

Tutorial. Programación en Abanq (II). Acciones

En el capítulo anterior de esta serie de artículos sobre programación en Abanq vimos cómo realizar pequeñas modificaciones sobre módulos ya existentes. Básicamente eran modificaciones sobre tablas y formularios en las que añadíamos campos o modificábamos sus propiedades.

En el presente artículo describiremos la forma de realizar lo que Abanq denomina acciones completas. Esto quiere decir que aprenderemos a crear nuestras propias tablas y formularios, y a relacionarlos con otros elementos ya existentes.

Además, dotaremos a nuestros formularios de funcionalidades añadidas mediante la programación de scripts, veremos la estructura del lenguaje QSA y usaremos las clases más importantes que Abanq pone a disposición de los programadores.

Para terminar, confeccionaremos un informe sencillo para obtener un resumen bien presentado de los datos contenidos en las nuevas tablas.

Antes de comenzar

Realizaremos todos los ejemplos prácticos sobre el módulo Principal del área Facturación. Por ello es necesario acceder a una base de datos con este módulo cargado, tal y como hicimos en el artículo anterior.

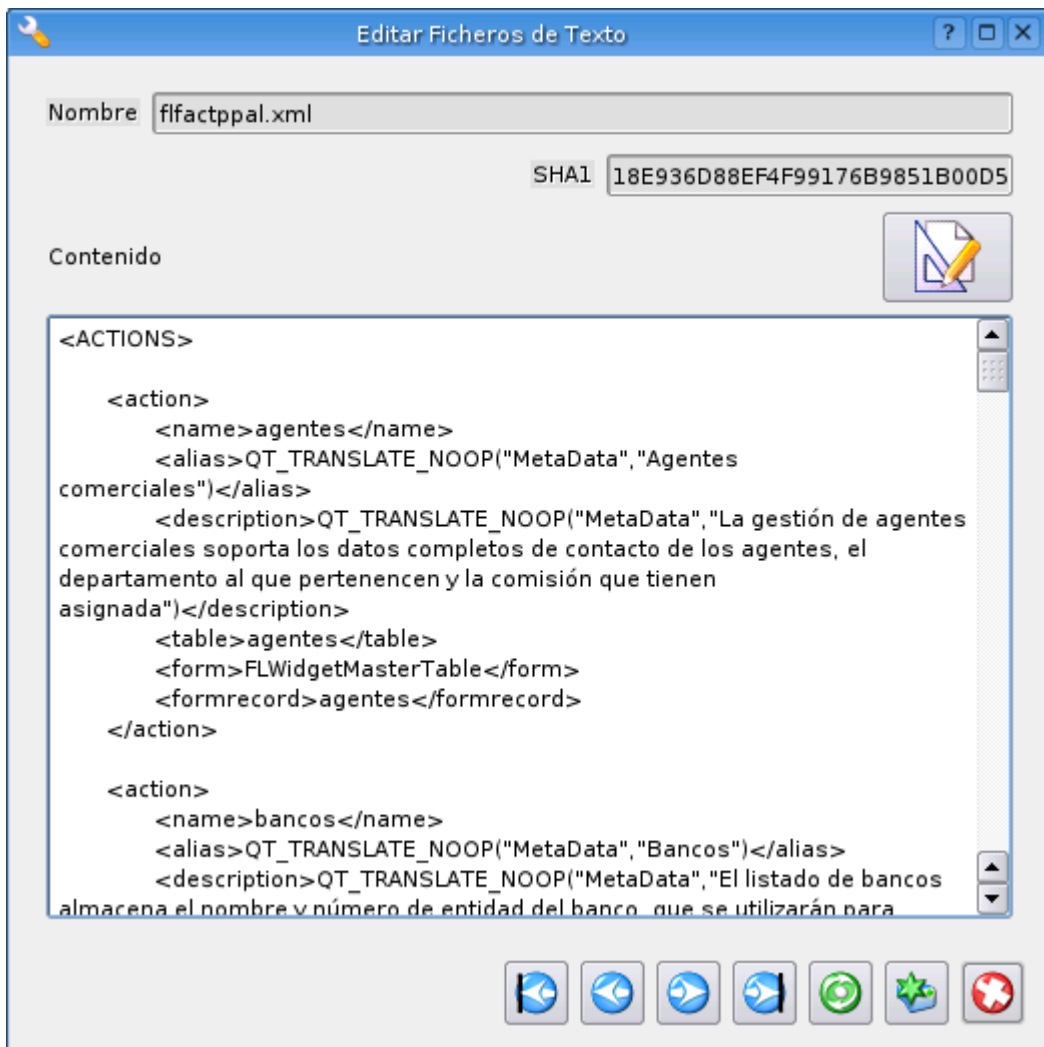
Si queremos disponer de una base de datos nueva para realizar estos ejemplos, basta con especificar el nuevo nombre de base de datos al arrancar Abanq, y cargar a continuación el módulo Principal de Facturación.

Las acciones en Abanq

Como vimos en el artículo anterior, el motor de Abanq lee e interpreta los metadatos (información sobre tablas, formularios, scripts, etc.) que el Sistema Gestor de Base de Datos (SGDB) le proporciona. Estos metadatos se organizan en lo que llamamos módulos. Cada módulo agrupa un conjunto de metadatos que implementan una funcionalidad concreta (facturación, almacén, etc.).

Vamos a ver ahora la estructura interna de estos módulos. El fichero de metadatos que define esta estructura es el fichero de acciones, y su nombre es `id_modulo.xml`, donde `id_modulo` es el identificador que cada módulo tiene asignado en la tabla Módulos del módulo de Administración.

Tomaremos como ejemplo el módulo Principal del área de Facturación. Su código es `ffactppal`, por tanto el fichero de acciones será `ffactppal.xml`. Lo visualizaremos de la forma ya descrita en el artículo anterior (Área de Sistema -> Módulo de Administración -> Módulos -> Editar `ffactppal` -> Seleccionar `ffactppal.xml` -> Ver registro). Aparecerá una ventana similar a la de la figura 1:



1. Fichero de acciones *fifactppal.xml*

Vemos que el fichero de acciones está compuesto por nodos `<action>`. Cada uno de estos nodos define una acción. Una acción es una unidad funcional concreta, que agrupa una serie de elementos necesarios para su correcto funcionamiento. Cada acción puede contener los nombres de una tabla, un formulario maestro, un formulario de edición, un script de formulario maestro y un script de formulario edición.

Por ejemplo, la acción clientes agrupa las funcionalidades de la gestión de clientes: la tabla donde se almacenan sus datos, los formularios utilizados para acceder a dichos datos, los scripts que gestionan los formularios, etc.

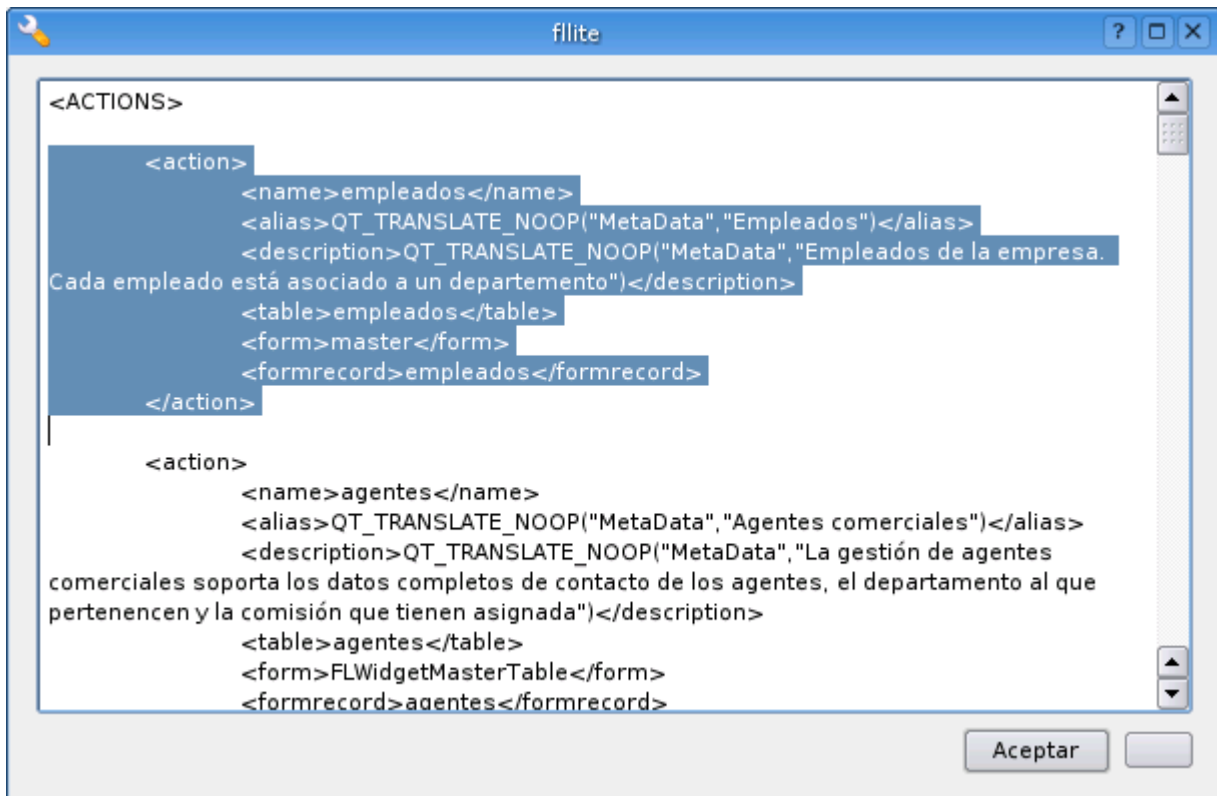
Las etiquetas que conforman cada acción son:

- `<name>` Nombre de la acción.
- `<alias>` Alias o título de la acción
- `<description>` Descripción de la funcionalidad acción
- `<table>` Nombre de la tabla asociada a la acción.
- `<form>` Nombre del formulario maestro asociado a la acción.
- `<formrecord>` Nombre del formulario de edición asociado a la acción.
- `<scriptform>` Nombre del script asociado al formulario maestro.
- `<scriptformrecord>` Nombre del script asociado al formulario edición.

Veremos qué significa cada una de estas etiquetas a medida que progreseemos en el desarrollo de nuestro ejemplo.

Creando nuestra acción

Vamos a suponer que nos es necesario llevar un control de los empleados de nuestra empresa. Para ello es necesario recoger los datos personales de cada uno de ellos. Cada empleado está asociado a un departamento de la empresa, y queremos poder emitir informes con un listado de los empleados de alta agrupados por departamento. Para realizar esta ampliación, el primer paso será crear la acción empleados (la acción departamentos ya existe en el módulo principal). Editaremos el fichero de acciones `ffactppal.xml`, añadiendo un nuevo nodo `<action>` tal y como aparece en la figura 2 (recuerda que para editar el fichero debes pulsar Editar Registro y seguidamente Editar Fichero).



2. Acción empleados en `ffactppal.xml`

En la nueva acción indicamos que la tabla de empleados debe definirse en el fichero `empleados.mtd`, y que los formularios maestro y de edición (veremos qué significan estos términos más adelante) son `i_master.ui` y `empleados.ui`. Por ahora no indicaremos los nombres de los scripts. Una vez guardados los cambios (*Aceptar* -> *Aceptar cambios del fichero* -> *Aceptar cambios del módulo*) ya tenemos nuestra acción creada.

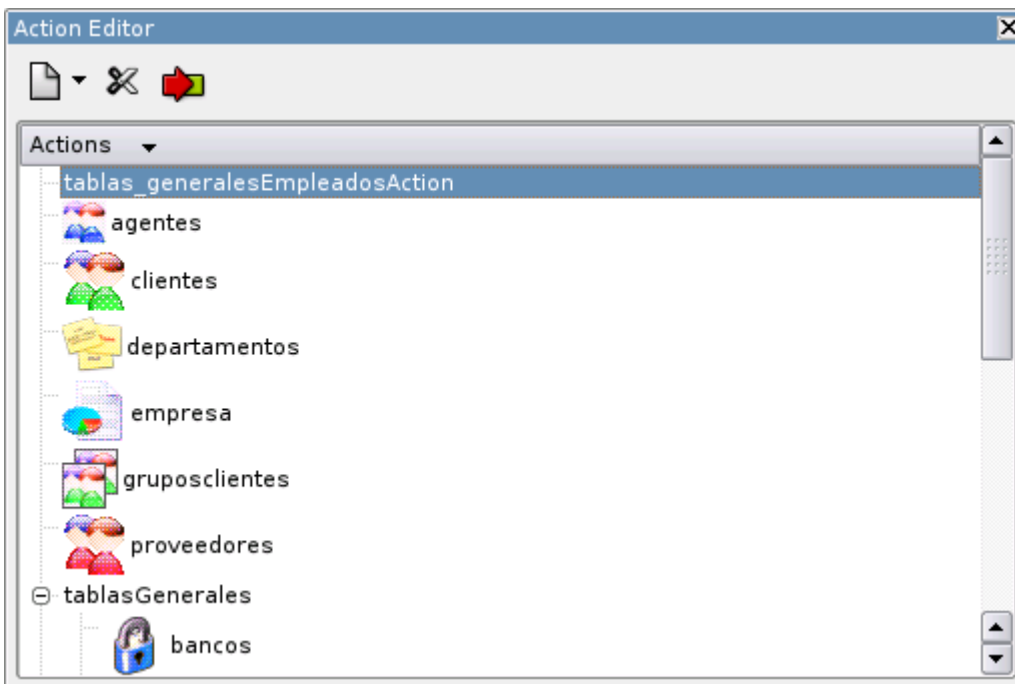
Para terminar este apartado, vamos a crear un acceso desde la ventana principal del módulo de Facturación, de manera que podamos acceder a la gestión de empleados desde una opción de menú o desde un botón de la barra de herramientas.

Las ventanas principales de cada módulo son formularios cuyo nombre sigue el esquema `id_modulo.ui`. En nuestro caso, deberemos editar el formulario `ffactppal.ui`. Una vez abierto el formulario mediante QtDesigner, incluiremos una nueva opción Empleados en el menú Tablas Generales (figura 3).



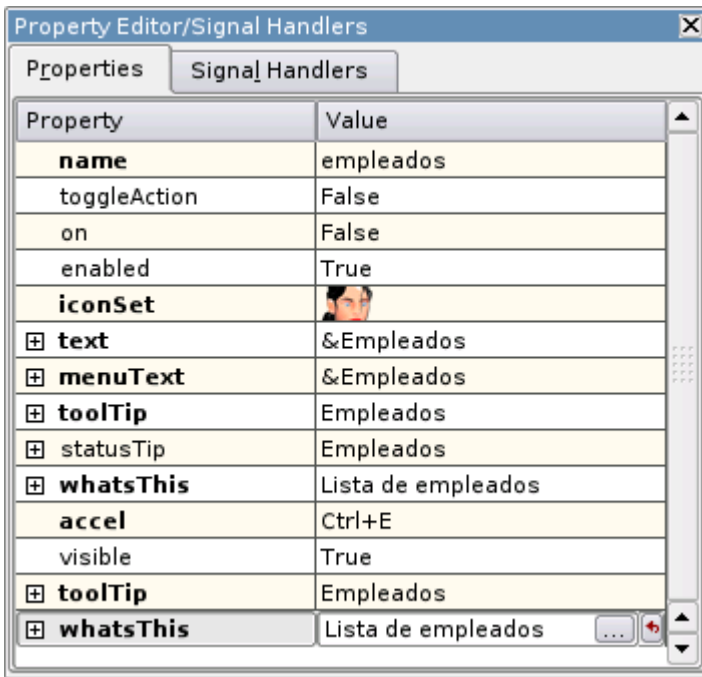
3. Menú Tablas Generales de flfactppal.ui

Al crear esta nueva opción y pulsar Intro hemos creado una nueva acción en la ventana principal del módulo. Para editar esta acción debemos abrir el editor de acciones (opción de menú Window -> Views -> Action Editor). Vemos que se ha creado una acción cuyo nombre por defecto es `tablas_generalesEmpleadosAction` (figura 4).



4. Nueva acción creada en el Action Editor

Cambiaremos este nombre y el resto de propiedades de la acción en la ventana de propiedades (Property Editor) como muestra la figura 5.



5. Property Editor de QtDesigner

Como podemos ver, hemos añadido un icono y especificado las teclas de acceso rápido a la acción. Para incluir un botón en la barra de herramientas arrastraremos el icono desde el Action Editor hasta la posición de la barra en la que deseemos ubicar el botón.

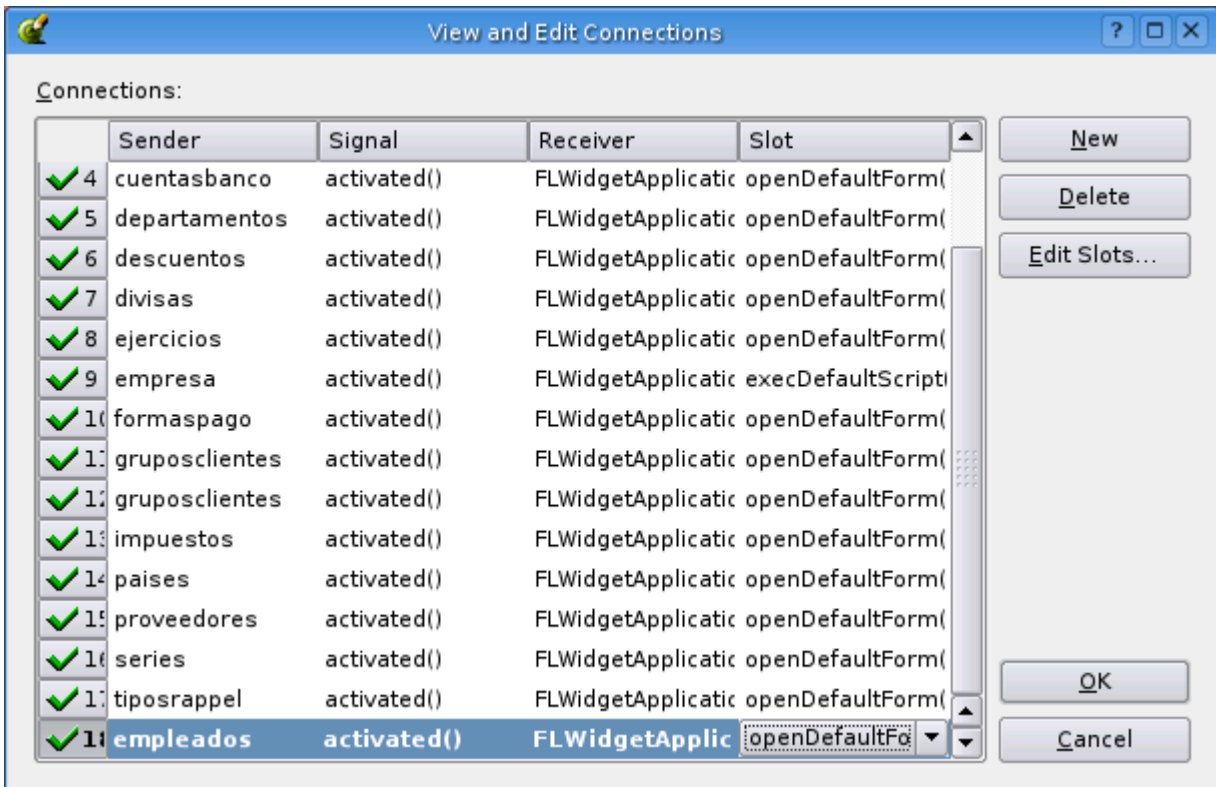
Cada acción de la ventana principal del módulo debe corresponderse con una acción del fichero de acciones. Esta correspondencia se establece haciendo coincidir la propiedad name de la acción de la ventana con la etiqueta <name> del correspondiente nodo action del fichero de acciones. En nuestro caso este valor es empleados.

Nos falta por último determinar qué sucederá cuando se seleccione la acción empleados en la ventana principal del módulo. Lo que haremos será abrir el formulario por defecto asociado a la acción. Para ello, en el Action Editor, seleccionamos la acción empleados y pulsamos el botón de conexiones:

En la ventana View and Edit Connections nos aparece la lista de conexiones establecidas. La última entrada de la lista nos propone conectar la acción empleados con el objeto *FLWidgetApplication*.

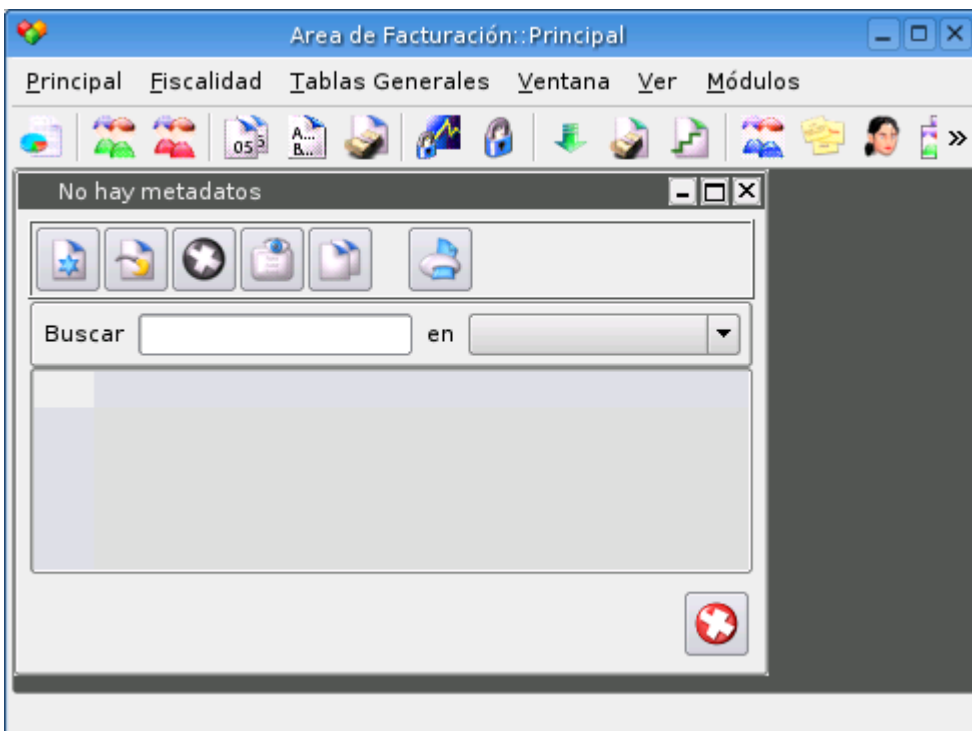
Como veremos más adelante, es muy común en la arquitectura de Abanq establecer conexiones entre objetos. Estas conexiones determinan que cuando un objeto -el emisor- envíe una determinada señal, otro objeto -el receptor- ejecutará un determinado método o slot.

En nuestro caso, deseamos que cuando el objeto acción empleados se active (emita la señal activated), el objeto ventana principal de la aplicación Abanq (*FLWidgetApplication*) muestre el formulario por defecto de la acción (slot *openDefaultForm*). La conexión debe quedar por tanto tal y como describe la figura 6.



6. Ventana de conexiones

Una vez establecida la conexión pulsamos Ok y guardamos los cambios en flfactppal.ui. Ya podemos probar nuestra acción. Si accedemos al módulo Principal del área de Facturación y pulsamos el botón o la opción de menú Empleados, Abanq nos mostrará una ventana como la de la figura 7.



7. Formulario de la acción empleados

El mensaje 'No hay metadatos' hace referencia a que no hemos definido todavía la tabla empleados. En el siguiente punto veremos cómo hacer esto.

Creando las tablas

Las tablas se definen dentro del directorio *tables* de los módulos, y tienen la extensión *mtd*.

El nombre del fichero de la tabla, según nuestra nomenclatura, debe ser *empleados.mtd*. Insertamos un registro en el módulo, igual que hicimos con la creación del fichero de acciones, para la nueva tabla con el nombre anterior y el contenido siguiente:

```
<!DOCTYPE TMD> <TMD>
  <name>empleados</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","Empleados")</alias>

  <field>
    <name>codempleado</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Código")</alias>
    <null>>false</null>
    <pk>>true</pk>
    <type>string</type>
    <length>18</length>
  </field>

  <field>
    <name>coddepartamento</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Departamento")</alias>
    <null>>false</null>
    <pk>>false</pk>
    <type>string</type>
    <length>6</length>

    <relation>
      <table>departamentos</table>
      <field>coddepartamento</field>
      <card>M1</card>
    </relation>
  </field>

  <field>
    <name>nombrecompleto</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Nombre Completo")</alias>
    <null>>false</null>
    <pk>>false</pk>    <!--Código de departamento-->
    <type>string</type>
    <length>150</length>
  </field>

  <field>
    <name>titulacion</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Titulación")</alias>
    <null>>false</null>
    <pk>>false</pk>
    <type>string</type>
    <length>20</length>
    <optionslist>Ninguna,Ed. Primaria,Ed. Secundaria,FP,Título Universitario</optionslist>
    <default>OFICINA</default>
  </field>

  <field>
    <name>fechaalta</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Fecha de alta")</alias>
    <null>>false</null>
```

```
<pk>>false</pk>
<type>date</type>
</field>
```

```
<field>
  <name>debaja</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","De baja")</alias>
  <null>>false</null>
  <pk>>false</pk>
  <type>bool</type>
  <default>>false</default>
</field>
```

```
<field>
  <name>sueldobruto</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","Bruto")</alias>
  <null>>true</null>
  <pk>>false</pk>
  <type>double</type>
  <partI>4</partI>
  <partD>2</partD>
</field>
```

```
<field>
  <name>impuestos</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","% Impuestos")</alias>
  <null>>true</null>
  <pk>>false</pk>
  <type>double</type>
  <partI>2</partI>
  <partD>2</partD>
</field>
```

```
<field>
  <name>sueldoneto</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","Neto")</alias>
  <null>>true</null>
  <pk>>false</pk>
  <type>double</type>
  <partI>4</partI>
  <partD>2</partD>
</field>
```

```
<field>
  <name>causabaja</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","Causa de la baja")</alias>
  <null>>true</null>
  <pk>>false</pk>
  <type>stringlist</type>
</field>
```

```
</TMD>
```

El campo seccion lo hemos definido como una lista de opciones con varias secciones de la empresa a modo de ejemplo. Es muy recomendable repasar la estructura xml de esta tabla y comprender bien todos los campos y etiquetas. Podemos repasar las especificaciones de las tablas.

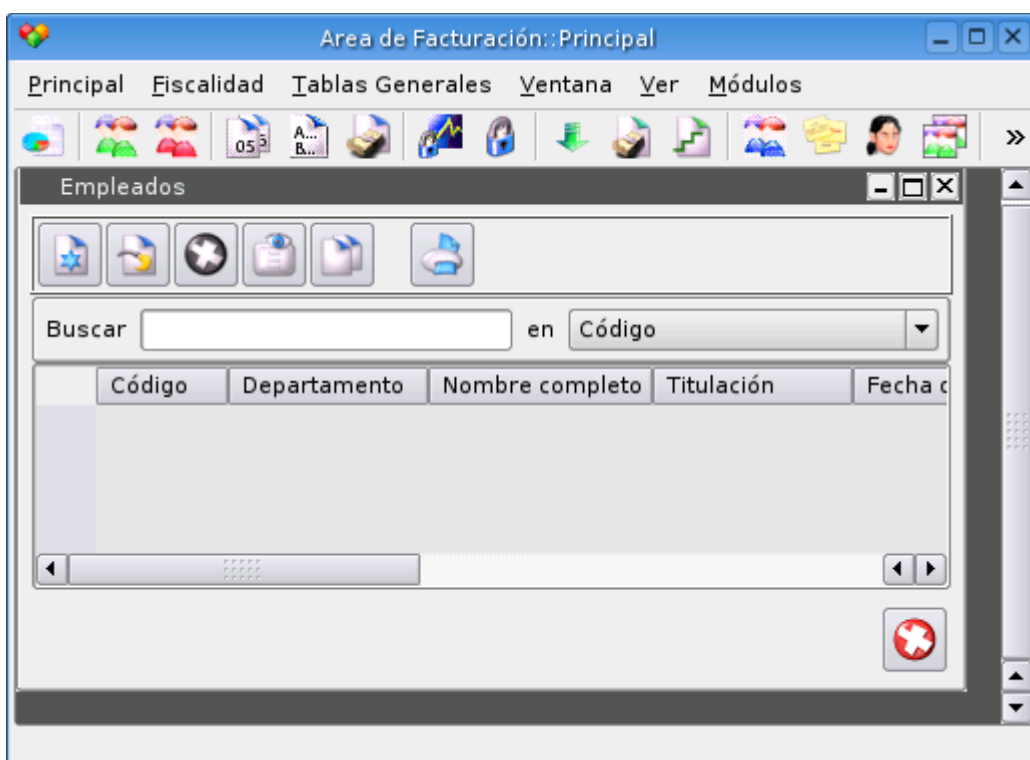
Pulsamos Aceptar cambios y cerrar formulario.

Creando los formularios

Abanq usa dos tipos principales de formularios: El formulario maestro y el formulario de edición. Veamos cómo es y cómo se crea cada uno de ellos.

El formulario maestro es el que muestra en una lista un subconjunto de los registros de la tabla, ofreciendo al usuario la posibilidad de realizar ciertas operaciones (crear, modificar, borrar, etc.) sobre el registro seleccionado. Sirve además para localizar un determinado registro realizando una búsqueda por cualquiera de los campos de la tabla. El formulario maestro asociado a una acción está determinado por el valor de la etiqueta `<form>` del nodo `<action>`.

En el caso de la acción de empleados, hemos nombrado a este formulario `i_master`. Éste no es un nombre al azar, sino el de un formulario predeterminado, ya incluido en Abanq. De hecho, podemos volver a pulsar sobre la acción de empleados y veremos que ya nos aparece un formulario Empleados con los campos de la tabla (figura 10).



10. Formulario maestro de empleados

En el caso de que quisiéramos crear un formulario maestro personalizado, procederíamos de forma similar a como se describe a continuación para el formulario de edición.

El formulario de edición es el que ofrece al usuario la posibilidad de visualizar, crear o modificar los datos de un determinado registro de la tabla. El formulario de edición asociado a una acción está determinado por el valor de la etiqueta `<formrecord>` del nodo `<action>`.

Vamos a crear el formulario de edición correspondiente a la acción empleados. Lo habíamos llamado empleados, luego el fichero deberá llamarse `empleados.ui`. Siguiendo el mismo procedimiento que para crear la tabla `empleados.mtd`, crearemos un nuevo registro en el módulo `flfactppal`. Al tener el fichero la extensión `.ui`, Abanq lanzará QtDesigner en lugar del editor de textos. En el cuadro de diálogo `New File` seleccionaremos el tipo `Widget`.

Nuestro formulario de edición deberá ser similar al de la figura 11.

11. Formulario empleados.ui

Hemos agrupado los controles FLFieldDB en tres controles groupBox para mayor claridad, aunque esto no es necesario.

Las principales propiedades de cada campo son las siguientes:

- 1: fdbCodEmpleado codempleado
- 2: fdbTitulacion titulacion
- 3: fdbNombreCompleto nombrecompleto
- 4: fdbCodDepartamento coddepartamento
- 5: fdbNomDepartamento nombre departamentos coddepartamento coddepartamento
- 6: fdbFechaAlta fechaalta
- 7: fdbDeBaja debaja
- 8: fdbSueldoBruto sueldobruto
- 9: fdbImpuestos impuestos
- 10: fdbSueldoNeto sueldoneto
- 11: fdbCausaBaja causabaja

Hemos optado por establecer los nombres de los controles como *fdb + NombreCampo*. Aunque la propiedad name puede tomar cualquier valor, en este ejemplo es recomendable mantener los de la tabla anterior, para que los scripts que crearemos a continuación no tengan que ser retocados.

En el campo 5 vamos a mostrar el nombre del departamento al que pertenece el empleado. Como este campo no pertenece a la tabla de empleados sino a la de departamentos, debemos establecer el resto de propiedades tal y como ya describimos en el artículo anterior.

Una vez guardado el formulario ya podemos probarlo. Si abrimos la acción empleados aparecerá el fomulario maestro (*i_master.ui*). Si pulsamos ahora sobre el botón Insertar Registro, se abrirá nuestro formulario *empleados.ui* con el aspecto de la figura 12.

12. Formulario de edición de empleados

Podemos apreciar cómo dependiendo del tipo de campo, el control *FLFieldDB* correspondiente toma el aspecto adecuado para mostrar su valor.

Llegados a este punto ya podríamos comenzar a trabajar con esta ventana. Los botones de aceptar y cancelar, así como las validaciones de datos de cada campo están plenamente operativos, de forma que si establecemos todos los campos requeridos (marcados como `<null>false</null>` en `empleados.mtd`) y pulsamos aceptar habremos creado nuestro primer empleado.

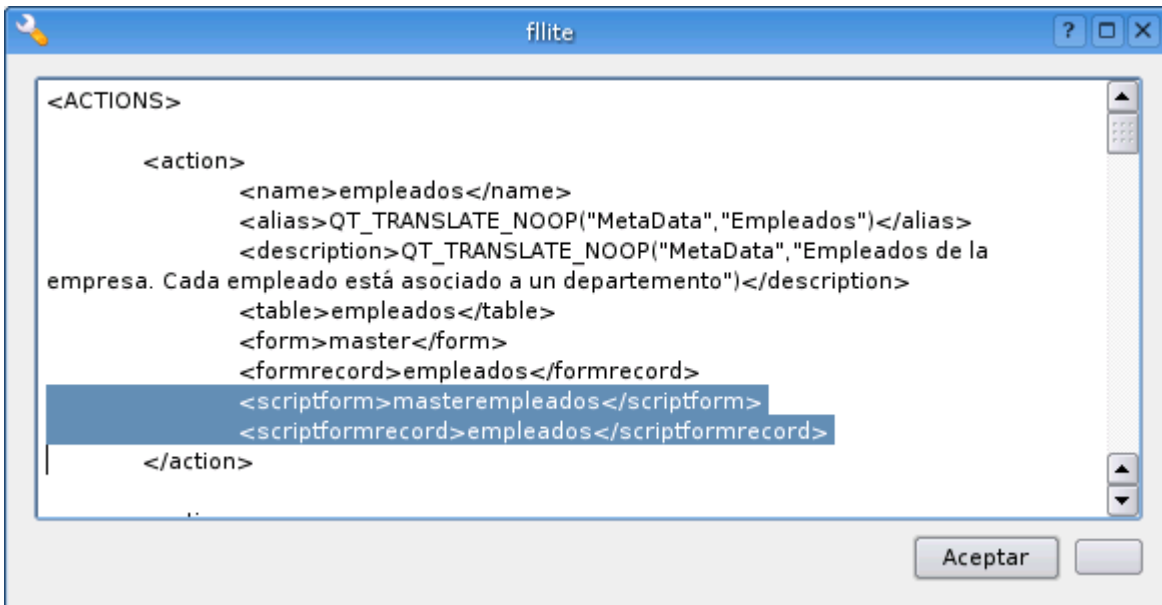
Creando los scripts

Vamos a dotar a nuestros formularios de una mayor funcionalidad asociándoles un script. No vamos a hacer una descripción demasiado formal del lenguaje QSA usado en Abanq, simplemente diremos que es muy similar a JavaScript e incluiremos comentarios en el código de los ejemplos que aclaren su funcionamiento.

Algo que sí es importante recalcar es que, al margen de las clases y funciones que QSA ofrece al programador de scripts, el motor de Abanq también publica una serie de clases que nos permiten acceder a ciertos objetos internos del motor. Como veremos, esto da una gran potencia a los scripts, ya que permite hacer muchas operaciones que de otra manera sólo podrían conseguirse recompilando el código C++ del motor.

Podemos consultar la documentación de QSA en <http://doc.trolltech.com/qsqa-1.2/index.html>, y la de la interfaz de objetos de Abanq en el apartado de documentación de esta web.

Lo primero será añadir las referencia a los scripts en el correspondiente nodo `<action>` del fichero de acciones (figura 13).



13. Añadiendo las referencias a los scripts en la acción empleados

Hemos asociado al formulario maestro el script *masterempleados.qs*, y al formulario de edición el script *empleados.qs*.

Crearemos primero el script asociado al formulario de edición, *empleados.qs*. A continuación mostramos el código completo del script que contiene algunas de las principales funciones que son llamadas automáticamente por el motor de Abanq en ciertos momentos de la ejecución del formulario.

El sistema de clases y herencias escapa al ámbito de este tutorial y se verá con posterioridad.

```

/*****
/*****
/** @file */
/** @class_declaration interna */
////////////////////////////////////
/// DECLARACION //////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
/// INTERNA //////////////////////////////////////
class interna {
  function calculateField(fN:String):String { return this.ctx.interna_calculateField(fN); }
}
/// INTERNA //////////////////////////////////////
////////////////////////////////////
/** @class_declaration oficial */
////////////////////////////////////
/// OFICIAL //////////////////////////////////////
class oficial extends interna {
  function oficial( context ) { interna( context ); }
  function bufferChanged(fN:String) {
    return this.ctx.oficial_bufferChanged(fN);
  }
}
/// OFICIAL //////////////////////////////////////
////////////////////////////////////
/** @class_declaration head */
////////////////////////////////////
/// DESARROLLO //////////////////////////////////////
class head extends oficial {
  function head( context ) { oficial ( context ); }
}

```

```

//// DESARROLLO //////////////////////////////////////
////////////////////////////////////
/** @class_declaration ifaceCtx */
////////////////////////////////////
//// INTERFACE
class ifaceCtx extends head {
    function ifaceCtx( context ) { head( context ); }
}
const iface = new ifaceCtx( this );
//// INTERFACE
////////////////////////////////////
/** @class_definition interna */
////////////////////////////////////
//// DEFINICION //////////////////////////////////////
////////////////////////////////////
//// INTERNA //////////////////////////////////////
/* Función que se llama al iniciar el formulario
Conecta la señal bufferchanged (cambio en el buffer,cambio en un campo) con el slot o función
bufferChanged
Si se la llama en modo alta inhabilitará el campo 'Causa de la baja'
*/
function interna_init()
{
    var cursor:FLSqlCursor = this.cursor(); // Objeto FLSqlCursor asociado al formulario

    connect(this.cursor(), "bufferChanged(QString)", this, "iface.bufferChanged");
    if (cursor.modeAccess() == cursor.Insert)
        this.child("fdbCausaBaja").setDisabled(true);
}
/* Función que calcula el valor de un campo. En este caso el sueldo neto cuando a partir del bruto y los
impuestos
fN: Nombre del campo a calcular
Resultado: Valor del campo
*/
function interna_calculateField(fN)
{
    var cursor:FLSqlCursor = this.cursor(); // Objeto FLSqlCursor asociado al formulario
    var valor:String = "";

    switch(fN) {
        // El sueldo neto será el sueldo bruto tras descontarle los impuestos
        case "sueldoneto":
            var sueldoBruto = parseFloat(cursor.valueBuffer("sueldobruto"));
            var impuestos = parseFloat(cursor.valueBuffer("impuestos"));
            valor = sueldoBruto * (100 - impuestos) / 100;
            break;
    }
}
/* Función que se llama al pulsar el botón aceptar y que decide si los datos del formulario son válidos.
Si los datos no son válidos, los datos no se guardarán y el formulario de edición no se cerrará.
Resultado: true si los datos son válidos, false en caso contrario
*/
function validateForm()
{
    var cursor = this.cursor();
    var util:FLUtil = new FLUtil();
}

```

```

// La fecha de alta no puede superar la fecha actual
var hoy = new Date();
var fechaAlta = cursor.valueBuffer("fechaalta");
if (util.daysTo(fechaAlta, hoy) < 0) {
    MessageBox.warning(util.translate("scripts", "La fecha de alta no puede ser superior a la fecha
actual"),
    MessageBox.Ok, MessageBox.NoButton);
    return false;
}

return true;
}
//// INTERNA //////////////////////////////////////
////////////////////////////////////
/** @class_definition oficial */
////////////////////////////////////
//// OFICIAL //////////////////////////////////////
function oficial_bufferChanged(fN:String)
{
    switch (fN) {
        case "sueldobruto":
        case "impuestos":
            this.child("fdbSueldoNeto").setValue(this.iface.calculateField("sueldoneto"));
            break;
    }
}
//// OFICIAL //////////////////////////////////////
////////////////////////////////////
/** @class_definition head */
////////////////////////////////////
//// DESARROLLO //////////////////////////////////////
//// DESARROLLO //////////////////////////////////////
////////////////////////////////////

```

Una vez creado el fichero podemos probar el script ejecutando el formulario y comprobar que cada una de estas cuatro funciones funciona correctamente.

Vamos a mejorar un poco más nuestro script. Por un lado, vamos a habilitar nuestro control Causa de la baja cuando el usuario active el check De baja. Para ello ampliaremos la función bufferChanged para que realice esto. El código será:

```

...
    case "debaja" : // Si se activa el control, el campo Causa de la baja se habilitará

        form.child("fdbCausaBaja").setEnabled(false);

        form.child("fdbCausaBaja").setValue("");
        form.child("fdbCausaBaja").setEnabled(true);

        break;
...

```

Ya tenemos plenamente operativo nuestro primer script. Vamos ahora con el script asociado al formulario maestro, *masterempleados.qs*. Este script conectará el botón Imprimir del formulario con la función que lanzará el informe de empleados que realizaremos en el siguiente apartado. Su código es el siguiente:

```

/*****
/*****
/** @file */
/** @class_declaration interna */
////////////////////////////////////
//// DECLARACION //////////////////////////////////////

```

```

////////////////////////////////////
////////////////////////////////////
/// INTERNA //////////////////////////////////
class interna {
    var ctx:Object;
    function interna( context ) { this.ctx = context; }
    function init() { this.ctx.interna_init(); }
}
/// INTERNA //////////////////////////////////
////////////////////////////////////
/** @class_declaration oficial */
////////////////////////////////////
/// OFICIAL //////////////////////////////////
class oficial extends interna {
    var btnImprimir:Object;
    function imprimir() {
        return this.ctx.oficial_imprimir();
    }
}
/// OFICIAL //////////////////////////////////
////////////////////////////////////
/** @class_declaration head */
////////////////////////////////////
/// DESARROLLO //////////////////////////////////
class head extends oficial {
}
/// DESARROLLO //////////////////////////////////
////////////////////////////////////
/** @class_declaration ifaceCtx */
////////////////////////////////////
/// INTERFACE
class ifaceCtx extends head {
}
const iface = new ifaceCtx( this );
/// INTERFACE
////////////////////////////////////
/** @class_definition interna */
////////////////////////////////////
/// DEFINICION //////////////////////////////////
////////////////////////////////////
////////////////////////////////////
/// INTERNA //////////////////////////////////
function interna_init()
{
    connect(this.child("toolButtonPrint"), "clicked()", this, "iface.imprimir");
}
/// INTERNA //////////////////////////////////
////////////////////////////////////
////////////////////////////////////
/** @class_definition oficial */
////////////////////////////////////
/// OFICIAL //////////////////////////////////
function oficial_imprimir()
{
    var util = new FLUtil();
    var consulta = new FLSqlQuery("empleados"); //Objeto de consulta de empleados
    if (!consulta.exec()) {
        MessageBox.critical(util.translate("scripts", "Falló la consulta"),
            MessageBox.Ok, MessageBox.NoButton);
    }
    return;
}

```

```

}

var rptViewer = new FLReportViewer(); //Objeto visor de informes
rptViewer.setReportTemplate("empleados"); // Establece la plantilla del informe
rptViewer.setReportData(consulta); // Establece los datos del informe
rptViewer.renderReport(); // Mezcla plantilla y datos
rptViewer.exec(); // Muestra el informe
}
//// OFICIAL //////////////////////////////////////
////////////////////////////////////
/** @class_definition head */
////////////////////////////////////
//// DESARROLLO //////////////////////////////////////
//// DESARROLLO //////////////////////////////////////
////////////////////////////////////

```

Si tratamos de ejecutar la función imprimir la aplicación fallará, dado que la consulta empleados.qry todavía no está creada.

Creando los informes

Para finalizar, vamos a crear un pequeño informe en el que recogeremos los datos de nuestro empleados organizados por departamentos. Para ello crearemos primero una consulta que extraiga dichos datos.

Crearemos el fichero *empleados.qry* con el siguiente contenido:

```

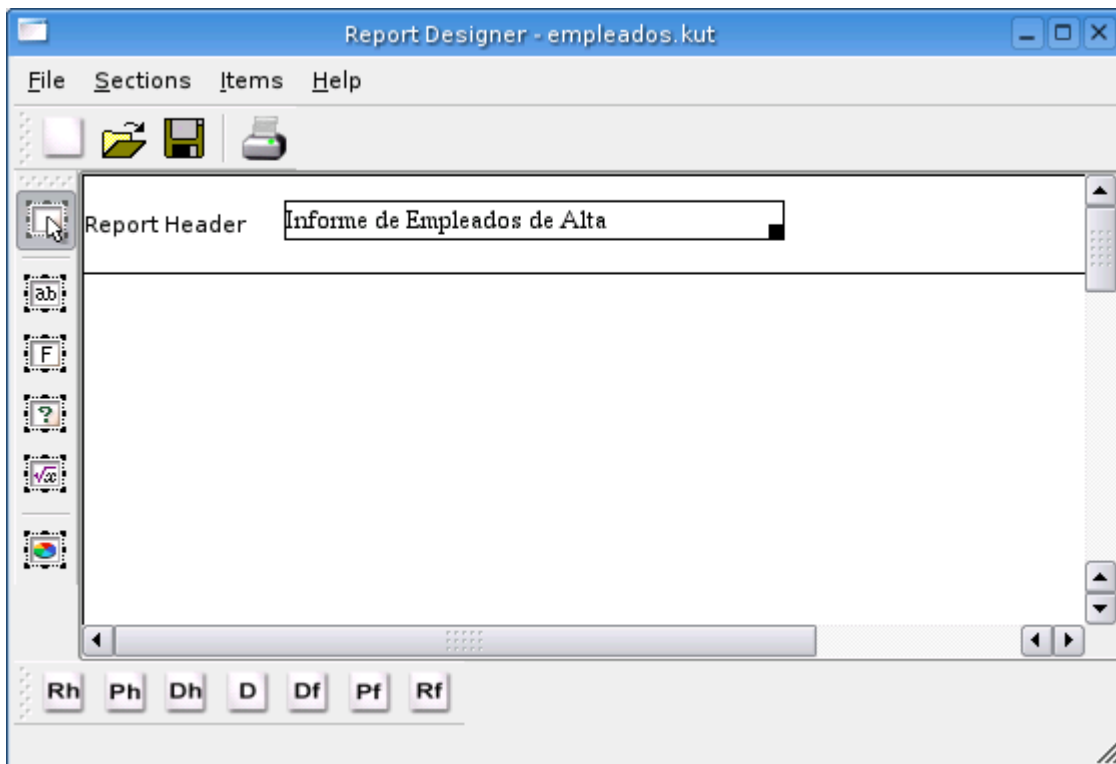
<!DOCTYPE QRY>
<QRY>
  <name>empleados</name>
  <tables>empleados,departamentos</tables>
  <group>
    <level>0</level>
    <field>departamentos.coddepartamento</field>
  </group>
  <select>
    departamentos.coddepartamento, departamentos.nombre,
    empleados.codempleado, empleados.nombrecompleto,
    empleados.sueldobruto, empleados.fechaalta
  </select>
  <from>
    departamentos INNER JOIN empleados
    ON departamentos.coddepartamento = empleados.coddepartamento
  </from>
  <where>
    empleados.debaja = false
  </where>
</QRY>

```

Una vez creada la consulta, el botón imprimir nos debe mostrar el visor de informes vacío. Esto es así porque todavía nos falta por crear el último elemento: La plantilla del informe. En la plantilla especificaremos la posición y formato de los campos, así como algunos campos especiales y de resumen.

Crearemos el informe *empleados.kut*. El editor que Abanq nos propone para este tipo de archivos es KuDesigner. Estableceremos el nombre de la plantilla como empleados y seleccionaremos el tamaño, orientación y márgenes del papel. Pusaremos Ok para acceder a la ventana de edición del informe.

Lo primero que haremos es crear el encabezado del informe. Para ello seleccionaremos *Sections -> Report Header*. Nos aparecerá una primera zona en el informe destinada al encabezado. Añadiremos una etiqueta para mostrar el título del informe. Seleccionaremos el icono Label de la barra de herramientas de la izquierda y pulsaremos sobre el punto del encabezado en el que deseemos ubicar la etiqueta. Pulsando con el botón derecho del ratón sobre la etiqueta creada editaremos sus propiedades.



14. Encabezado del informe de empleados

Por ahora nos preocuparemos únicamente de la estructura del informe, dejando la apariencia para más tarde. Las únicas propiedades que hemos cambiado son: Text = Informe de Empleados de Alta y Width = 250.

De la misma forma que hemos creado el encabezado de informe crearemos el detalle de nivel 0, que contendrá los datos de los departamentos. Para ello crearemos una sección Detail de nivel (level) 0. Una vez creada añadiremos dos campos Label y dos campos Field. La propiedad Field de los campos Field será departamentos.coddepartamento para el primero y departamentos.nombre para el segundo (Figura 15).

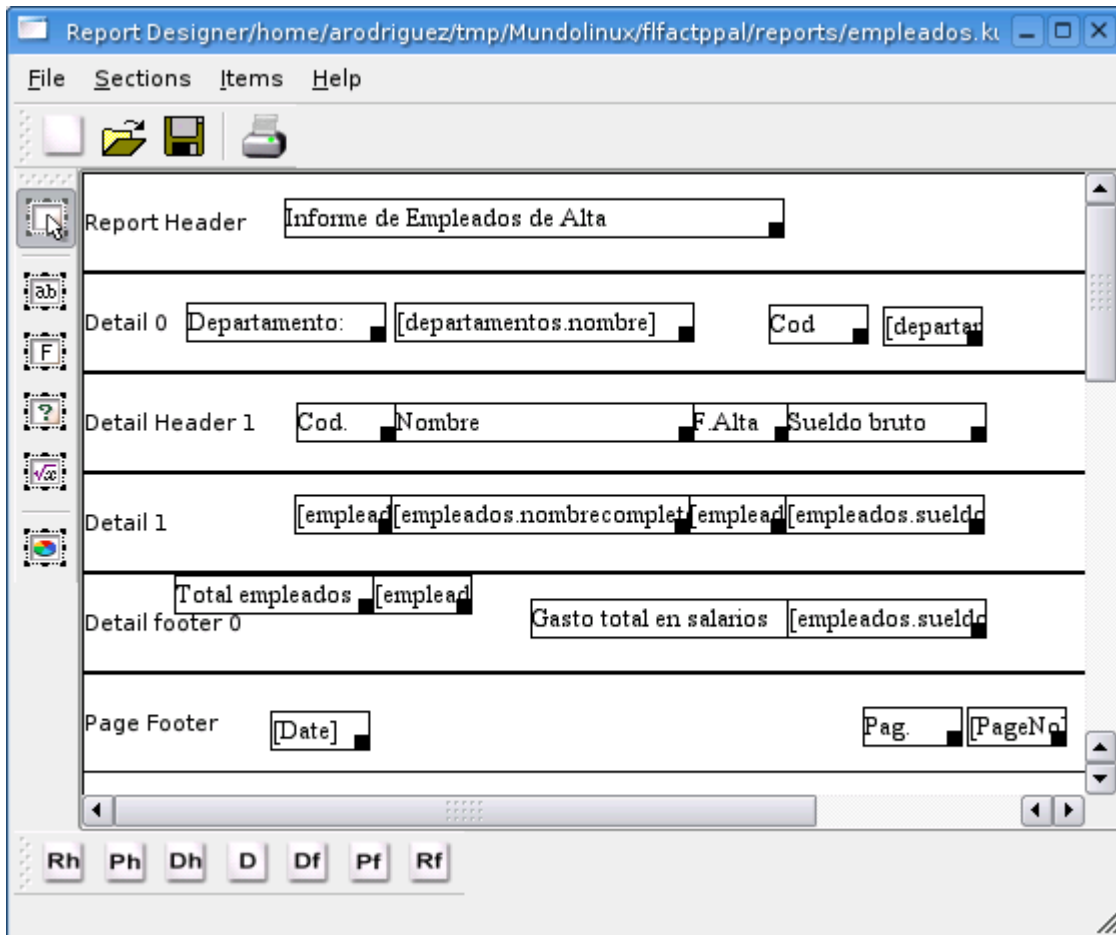
Crearemos ahora el encabezado de nivel 1 (Detail Header). En esta sección incluiremos una serie de campos Label concatenados, que actuará como cabecera para la lista de empleados de cada departamento.

La siguiente sección es el detalle de nivel 1 (Detail), con la información de los empleados (campos Field) en el mismo orden que en el encabezado de nivel 1. Modificaremos la propiedad DataType para los campos empleados.fechaalta y empleados.sueldobruto a los valores 3 (fecha) y 4 (moneda) respectivamente.

Introduciremos un pie de detalle de nivel 0 (Detail Footer) que mostrará un total de los empleados y del gasto en salarios de la empresa. Para ello introduciremos dos campos calculados, el primero con Field = empleados.codempleado y CalculationType = 0 (cuenta) y el segundo con Field = empleados.salariobruto y CalculationType = 1 (suma).

Por último cerraremos el informe con un pie de página (Page Footer) en el que mostraremos la fecha de generación del informe y el número de página. Cada uno de estos datos se consigue insertando un campo de tipo Special, la fecha con Type = 0 y el número de página con Type = 1.

El informe debe presentar un aspecto similar al de la figura 15.



15. Plantilla del informe de empleados

Una vez guardado ya podemos probar el informe. El visor de informes debe mostrarnos algo parecido a la figura 16.

Visor de informes

Menú

Informe de Empleados de Alta

Departamento: Compras Cod 000001

Cod.	Nombre	F.Alta	Sueldo bruto
000001	Francisco Javier Peláez Jim	02.08.200	2.000,00
000002	Luis Miguel Parra Martínez	17.07.200	1.800,00

Departamento: Ventas Cod 000002

Cod.	Nombre	F.Alta	Sueldo bruto
000003	Juan Pelotas Campanas	01.07.200	2.200,00

Total empleados 3

Gasto total en salarios 6.000,00

16. Informe de empleados

Bien, nuestro informe funciona, pero todavía no estamos en condiciones de ir enseñándolo por ahí. Ahora debemos darle una buena presentación. Como este trabajo es cuestión de gustos, dejaremos que cada uno dé al informe la apariencia que prefiera.

Podemos obtener más información sobre las distintas propiedades de las secciones y campos de los informes en el apartado Documentación de <http://www.Abanq.org>

Como ejemplo de presentación podemos ver el informe que muestra la figura 17.

Menú

Informe de Empleados de Alta

Departamento: 000001 - Compras

Cod.	Nombre	F.Alta	Sueldo bruto
000001	Francisco Javier Peláez Jimeno	02-08-2005	2.000,00
000002	Luis Miguel Parra Martínez	17-07-2005	1.800,00

Departamento: 000002 - Ventas

Cod.	Nombre	F.Alta	Sueldo bruto
000003	Juan Pelotas Campanas	01-07-2005	2.200,00

Total empleados: 3 Gasto total en salarios: 6.000,00

17. Ejemplo de formato de informe

Conclusión

Hemos explotado la flexibilidad de Abanq para crear una funcionalidad que necesitábamos y que no existía en los módulos actuales. Con un poco de práctica seremos capaces de realizar este tipo de ampliaciones / modificaciones en muy poco tiempo y, como hemos comprobado, con resultados profesionales.

Éste es en esencia el objetivo de Abanq: proporcionar un solución informática que se adapte a la forma de trabajar de la empresa, y no al revés como por experiencia sabemos que tantas veces sucede.

En el próximo artículo iremos un poco más lejos, creando nuestro propio módulo y analizando cómo funciona el motor de Abanq.

Actualizado el 16/02/2007