MUNICIPALIDAD DE ROSARIO

Sistema Integrado de Administración Tributaria
SIAT

Arquitectura SIAT
Anexo II - Estándares

Noviembre 2007

Contenido

Características del documento

Versiones

Fecha	Version	Descripcion	Autor
08/11/07	1.1	Versión inicial	tecso:

Objetivos del documento

El objetivo del documento es describir los estándares que fueron definidos para la construcción del proyecto SIAT

Suponiendo conocida la arquitectura "demoda", enunciaremos las definiciones generales y después las correspondientes a cada una de las tres capas que la componen.

Generales

En esta sección se describen los estándares que aplica a todo el proyecto SIAT.

Paquetes

Se realiza un paquete de java por cada módulo de siat.
Se toma el nombre corto de los módulos.
De este modo existirán los siguientes paquetes en SIAT:

```
package ar.gov.rosario.siat.bal.*;
package ar.gov.rosario.siat.def.*;
package ar.gov.rosario.siat.ef.*;
package ar.gov.rosario.siat.for.*;
package ar.gov.rosario.siat.gde.*;
package ar.gov.rosario.siat.not.*;
package ar.gov.rosario.siat.tri.*;
```

Clases Java

Se definió para la construcción de cualquier clase Java, ya sea un Bean, Dao, Servicio, Interfaz, Value Object, Controlador de la vista, etc., lo siguiente:

Orden de las secciones se será,

- Declaración de las variables de clase, públicas y privadas.
- Constructor/es
- Getters y Setters
- Métodos de clase
- Métodos que resuelven lógica de negocio, vista, etc.

Comentarios

En caso de que el nombre de una variable no sea claramente representativo se debe acompañar del comentario de tipo // en la misma linea de la declaración.
Siempre se colocan los comentarios del tipo // una linea antes del comienzo de bloques de métodos como // Constructores, // Metodos de clase, // Getters y Setters, // Validaciones, también se hace lo mismo para secciones de variables como // Buss Flag, // View Flag, // View Constants y siempre que sea necesario.

Cuando un grupo de métodos de una clase opera sobre otra clase, por ejemplo el caso de la implementación de un servicio donde un grupo de métodos opera sobre un bean, se utiliza el siguiente comentario // ---> ABM de otra clase y se cierra con // <--- Fin ABM otra clase.
Nótese que todos los comentarios se comienzan con mayúscula y no poseen acentos, eñes ni otros caracteres no ascii.

Javadoc

Todos los métodos llevan un comentario javadoc menos los getters y setters, constructores, excepto cuando estén sobrecargados..
Se distinguen dos tipos de métodos según los comentarios a realizar:

Métodos 'Genericos': pej: getById(id), getListActivos(), ...: Para estos métodos utilizamos una descripción, genérica, ubicua. Pej: Retorna una instancia del Objeto cuyo id es el del parámetro.

Métodos 'de Negocio': Estos métodos requieren un comentario descriptivo adecuado y especializado al método en cuestión; Es obligatorio utilizar los atributos @return, @autor. El atributo @param, se podrá dejar en blanco o no especificarlo, según sea conveniente hacer referencia a los parámetros de entrada en la explicación.

Iface

En esta sección se describen los estándares que aplica a la capa Iface.

Value Objects y Clases para resolver la vista

Los nombres de los Value Object deben ser los mismos que los de los bean, pero finalizados en al cadena "VO".

```
class ar.gov.rosario.siat.def.iface.model.AtributoVO
class ar.gov.rosario.siat.def.iface.model.TipoAtributoVO
class ar.gov.rosario.siat.def.iface.model.RubroVO
```

Cuando sea necesario realizar acciones de búsqueda y ABM, se crearan los siguientes models en el mismo paquete para resolver la paginación, permisos, visualización, etc. El nombre de los mismos es dado por el nombre del bean mas "SearchPage" o "Adapter"

```
class ar.gov.rosario.siat.def.iface.model.AtributoSearchPage
class ar.gov.rosario.siat.def.iface.model.AtributoAdapter
class ar.gov.rosario.siat.def.iface.model.RubroSearchPage
class ar.gov.rosario.siat.def.iface.model.RubroAdapter
```

Para las clases que se creen en esta capa se definió que:

Las variables que correspondan a ensambles a otras clases generalmente deberán instanciarse en la declaración.

Las variables "View" de tipo String creadas para la resolver lógica de presentación solo tendrán getters, y se setearan dentro del setter de la variable original.

Interface de Servicios

Los nombres de las interfaces de servicio se definen en el paquete correspondiente al módulo con la letra "I" seguida del nombre del submódulo en singular más la cadena "Service".

```
class ar.gov.rosario.siat.def.iface.service.IGravamenService;
class ar.gov.rosario.siat.def.iface.service.IAtributoService;
class ar.gov.rosario.siat.def.iface.service.IObjetoImponibleService;
class ar.gov.rosario.siat.def.iface.service.IContribuyenteService;
```

Buss

En esta sección se describen los estándares que aplica a la capa Buss.

Beans

La ubicación y nombre de los beans se corresponden con la las tablas del modelo de datos.

Los beans se acomodan en los paquetes según la ubicación de cada tabla dentro de un módulo. Los bean NO se jerarquizan según los submódulos, es decir que los beans se alojan todos en el paquete del módulo.

Por ejemplo:
Para el submódulo Atributo:

```
class ar.gov.rosario.siat.def.buss.bean.Atributo;
class ar.gov.rosario.siat.def.buss.bean.TipoAtributo;
```

Y para el submódulo Gravamen:

```
class ar.gov.rosario.siat.def.buss.bean.Categoria;
class ar.gov.rosario.siat.def.buss.bean.Rubro;
class ar.gov.rosario.siat.def.buss.bean.SubRubro;
```

Todos están dentro del mismo paquete "def".

Los nombres de las clases de esta capa se corresponden los de las tablas de la DB, pero sin el prefijo que indica el módulo, por ejemplo para la tabla "*def_atributo*" tendremos la clase "*Atributo.java*"

Del mismo modo los nombres de las propiedades de las clases se corresponden con los nombres de los campos de la tabla.

El nombre de las propiedades de tipo List, comenzará con la cadena "list" y se continua con el nombre de clase que contendrá, por ejemplo "listTipoAtributo".

Hibernate Annotations

El mapeo Clase-Tabla se realiza mediante Hibernate Annotations, para lo cual es importante saber que en la clase BaseBO perteneciente a demoda y de la cual extienden todos los beans, se encuentran los mapeos con los campos Id, usuario, fechaultmdf y estado como podemos ver en fragmento de código:

```
@MappedSuperclass
public class BaseBO extends Common {

	@Id @GeneratedValue
	private Long id;

	@Column(name = "usuario")
	private String	usuarioUltMdf; // Usuario que ingresa o modifica datos

	@Column(name = "fechaUltMdf")
```

```
	private Date	fechaUltMdf; // Fecha de ultima modificacion en la base

	@Column(name = "estado")
	private Integer	estado; // Estado en la base

	@Transient
	private Log log = LogFactory.getLog(BaseBO.class);

	@Transient
	private Long errCode= new Long(0);
```

Las anotaciones utilizadas y la forma de hacerlo:

Antes de la declaración del nombre de la clase usamos @Entity y @Table, para esta última solo utilizamos el elemento name.

```
@Entity
@Table(name = "nombre_tabla")
public class MiClase extends BaseBO {
```

Para mapear con campos se utiliza @Column, donde solo se usa el elemento name.

```
@Column(name = "campo_1")
private String propiedad_1;

@Column(name = "campo_n")
private Long propiedad_n;
```

Para el caso del los campos relacionales, se utilizan las siguientes anotaciones según correspondan, @ManyToOne o @OneToMany.

```
@ManyToOne(optional=false, fetch=Lazy)
@JoinColumn(name="idCampoRelacional")
private BeanComponente propiedad_Componente;
```

```
@OneToMany()
@JoinColumn(name="idBean")
private List<Bean> listBean;
```

Si se desea que una propiedad no sea mapeada contra un campo, se utiliza @Transient.

```
@Transient
private Integer propiedadNoMapeada
```

Servicios

Las implementaciones de los servicios se corresponden uno a uno con las interfaces definidas en el iface, por ende hay una por cada submódulo dentro de un módulo y su nombre se designa con en nombre en singular más la cadena “ServiceHbmImpl”.

```
class ar.gov.rosario.siat.def.buss.service.GravamenServiceHbmImpl;
class ar.gov.rosario.siat.def.buss.service.AtributoServiceHbmImpl;
class ar.gov.rosario.siat.def.buss.service.ObjetoImponibleServiceHbmImpl;
```

```
class ar.gov.rosario.siat.def.buss.service.ContribuyenteServiceHbmImpl;
```

Managers

En los Managers se delegan las responsabilidades que no puede ser asumidas por los objetos de negocio y del mismo modo que los servicios, existe uno por submódulo, y su nombre esta dado por el nombre completo y singular del submódulo mas la cadena "Manager".

Por ejemplo:

```
class ar.gov.rosario.siat.def.buss.bean.GravamenManager;
class ar.gov.rosario.siat.def.buss.bean.AtributoManager;
class ar.gov.rosario.siat.def.buss.bean.ObjetoImponibleManager;
class ar.gov.rosario.siat.def.buss.bean.ContribuyenteManager;
```

View

En esta sección se describen los estándares que aplica a la capa View.

Ubicación de los archivos JSP

Los jsp se disponen en carpetas representando módulo y submódulo siat.
Por ejemplo para el modulo definición "def" tenemos las carpetas:

```
/siat/view/src/def/atributo
/siat/view/src/def/contribuyente
/siat/view/src/def/gravamen
/siat/view/src/def/objetoImponible
```

Nombres de JSP

El nombre de los archivos jsp se designa según corresponda a una página de búsqueda, una de view o edit para adapters que contengan o no encabezado y detalles.
Según lo anterior nos queda:

Nombre del bean en minúscula mas "SearchPage" para las páginas de búsqueda.
Nombre del bean en minúscula seguido de la letra "V" para designar una página de View o la letra "E" para una página de edición de datos y finalizado con la cadena "Adapter".

Para el caso de adapters que contienen uno o mas detalles, se utiliza la misma regla, pero anteponiendo la cadena "Enc" (encabezado) ante la letra "V" o "E" para las páginas de view o edit del encabezado y con la cadena "EncDet" (encabezado/detalle) para las página de edición de encabezado o detalle.

Para el bean Atributo tenemos:

```
atributoSearchPage.jsp: (Búsqueda)
atributoViewAdapter.jsp: (View) (Ver/ Eliminar/ Activar/ Desactivar/ Etc.)
atributoEditAdapter.jsp: (Edit) (Agregar/ Modificar) Sin Encabezado/Detalle
```

Y para Recurso:

recursoSearchPage.jsp: (Búsqueda)

recursoEncViewAdapter.jsp: (View para Encabezado) (Eliminar/ Activar/ Desactivar/ Etc.)
recursoEncEditAdapter.jsp: (Edit para Encabezado) (Agregar/ Modificar)

recursoEncDetEditAdapter.jsp: (Edit Encabezado o Detalle) (Acciones ABM sobre el o los detalles / Modificar Encabezado)

recursoEncDetViewAdapter.jsp

Archivos de configuración de Struts

Están organizados en carpetas por módulos dentro de la carpeta WEB-INF, y formada por los archivos struts-config-<modulo>.xml y el tiles-defs-<modulo>.xml,

La única premisa, pero muy importante dado el gran tamaño al que llegan estos archivos con el avance del proyecto, es colocar comentarios encerrando los bloques de entradas que resuelven cada caso de uso.

```
<!-- Caso de Uso -->
    <action ...>
        <forward ..../>
    </action>
<!-- Fin Caso de Uso -->
```

Actions de la vista

Los actions se encuentran en paquetes que representan módulos de SIAT.
Para un ABM simple tememos dos, cuyo nombre esta dado por "Administrar" o "Buscar" más nombre del bean en mayúscula más "DAction".
Para Atributo tenemos:

```
package ar.gov.rosario.siat.def.view.struts.AdministrarAtributoDAction
package ar.gov.rosario.siat.def.view.struts.BuscarAtributoDAction
```

Si el ABM es para objetos que poseen encabezado/detalle, se creará uno más para el encabezado, "AdministrarEnc":

```
package ar.gov.rosario.siat.def.view.struts.AdministrarEncAtributoDAction
```

Managers y Errores

Los mensajes o errores pueden ser cargados desde el negocio o desde la vista de la siguiente manera.

En el servicio:

```
// Seteo de warning
ret.addMessage(DefError.MI_WARNING );
ret.addMessage(DefError.MI_OTRO_WARNING );
```

```
// Seteo de Error
ret.addRecoverableError(DefError.TEST_ERROR);
```

En la vista:
La función populateVO() se encarga, en caso de encontrar errores al validar el formato de los datos submitidos por una página, de llenar la lista de errors para ser visualizados.

Para que sean visualizados en una pantalla determinada, en el controlador se debe incluir las siguiente sentencia:

```
saveDemodaMessages(request, atributoAdapterVO);
```

antes de:

```
return actionForward;
```

Y por ultimo en todos los archivos jsp se coloca el bloque a continuación de la línea "<html:form..."

```
<!-- Mensajes y/o Advertencias -->
<%@ include file="/base/warning.jsp" %>
<!-- Errors  -->
<html:errors bundle="base"/>
```

Con esto queda contemplada la visualización de errores y mensajes, pudiéndose configurar un estilo para cada uno a fin de diferenciarlos.

Navegación desde los SearchPage

Para la navegación que se dispara desde los SearchPage se utilizan los siguientes métodos definidos en el BaseDispatchAction:

```
forwardVerSearchPage()
forwardModificarSearchPage()
forwardEliminarSearchPage()
forwardAgregarSearchPage()
```

Estos métodos básicamente setean el act del navModel (Ver, Modificar, Eliminar, Agregar) y nos dirigen al forward pasado como parámetro. El act seteado es usado luego en el action al que se llama para decidir que servicio usar.

En el caso de los SearchPage volverán al método "buscar" del action que dispara el método; por lo que en los Buscar<Bean>DAction siempre debe estar definido el método "buscar".

Navegación desde los Adapters

Para la navegación que se dispara desde los Adapter se utilizan los siguientes métodos definidos en el BaseDispatchAction:

```
forwardVerAdapter()
forwardModificarAdapter()
forwardEliminarAdapter()
forwardAgregarAdapter()
```

El funcionamiento es similar al descripto para los SearchPage, solo que en este caso el volver va al método "refill()" del action que dispare la acción; por lo que en los Administrar<Bean>DAction siempre debe estar definido el método "refill()".
Estos métodos están pensados para ser usados en Adapters compuestos, del tipo Encabezado/Detalle.

Navegación general

Otros métodos de navegación general disponibles también en BaseDispatchAction son:

El método baseVolver() utilizado por una pagina para volver a la que la llamó:

```
baseVolver(ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response,
                String VOName)
```

El método baseForward() es el básico utilizado por todos los forward<Accion>Adapter y forward<Accion>SearchPage descriptos anteriormente.

Su función es setear los valores de navegación en el NavModel, método a donde volver (parámetro "actVolver") y el forward a donde me dirijo con su correspondiente act (parámetros "forward" y "act" respectivamente):

```
baseForward(ActionMapping mapping,
                        HttpServletRequest request,
                        String funcNameOrigen,
                        String actVolver,
                        String forward, String act)
```

Gestión de permisos

En esta sección se intenta establecer las convenciones de uso y creación de Acciones y Métodos para establecer permisos en SWE.

El manejo de los permisos involucra el código de los JSP, los VO y las entradas en SWE. Los JSP se pueden distinguir en dos grupos: JSP De Búsqueda, JSP de formularios.

JSP De Búsqueda / Administradores de Cabeceras

En estos JSP típicamente no se altera el estado de la aplicación pero si llevan a otros JSP desde donde se puede alterar.

Generalmente son los jsp que muestran listas en formas de tablas en los que se permiten:

Botón Buscar: No posee control de activación/inactivación
Botón Limpiar: No posee control de activación/inactivación
Botón Volver: No posee control de activación/inactivación
Botón Ver: Posee control manejado por bandera VerEnabled
Botón Modificar: Posee control manejado por bandera en ModificarEnabled

Botón Eliminar: Posee control manejado por bandera en EliminarEnabled
Botón Agregar: Posee control manejado por bandera en AgregarEnabled
Botón que lleva a 'Otro' formulario: Posee control manejado por bandera 'Otro'FormEnabled
Botón que lleva a SearchPage o lista: Posee control manejado por bandera en BuscarXXXEnabled()

En el correspondiente model SearchPage se implementan los respectivos get<Bandera>Enabled().

Si corresponde a botones que llevan a formularios se utiliza la acción y método que corresponde al Administrador de dicho formulario:

Por ejemplo para el SearchPage del ABM de Atributo:

```
getVerEnabled():          accion="/def/AdministrarAtributo"   method="ver"
getModificarEnabled():    accion="/def/AdministrarAtributo"
method="actualizar"
getEliminarEnabled():     accion="/def/AdministrarAtributo"
method="borrar"
getAgregarEnabled():      accion="/def/AdministrarAtributo"
method="agregar"
getOtroFormEnabled():     accion="/def/AdministrarAtributo"
method="otraCosa"
```

Si corresponde a botones que llevan a otros SearchPage o Listas, se utiliza el método "inicializar" de dicho SearchPage o Lista.

Por Ejemplo:

```
getBuscarCasoEnabled():   accion="/seg/BuscarCaso"    method="inicializar"
```

JSP de formularios Agregar / Modificar / Eliminar / Otras Acciones

Generalmente desde estos formularios es desde donde realmente se disparan las acciones que alteran el estado de la aplicación.
Típicamente estos formularios poseen botones que no tiene sentido habilitar / deshabilitar porque lo que se controla es el acceso o no a mismo:

Botón Volver: No posee control de activación/inactivación
Botón Modificar/Eliminar/Agregar: No posee control de activación/inactivación

Acción a dar de alta para los ABM en general

Las acciones a dar de alta en SWE tienen que ver mucho con la forma de diseñar y programar cada aplicación. Siguiendo las convenciones de arriba los valores a dar de alta por cada ABM serian por ejemplo:

```
acción="/seg/AdminstrarAlgo"      método="inicializar"
acción="/seg/AdminstrarAlgo"      método="ver"
acción="/seg/AdminstrarAlgo"      método="insertar"
acción="/seg/AdminstrarAlgo"      método="actualizar"
acción="/seg/AdminstrarAlgo"      método="borrar"
acción="/seg/AdminstrarAlgo"      método="otroForm"
acción="/seg/BuscarOtraCosa"      método="inicializar"
```

Archivos de Recursos

Son los archivos de recursos de struts, donde se ingresan todos los mensajes que serán mostrados en las diferentes pantallas, tales como títulos, leyendas, etiquetas, mensajes de advertencia o error, etc.

Se encuentran todos en la misma carpeta y existe uno por cada módulo, quedando de la siguiente manera:

```
siat/view/src/WEB-INF/src/resources/<modulo>.properties
```

Cada archivo está dividido en dos grandes secciones "Entidad" y "GUI".
En la primera están las entradas correspondientes a cada entidad definida en el iface para el módulo en cuestión, osea que habrá un grupo de etiquetas para cada VO, y solo para estos y no para los Adapter ni SearchPage.

En las segunda sección "GUI", estarán las etiquetas correspondientes a cada jsp y que no se pueda obtener de la primer sección.

Sección Entidad

Después del comentario que indica el comienzo de la entidad, y siguiendo la siguiente regla para la escritura de las claves [modulo].[bean].[propiedad].[modificador] tenemos:

```
# Atributo ----------------------------------------
def.atributo.label=Atributo
def.atributo.title=Datos del Atributo
def.atributo.unique=El Atributo ya Existe
def.atributo.codAtributo.label=C\u00F3digo
def.atributo.codAtributo.ref=C\u00F3digo Atributo
def.atributo.codAtributo.required=Debe ingresar un C\u00F3digo de Atributo
def.atributo.codAtributo.unique=El C\u00F3digo ya Existe
```

Los distintos modificadores son:

title: Para reutilizar cada vez que se necesite mostrar un cuadro con los datos:

```
def.atributo.title=Datos del Atributo
```

label: Para representar el nombre descriptivo de la entidad.

```
def.atributo.label=Atributo
```

También para representar las propiedades del bean, en su propio mantenedor o en casos de uso que lo afectan directamente:

```
def.atributo.codAtributo.label=C\u00F3digo
def.atributo.desAtributo.label=Descripci\u00F3n
def.atributo.valorDefecto.label=Valor por Defecto
```

ref: Cuando necesitamos hacer referencia a una propiedad del bean desde otro mantenedor, caso de uso, desde columnas en el resultado de una búsqueda, etc., el valor es generalmente abreviado y contiene en nombre del bean.

```
def.atributo.codAtributo.ref=C\u00F3digo Atributo
```

```
def.atributo.desAtributo.ref=Desc. Atributo
```

required: Para cargar los mensajes relacionados a propiedades requeridas.

```
def.atributo.codAtributo.required=Debe ingresar un C\u00F3digo de Atributo
```

formatError: Se utiliza para la validación de formatos de fecha, horas y tipos numéricos, esta regla no se puede variar, ya que la función populateVO() utilizada en el controlador para cargar los datos submitidos por una página al correspondiente model, generará claves de esta forma y las cargará a las lista de errores para ser mostradas.

```
def.atributo.fechaDesde.formatError=Formato inválido en Fecha Desde
def.atributo.fechaHasta.formatError=Formato inválido en Fecha Hasta
```

unique: Cuando tengamos que realizar validaciones de unicidad.

```
def.atributo.unique=El Atributo ya Existe
def.atributo.codAtributo.unique=El C\u00F3digo ya Existe
```

hasref: Para representar mensajes de error correspondientes a referencias (FK) se usa la regla [modulo].[beanP].[beanH].hasref donde:

[beanP] es el nombre de la clase 'Padre' con la primer letra en minúscula
[beanH] es el nombre de la clase 'Hija' con la primer letra en minúscula

Por ejemplo:

```
seg.aplicacion.usrApl.hasref=La Aplicación posee Usuario
```

Siguiendo las reglas anteriores, si se deben incorporar otras validaciones de negocio que no estén contempladas, nos quedaría:
[modulo].[bean].[propiedad.[propiedad]].cualquierCosa

Ejemplo:

```
def.atributo.codAtributo.caracteresInvalidos=El código de atributo no puede contener espacios.
```

Sección GUI

Para esta sección la regla es:

```
[modulo].[jsp].[modificador]
```

Casos Search Page:

```
def.atributoSearchPage.title=Administraci\u00F3n de Atributos
def.atributoSearchPage.legend=Permite buscar, ver, modificar y agregar Atributos y sus Dominios
```

Caso Formularios de abms(Adapter):

```
def.atributoAdapter.title=Administraci\u00F3n de Atributos
```