8082/Jornadas?offset=91&limit=1"
    }
]
}
}
Retcode: 200

```

Todo esto coincide en líneas generales con el diseño del REST de base de datos: "OK" para cuando la operación salió bien, y sigue un array "data" con las jornadas paginadas, y un array links para la paginación en sí. Cada jornada tiene su número y fecha, y un par de booleanos que dice si fue bajada (para reporte AFIP) y si es pasible de ser borrada en caso de necesitarse espacio.

Una variable que se puede pasar, además de las ya conocidas "offset" y "limit", es "from" para poder buscar zetas por fecha. Como el paginado es voluntario, el "to" no hace falta:

```

curl http://myhost:8082/jornadas --request GET --data from=160801 --header
"Accept: application/json" --write-out \nRetcode: %{http_code}

```

Cabe aclarar que al poner datos es necesario **--request GET**. Esto devuelve:

```

{
  "OK": {
    "data": [
      {
        "class": "Jornada",
        "Z": 89,
        "Fecha": "160802",
        "Bajada": false,
        "Borrable": false,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/89"
          },
          {
            "rel": "docs",
            "href": "http://myhost:8082/Jornadas/89/Documentos"
          }
        ]
      },
      {
        "Z": 90,
        "Fecha": "160808",
        "Bajada": false,
        "Borrable": false,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/90"
          },
          {
            "rel": "docs",
            "href": "http://myhost:8082/Jornadas/90/Documentos"
          }
        ]
      }
    ]
  }
}

```

```

    ],
    "links": [
      {
        "class": "link",
        "rel": "last",
        "href": "http://myhost:8082/Jornadas?offset=89&limit=2"
      }
    ]
  }
}
Retcode: 200

```

Es decir, jornadas posteriores a la fecha indicada.

Jornada

URL: <http://myhost:8082/jornadas/z>

Devuelve una jornada en particular (la z del URL) de memoria de auditoría. Por ejemplo:

```
curl http://myhost:8082/jornadas/3 --header "Accept: application/json"
--write-out "\\nRetcode: %{http_code}"
```

devuelve la zeta 3:

```

{
  "Jornada": {
    "Z": 3,
    "Fecha": "160304",
    "Bajada": true,
    "Borrable": true,
    "links": [
      {
        "class": "link",
        "rel": "self",
        "href": "http://myhost:8082/Jornadas/3"
      },
      {
        "rel": "docs",
        "href": "http://myhost:8082/Jornadas/3/Documentos"
      }
    ]
  }
}
Retcode: 200

```

A diferencia del formato de retorno de la base de datos, aquí hay dos tipos de links individuales: el link a la zeta en sí, y un link a los documentos contenidos en la zeta. Si la Z especificada no existiera, devolvería 404 (NOT FOUND)

Documentos por jornada

URL: <http://myhost:8082/jornadas/z/documentos>

Devuelve todos los documentos que hay en una zeta determinada de memoria de auditoría. Por ejemplo:

```
curl http://myhost:8082/jornadas/4/documentos --header "Accept:
application/json" --write-out "\\nRetcode: %{http_code}"
```

Devuelve:

```
{
  "OK": {
    "data": [
      {
        "class": "Documento",
        "Z": 4,
        "Orden": 0,
        "Fecha": "160304",
        "Hora": "115517",
        "Tipo": "DocumentoNoFiscalInterno",
        "SubTipo": "Auditoria",
        "Calificador": "AuditoriaPorFechas",
        "Estacion": "Ticket",
        "Numero": 2,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/4/Documentos/0"
          },
          {
            "rel": "items",
            "href": "http://myhost:8082/Jornadas/4/Documentos/0/Items"
          }
        ]
      },
      {
        "Z": 4,
        "Orden": 1,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/4/Documentos/1"
          },
          {
            "rel": "items",
            "href": "http://myhost:8082/Jornadas/4/Documentos/1/Items"
          }
        ]
      },
      {
        "Z": 4,
        "Orden": 2,
        "Fecha": "160304",
        "Hora": "124903",
        "Tipo": "DocumentoFiscal",
        "SubTipo": "TicketConsumidorFinal",
        "Calificador": "B",
        "Estacion": "Ticket",
        "Numero": 2,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/4/Documentos/2"
          },
          {
            "rel": "items",
            "href": "http://myhost:8082/Jornadas/4/Documentos/2/Items"
          }
        ]
      }
    ]
  },
  "links": [
```

```

    {
      "class": "link",
      "rel": "next",
      "href": "http://myhost:8082/Jornadas/4/Documentos?offset=3&limit=3"
    },
    {
      "rel": "last",
      "href": "http://myhost:8082/Jornadas/4/Documentos?offset=9&limit=2"
    }
  ]
}
}
Retcode: 200

```

Documento

URL: <http://myhost:8082/jornadas/z/documentos/n>

Devuelve el documento *n* de la zeta *z*. Por ejemplo:

```
curl http://myhost:8082/jornadas/4/documentos/2 --header "Accept:
application/json" --write-out "\nRetcode: %{http_code}"
```

Devuelve:

```

{
  "Documento": {
    "Z": 4,
    "Orden": 2,
    "Fecha": "160304",
    "Hora": "124903",
    "Tipo": "DocumentoFiscal",
    "SubTipo": "TicketConsumidorFinal",
    "Calificador": "B",
    "Estacion": "Ticket",
    "Numero": 2,
    "Cancelado": false,
    "Cerrado": true,
    "links": [
      {
        "class": "link",
        "rel": "self",
        "href": "http://myhost:8082/Jornadas/4/Documentos/2"
      },
      {
        "rel": "items",
        "href": "http://myhost:8082/Jornadas/4/Documentos/2/Items"
      }
    ]
  }
}
}
Retcode: 200

```

En este caso, siempre viene con el detalle completo. Al igual que el caso de las zetas, hay un segundo link que permite llegar a los ítems del documento.

Items

URL: <http://myhost:8082/jornadas/z/documentos/n/items>

Devuelve los ítems del documento *n* de la jornada *z*. Por ejemplo:

```
curl http://myhost:8082/jornadas/4/documentos/2/items --header "Accept: application/json" --write-out "\\nRetcode: %{http_code}"
```

devuelve los primeros tres ítems del tercer documento de la cuarta jornada fiscal:

```
{
  "OK": {
    "data": [
      {
        "class": "AtributosImpresion",
        "Z": 4,
        "Numero": 2,
        "Orden": 0,
        "Personalidad": "ECR",
        "Producto": "R-HAS-6100-FAR 6.50A",
        "Modelo": "Hasar R-HAS-6100-FAR",
        "ModeloImpresor": "AB320M",
        "Estacion": "Ticket",
        "AnchoPapel": 42,
        "AltoPapel": 0,
        "TipoHojaAlto": "AltoA4",
        "TipoHojaAncho": "AnchoNormal",
        "ImprimeLeyendasOpcionales": true,
        "ImprimeCodigos": "NoImprimeCodigos",
        "ImprimeUnidadesMedida": false,
        "ImprimeMarcos": false,
        "ImprimeCajero": false,
        "ModoEntrenamiento": false,
        "SignoMonetario": "$",
        "UsarColorAlternativo": false,
        "ColorAlternativo": "Negro",
        "ImprimeQR": true,
        "TipoInterlineado": "InterlineadoNormal",
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/4/Documentos/2/Items/0"
          }
        ]
      },
      {
        "class": "Emisor",
        "Z": 4,
        "Numero": 2,
        "Orden": 1,
        "CUIT": "20231914448",
        "RazonSocial": "Leandro Fanzone",
        "NroRegistro": "ZQECRH0000000001",
        "FechaInicioActividades": "140806",
        "IngBrutos": "Et tu, Brute?",
        "TipoEmisor": "EmisorInscripto",
        "TipoHabilitacion": "ComprobantesA",
        "LimiteM": 1000,
        "links": [
          {
            "class": "link",
            "rel": "self",
            "href": "http://myhost:8082/Jornadas/4/Documentos/2/Items/1"
          }
        ]
      }
    ]
  },
}
```

```

{
  "class": "Apertura",
  "Z": 4,
  "Numero": 2,
  "Orden": 2,
  "TipoDocumento": "DocumentoFiscal",
  "SubTipoDocumento": "TicketConsumidorFinal",
  "CalificadorDocumento": "B",
  "Estacion": "Ticket",
  "TipoAFIP": 83,
  "POS": "00007",
  "Fecha": "160304",
  "Hora": "124903",
  "NumeroDocumento": 2,
  "NumeroCompleto": "P.V.N° 00007 - N° T. 000000002",
  "NumeroCajero": 1,
  "NombreCajero": "Leandro",
  "IdentificadorCopia": "Original",
  "links": [
    {
      "class": "link",
      "rel": "self",
      "href": "http://myhost:8082/Jornadas/4/Documentos/2/Items/2"
    }
  ]
},
"links": [
  {
    "class": "link",
    "rel": "next",
    "href": "http://myhost:8082/Jornadas/4/Documentos/2/items?offset=3&limit=3"
  },
  {
    "rel": "last",
    "href": "http://myhost:8082/Jornadas/4/Documentos/2/items?offset=6&limit=3"
  }
]
}
}
Retcode: 200

```

Item

URL: <http://myhost:8082/jornadas/z/documentos/n/items/i>

Devuelve los ítems del documento *n* de la jornada *z*. Por ejemplo:

```

curl http://myhost:8082/jornadas/4/documentos/2/items/2 --header
"Accept: application/json" --write-out "\\nRetcode: %{http_code}"

```

devuelve el tercer ítem del tercer documento de la cuarta jornada:

```

{
  "Apertura": {
    "Z": 4,
    "Numero": 2,
    "Orden": 2,
    "TipoDocumento": "DocumentoFiscal",
    "SubTipoDocumento": "TicketConsumidorFinal",
    "CalificadorDocumento": "B",
    "Estacion": "Ticket",
    "TipoAFIP": 83,
    "POS": "00007",

```

```

"Fecha": "160304",
"Hora": "124903",
"NumeroDocumento": 2,
"NumeroCompleto": "P.V.N° 00007 - N° T. 000000002",
"NumeroCajero": 1,
"NombreCajero": "Leandro",
"IdentificadorCopia": "Original",
"links": [
  {
    "class": "link",
    "rel": "self",
    "href": "http://10.0.7.87:8082/Jornadas/4/Documentos/2/Items/2"
  }
]
}
}
Retcode: 200

```

4 - Archivos

Esta interfaz en realidad no es REST, sino HTTP, con acceso completo de lectura, escritura y borrado. Su función es proveer acceso HTTP normal para los programas en JavaScript, por ejemplo, la aplicación para programar la caja registradora, el pricechecker, o la futura registradora virtual. Tiene una estructura predefinida con contenido sugerido:

- / (directorio raíz): aquí van todos los html
- /scripts: los programas (JavaScript)
- /css: las plantillas de estilo
- /lib: las librerías JavaScript (jQuery, etc.)
- /images: fotos de artículos, headers, etc.
- /icons: íconos

Esta estructura internamente se divide en dos tipos: ejecutables e imágenes. Un GET a recursos dentro de esos directorios devuelve el archivo en cuestión, como un servidor HTTP común y corriente, salvo que se verifica quién lo accede en algunos casos, y puede negarse el acceso (ver más adelante). Se puede hacer PUT/POST de archivos, y también borrarlos (DELETE). El directorio raíz y los directorio css, scripts y lib admiten archivos de texto, potencialmente ejecutables (html, js, css). Estos archivos, para poder subirlos con POST o PUT (es indiferente cuál) deben estar firmados por Hasar. Antes de firmarlos, se insertan en un contenedor simple que tiene las siguientes líneas ASCII como encabezado:

- Una primera línea con el texto "20130814"
- Una línea que empieza con "status: " y sigue con "public" o "restricted", dependiendo del tipo de acceso que se requiere: **public** lo puede acceder cualquiera, mientras que **restricted** implica que sólo podrá ser accedida por un cliente autorizado (ver más adelante).
- Una línea que empieza con "regnum: " y sigue o bien con un número de registro específico o bien con un asterisco, implicando que cualquiera lo puede utilizar. Este campo se utiliza para que Hasar pueda distribuir un programa específico para un cliente, por ejemplo: un módulo específico que se compra y que, al distribuirlo, se quiere impedir que se use indiscriminadamente en cualquier máquina.
- Una línea en blanco.
- El contenido del archivo en cuestión (el html, css, etc.)

El archivo así generado se firma para que no pueda ser alterado, y eso es lo que se sube utilizando el servicio de POST/PUT. El DELETE, en cambio, no lleva restricciones, porque borrar un archivo no puede ser utilizado comercialmente como un arma de doble filo.

Ahora, los directorios de imágenes sólo admiten imágenes (bmp, png, jpg, gif, ico, svg), y como estos archivos no son ejecutables, no tienen otra restricción que la del espacio utilizado: si no se dispone de una microSD (optativa), las imágenes sólo pueden utilizar un tercio de la NAND (unos 35MB), para no impedir el despliegue de la memoria de auditoría. No se debe firmar los archivos, y los puede subir cualquiera con POST o PUT, o borrarlos con DELETE.

5 - Autorizaciones y restricciones de archivos

Los archivos restringibles son, como se dijo, los ejecutables, y las restricciones consisten en decir qué archivos pueden ser accedidos por qué clientes, en términos de **dirección de MAC** de dichos clientes. Se manejan a través de REST en /auth/filename, donde filename es el nombre del archivo a restringir, no importa en qué directorio se encuentre (siempre dentro de los directorios de ejecutables). Así, hacer un GET sobre un archivo devuelve la lista de clientes (direcciones de MAC) permitidos. Un POST pone un archivo en la lista negra, es decir, ese archivo sólo podrá ser accedido por clientes autorizados. Si el archivo ya estaba en la lista negra, un POST no hace nada; si no estaba, a partir de ese momento nadie puede acceder hasta que no se haga un PUT con al menos un cliente. El PUT, naturalmente, también implica la firma de Hasar. A diferencia de los archivos firmados, aquí se trata de un archivo ASCII con dos líneas: el string "mac: " con una dirección de MAC a continuación (con dos puntos separando cada valor hexa), y otra línea con el string "file: " seguido por el nombre del archivo a autorizar. Estas dos líneas van firmadas por Hasar, y como ya hablamos de REST, metidas dentro un JSON en la forma de un array, que se suministra como dato al PUT de /auth/archivo. En esta operación, entonces, se pone en lista blanca una MAC en particular para un archivo que está en lista negra; si no lo estuviera, pasaría a la lista negra. Un DELETE sobre auth/archivo/mac saca de la lista blanca a una IP particular, pero nunca saca de la lista negra a un archivo.

Ejemplos utilizando CURL:

PROCEDIMIENTO RESERVADO A COMPAÑÍA HASAR

Digamos que se quiere subir un archivo p.html nuevo al directorio raíz. Primero hay que armar el contenedor:

```
echo 20130814 > p.html.auth
echo status: public >> p.html.auth
echo "regnum: *" >> p.html.auth
echo >> p.html.auth
cat p.html >> p.html.auth
```

Luego se debe firmar el archivo con la clave privada de Hasar:

```
openssl cms -nodetach -binary -outform pm -sign -in p.html.auth -signer [certificado] -
inkey [clave privada Hasar]
```

El resultado es un archivo CMS. El contenido del CMS tiene que estar en un archivo con el mismo nombre que se le quiere dar en el servidor, en este caso, p.html. Este nuevo archivo con el contenido firmado se sube al servidor utilizando POST:

```
curl --request POST $myhost:8082/p.html --header "Content-Type: application/x-pem-file"
--write-out "%{http_code}" --data-binary @p.html
```

Si todo anduvo bien, curl imprime en pantalla "201" (el código de recurso creado), y se puede *“recibir el saludo”* direccionando el navegador a http://\$myhost:8082/p.html

Ahora vamos a restringir el acceso a ese archivo:

```
curl --request POST $myhost:8082/auth/p.html --header "Accept: application/json" --
write-out "%{http_code}"
```

A partir de ese momento, p.html *“retira el saludo”* por navegador hasta que se autorice el acceso a alguien, y devuelve a cambio un mensaje de denegación.

Para poder habilitar la propia máquina, hay que conocer la dirección de MAC. En caso de Linux, la manera más fácil es a través de **ifconfig**, o través del comando **ip**:

```
echo mac: $(ip addr | grep link/ether | awk '{print $2}') > mac
echo file: p.html >> mac
```

Se firma el archivo:

```
openssl cms -nodetach -binary -outform pem -sign -in mac -out mac.pem -signer  
[certificado] -inkey [clave privada Hasar]
```

y se obtiene un mac.pem similar a cuando se firma un archivo. Luego ese PEM hay que meterlo dentro de un array de JSON, línea por línea. Le ponemos un nombre ("auth.json") al archivo y después hay que pasarlo con CURL por la interfaz REST :

```
curl --request PUT --noproxy $myhost $myhost:8082/auth/p.html --header "Content-Type:  
application/json" --write-out "%{http_code}" --data-binary @auth.json
```

Eso devuelve 200 cuando anduvo bien.

6 - Información general

Se accede con el recurso **/informacion** y sólo admite GET. Devuelve información general del controlador, y está disponible en todas las versiones (registradora, impresora fiscal o incluso pricechecker standalone). Devuelve un objeto información con los siguientes campos:

- **FechaActual:** AAMMDD
- **HoraActual:** HHMMSS
- **IPCliente:** la IP que el cliente (navegador) está utilizando, no la IP del controlador fiscal.
- **Version:** La versión oficial del producto (ejemplo: 1.00)
- **Motor:** La versión de Durango (ejemplo: 6.10)
- **NumeroSerie:** en controladores, el número de registro. En pricechecker standalone, el número de MAC.
- **Marca:** Hasar, Crams, etc.
- **Producto:** Nombre oficial del producto (ejemplo: R-HAS-6100-FAR)
- **AnchoDescripcion:** el ancho disponible para la impresión de la descripción de un artículo.
- **AnchoLogo:** el ancho en pixeles admitido para el logo de usuario.
- **AltoLogo:** el alto en pixeles admitido para el logo de usuario.
- **LineasUsuario:** la descripción de las distintas zonas de líneas de usuario, en la forma de un array de objetos con los siguientes campos:
 - **Tipo:** HeaderZone1, Fantasy, OwnerAddress, etc.
 - **Nombre:** nombre arbitrario o descripción del tipo.
 - **Maximo:** máxima cantidad de líneas disponible en esa zona.
- **AnchoLineaUsuario:** cantidad máxima de caracteres en una línea de usuario cualquiera.
- **EstacionPorDefecto:** nombre de la estación por defecto (útil para saber a dónde mandar las líneas de usuario)
- **PuertoPrincipalHTTP:** si bien el puerto 8082 es inmutable en REST, el puerto por defecto de HTTP puede ser distinto en el controlador fiscal (80) y en el emulador (8080, o programado). Aquí puede saberse.

7 - Información de registradora

Es similar al punto anterior, pero con campos específicamente asociados a la registradora: su estado, sus constantes, etc. Parámetros asociados al recurso **/informacion**.

- **InicioJornadaFiscal:** booleano, si está o no en el inicio de la jornada, es decir, no se emitió ningún documento luego de la zeta.
- **Bloqueo:** booleano, si el controlador se encuentra bloqueado.
- **DescripcionBloqueo:** la descripción de la razón del bloqueo. Si no está bloqueado, no aparece.
- **ContextoBloqueo:** el contexto de la razón del bloqueo. Si no está bloqueado, no aparece.
- **CicloVida:** momento en el que se encuentra el controlador fiscal en este instante. Las posibilidades son:
 - Incierto: no se sabe por razones ajenas a esta editorial.
 - No inicializado: no tiene número de registro, pre-producción.
 - Modo entrenamiento: tiene número de registro, pero no datos fiscales.
 - Modo fiscal: tiene todos los datos fiscales cargados.

- Memoria fiscal llena: la memoria fiscal se llenó.
- Memoria de auditoría completa: la memoria de auditoría se completó.
- Memoria de auditoría cerrada: la memoria de auditoría se completó y no está abierta a más entradas.
- **Estado:** el estado del controlador fiscal. Las posibilidades, autodocumentadas, son estas, aunque no todas sean posibles realmente con la caja registradora:
 - No inicializado
 - Inicio jornada fiscal
 - En jornada fiscal
 - Memoria de auditoría completa, esperando última zeta
 - Memoria de auditoría completa
 - Esperando dar de baja
 - Impresor dado de baja
 - Controlador fiscal bloqueado
 - Inicializando el controlador fiscal
 - Imprimiendo reporte
 - Realizando cierre Z
 - Realizando cierre X
 - Estado 1 previo a cierre
 - Estado 2 previo a cierre
 - Estado 3 previo a cierre
 - Cerrando un documento
 - Cerrando un documento no fiscal
 - Cancelando documento
 - Realizando auditoría
 - Cambiando parámetros de red
 - Documento abierto, vendiendo items
 - Documento abierto, imprimiendo texto fiscal
 - Documento abierto, se realizó una operación global sobre IVA
 - Documento abierto, se realizó un ajuste
 - Documento abierto, se realizó un anticipo
 - Documento abierto, se emitió el comando de otros tributos
 - Documento abierto, pagando
 - Documento abierto, se acaba de imprimir una línea con el concepto del recibo
 - Un comprobante no fiscal genérico se encuentra abierto
 - Recuperando documento, paso 1
 - Recuperando documento, paso 2
 - Recuperando documento, paso 3
 - Cancelando copias
 - Estado de impresión de copias en cierre
 - Estado final de cierre
 - Haciendo copia de documento
 - Estado desconocido
- **JornadaActual:** aparece sólo si la memoria fiscal está en condiciones de afirmar algo. Es el número de jornada que se cerrará cuando se haga la zeta.

Parámetros asociados al recurso **/ecr/informacion**.

- **NumeroCajero:** el número de cajero en funciones, si es que hay uno.
- **NombreCajero:** el nombre del cajero en funciones, si es que hay uno.
- **AnchoDisplay:** el ancho en caracteres que informa el driver de display. En el caso de la 6100 es 16.
- **AltoDisplay:** el alto en líneas que informa el driver de display. En el caso de la 6100 es 4.
- **DistribucionTeclado:** el nombre de la distribución de teclado que está utilizando en este momento la caja registradora. En la 6100 siempre será "ZQECR1200", pero puede cambiar en el caso de las registradoras virtuales.

8 - Reportes

Se generan haciendo GET a **/ecr/reportes**. Los parámetros se pasan como variables de URL. Un ejemplo de un reporte sería un GET a **\$myip:8082/ecr/reportes?CerrarJornada=true** para hacer una zeta. Es un GET y no un POST, que sería lo intuitivo, porque en realidad lo que se obtiene (GET) es el reporte en JSON/XML, mientras que el servidor, en términos de REST, no se modifica realmente, con lo cual POST y PUT, asociados a altas y modificaciones, no serían verbos correctos.

Los parámetros y sus opciones posibles se detallan a continuación:

ImprimirAcumulados	Define si se imprime el resultado de la acumulación. Valores posibles: - Json: true/false - Xml : si/no
FechaDesde	Fecha de inicio del reporte, formato AAMMDD
FechaHasta	Fecha final del reporte, formato AAMMDD
Tipo	Indica el tipo de acumulación, valores posibles: - AcumuladoFamilias - AcumuladoPLUS
Detallado	Indica si el reporte es de totales o detallado, valores posibles: - Json: true/false - Xml : si/no
OpcionCajero	Opción válida para el tipo de reporte 'AcumulacionFamilias' y los valores posibles son: - AcumulacionUnCajero - AcumulacionTodosLosCajeros - AcumulacionTotalCajeros - AcumulacionTotalDiario
Cajero	Opción válida para el tipo de reporte 'AcumulacionFamilias' e indica el número de cajero a acumular.
CerrarJornada	Indica si se cierra la jornada luego de ejecutar el reporte. Los valores posibles son: - Json: true/false - Xml : si/no

9 - Listas de precios

Un programa de configuración querría establecer la lista de precios actual. El recurso que se utiliza en este caso es **/ecr/listaprecios**. Si se hace un GET se obtiene un objeto con un campo ListaPreciosActual con el número de la lista de precios vigente. Si se hace un POST con un objeto equivalente, se modifica.

10 - Display

El recurso es **/ecr/display**. Se puede utilizar si el aplicativo debe mostrar un mensaje por el display de la registradora, para informar de un estado o una condición, por ejemplo. Sólo acepta POST con un objeto que tenga un campo

Texto y opcionalmente un campo línea. Si el campo línea no es suministrado, se utilizan todas las líneas para mostrar el mensaje. Si el texto está vacío o no es suministrado, se borra la línea en cuestión o todas las líneas si no se especificó ninguna.



Noviembre 2, 2016 – Rev. 002

COPYRIGHT © 1997/2016 - Compañía HASAR SAIC

_ El presente documento se halla sujeto a cambios sin previo aviso.

_ **Cía. HASAR SAIC** no asume responsabilidad alguna por errores u omisiones contenidas en este documento, ni asume responsabilidad alguna por los datos y/o perjuicios que el uso de esta información pudiera causar.

Marcos Sastre 2214 [B1618CSD] Ricardo Rojas | Tigre | Buenos Aires | Argentina

Tel: [54.11] 4117.8900 | Fax: [54.11] 4117.8998 | www.grupohasar.com