PuntoExe Tools

Manual de Uso – Volumen 6

Montaje Final

INTRODUCCIÓN

Este Volumen 6 del Manual de Uso de las PXTools es la transcripción del sexto de los seis videos que contienen la versión completa del Curso de Entrenamiento dictado por el Ing. Juan Marcelo Bustamante (autor de las PXTools) a un grupo de nuevos usuarios de esta herramienta, a fines de 2007.

Todas las referencias internas para la ubicación de sus contenidos temáticos o imágenes remiten al momento en que fueron abordados o mostrados en la grabación original, de modo de poder ampliar en ella cualquier asunto que pueda haber perdido claridad en la transcripción.

Los títulos de cada tema hacen referencia expresa al video y momento de la reproducción en el formato:

V Nº | HH:MM:SS (HoraHora:MinutoMinuto:SegundoSegundo)

de modo que el índice de cada volumen permite ubicar los temas en el propio documento o en el correspondiente video indistintamente.

Las imágenes se identifican con una combinación de video y momento de la reproducción en el formato:

VH:MM:SS (VideoHora:MinutoMinuto:SegundoSegundo)

para que puedan ser referenciadas en cualquiera de los volúmenes del manual.

Con el fin de minimizar el peso de estos documentos en su versión digital, todas las imágenes contenidas tienen baja resolución, por lo que se recomienda regular el nivel de Zoom del visualizador antes de proceder a su lectura.

INDICE

Diseño Gráfico - Directorios de Imágenes V 6 00:00:00

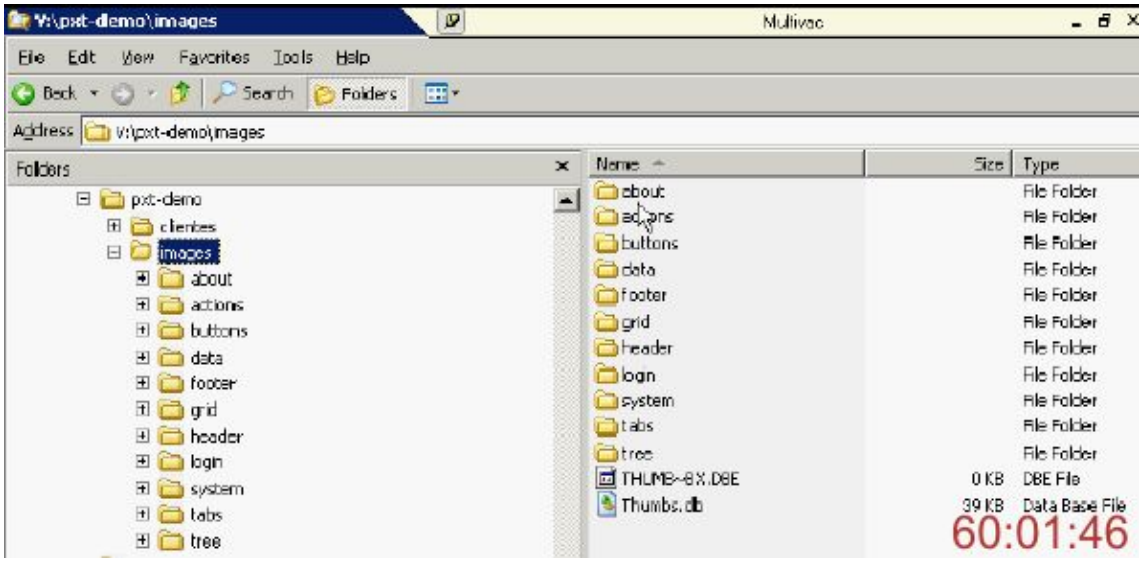
Ahora vamos a ver lo que sería la última parte de la implantación de un sistema, que es aplicar el diseño gráfico. Por lo menos de acuerdo con lo que nosotros hacemos, lo que no quiere decir que tenga que ser así.

Quizás, podría ser más conveniente conquistar a un cliente primero por los ojos y después por la funcionalidad y eso depende de cada proyecto.

Por último y salomónicamente podríamos acordar que son fases que podrían correr en paralelo. Nosotros preferimos hacerlo al final porque entonces es cuando se evalúan todos los posibles requerimientos a nivel de diseño gráfico.

Lo primero que vamos a ver es como tenemos organizada la estructura de directorios de imágenes para que si tienen que manipular esas imágenes tengan una idea de cómo funcionan.

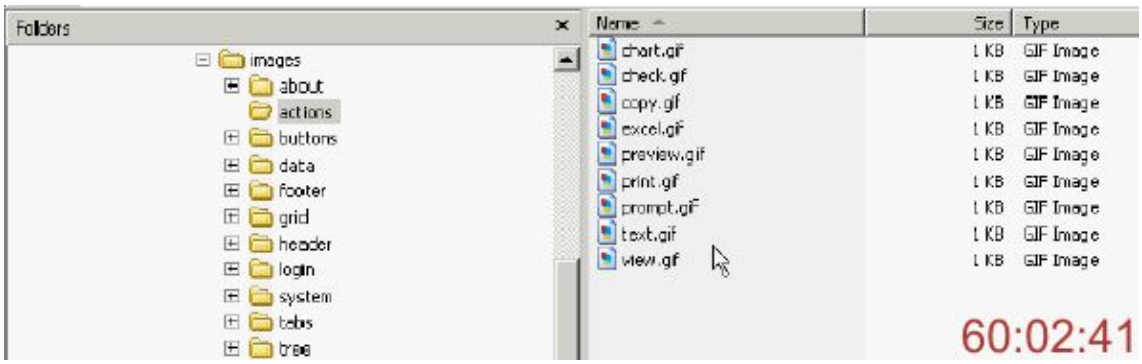
El directorio de imágenes es básicamente éste que estamos viendo:



Algunos son superabundantes, por ejemplo el “abuot” está pensado para poner la imagen del Abuot de la Empresa pero no atañe a las funcionalidades de las PXTools, que no lo usa y es parte de lo que sería la Master Page.

Actions V 6 00:02:08

Guarda las imágenes de aquellas Acciones de tipo imagen:

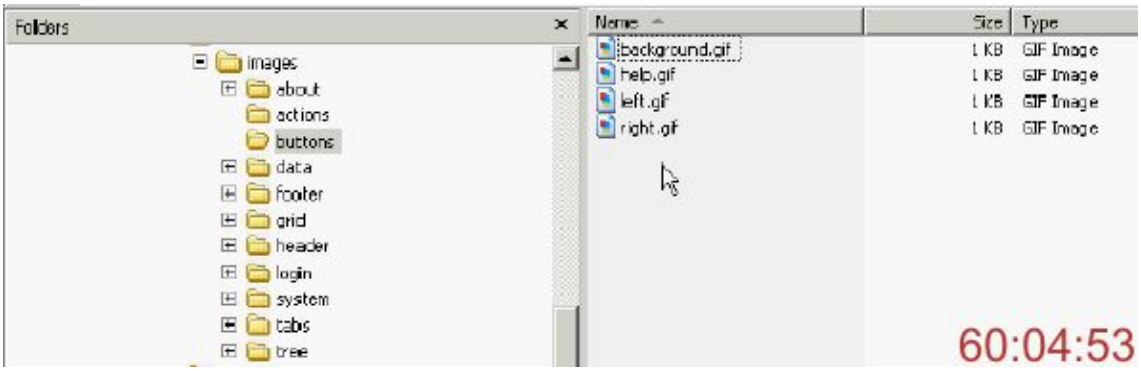


que están predefinidas por el sistema y que vimos cuando hablamos de las "DefinedImages" en la figura 20:41:22.

- Chart para la representar la Acción que permite graficar el contenido de la Grilla, que vimos al analizar la figura 11:30:09.
- Check: , que se usa en las pantallas de aprobación en las que se muestran varios registros y se debe seleccionar uno. No es un MultiRow Selection donde tendríamos un CheckBox real, sino en una pantalla donde tenemos que marcar algo o seleccionar algo mediante una Acción de tipo imagen.
- Copy está claro.
- Excel para el Export a Excel.
- Preview para pre visualizar alguna información.
- Print está claro.
- Prompt que además de estar acá, lo que se utiliza normalmente es el Prompt de GeneXus que está en el directorio raíz del Static.
  Hay dos imágenes que no están en el directorio "images" pero sí se utilizan en el sistema, que son el prompt común y el prompt de DatePicker , para seleccionar una fecha en el calendario.
- Text, para una especie de salida Export a Texto que no es una funcionalidad que venga predefinida por las PXTools pero se agregó como una imagen que podrían usar ustedes para generar algún archivo de texto.
- View que es la imagen para llamar al View del "Trabajar Con" si hubiera que hacerlo explícito. Acá lo que hace el Pattern por defecto es poner un hipervínculo en un campo de la Grilla (Description Attribute) pero en algún caso puede ser más claro poner esta "lupita" para ilustrar la Acción de ir al View.

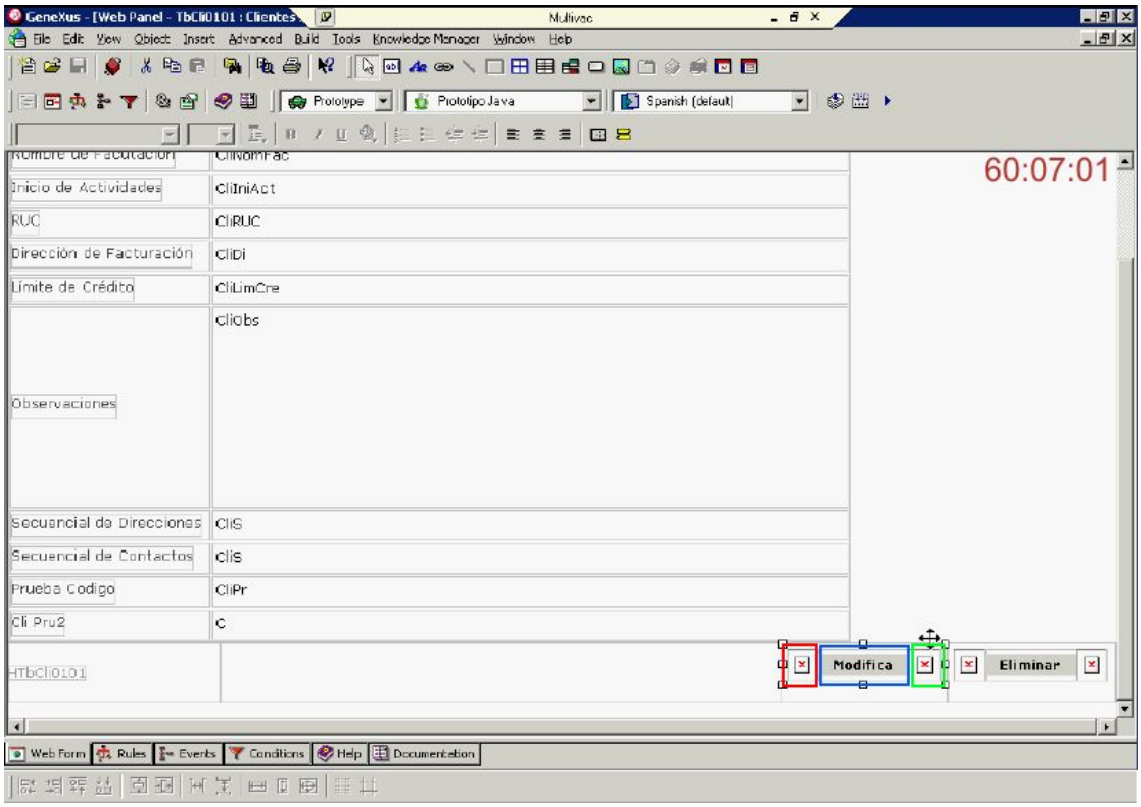
Buttons **V 6** **00:04:48**

Después tenemos los botones:



para cuando estamos definiendo Acciones de tipo Texto fuera de la Grilla en las que estamos involucrando tres tipos de imágenes

Si abrimos el Form del Web Panel del primer Tab de Clientes generado por las PXTools en el ejercicio de práctica que estamos haciendo, donde se debían programar las Acciones tipo Texto de Modificar y Eliminar el registro seleccionado:



Para representar los botones se están manejando tres elementos:

- Una imagen izquierda (recuadro rojo), que es la que da la apertura a la figura del botón;
- El botón en sí (recuadro azul), que básicamente lo único que tiene es un Background que se repite horizontalmente con el efecto mosaico;
- Una imagen derecha (recuadro verde), que es la que cierra la figura del botón.

Esto nos permite definir botones con un texto de largo variable y que la figura del botón se acomode a su tamaño, en vez de trabajar con una figura de botón con un Background de tamaño fijo y tener que acomodar el texto a ese tamaño.

De modo que siempre que trabajamos con un botón estamos trabajando con tres tipos de imágenes que son las que se muestran en la figura 60:04:53:

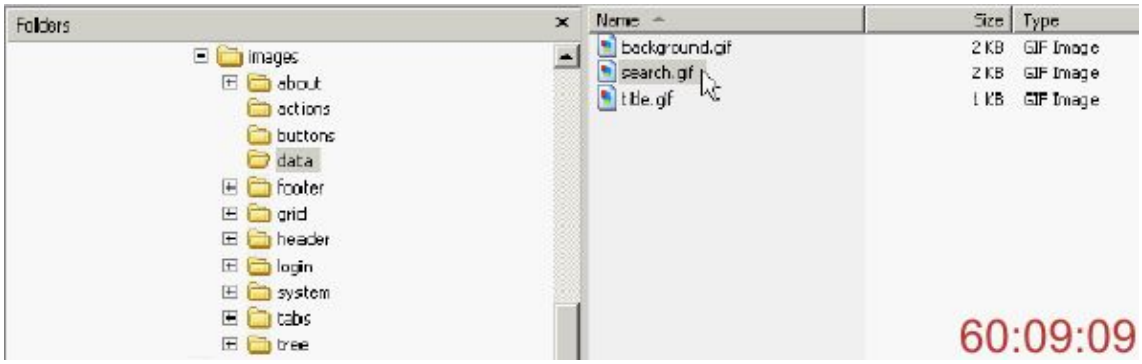
- Left:  , que es la imagen de la izquierda;
- Right:  , que es la imagen de la derecha y
- Background:  , que no es más que un segmento vertical de un pixcel. Este fondo después se extiende a todo lo largo del texto del botón por el efecto mosaico de las imágenes de Background.

Lo que les puede pasar, si se exceden al definir su altura (luego vamos a ver en el Tema hay un lugar donde se define la altura de los botones) es que este efecto mosaico les repita también verticalmente esta imagen dentro del botón.

En todo caso estas tres imágenes tienen que tener el mismo alto.

Data **V 6** **00:09:05**

Esta carpeta tiene pocas imágenes:



y la más importante es Search: , con el que hacemos normalmente la búsqueda de información refrescando la Grilla. Podría tener texto o no dependiendo del diseño, en este caso es una imagen muy pequeña.

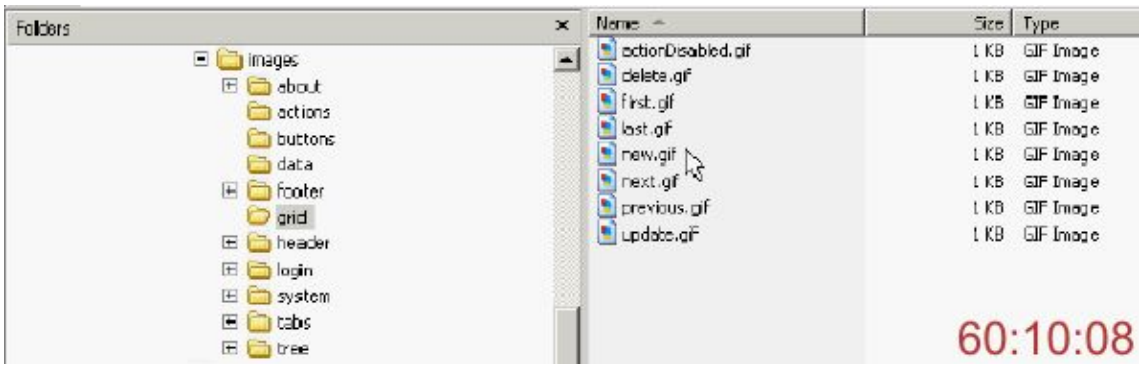
Arriba, Background es una imagen que se puso para “pintar” el Área de Datos, en combinación con la Master Page que luego vamos a abrir y empezar a manipularla para ilustrar cómo se hace, pero en todo caso esta imagen ya no es obligatoria, depende de ustedes tener una imagen con este fin.

Lo mismo con el Title: , que es un Background para el Título de la página que también, podría aplicar o no, dependiendo del diseño grafico.

Footer y Header contienen las imágenes de la impronta corporativa del sistema que se esté desarrollando y por lo tanto son ajenos a lo que el Pattern hace. También estas imágenes aplican en combinación con la Master Page.

Grid **V 6** **00:10:08**

Aquí se encuentran las imágenes de los botones de Acciones InGrid:



o sea , las Acciones que tenemos definidas dentro de la Grilla.

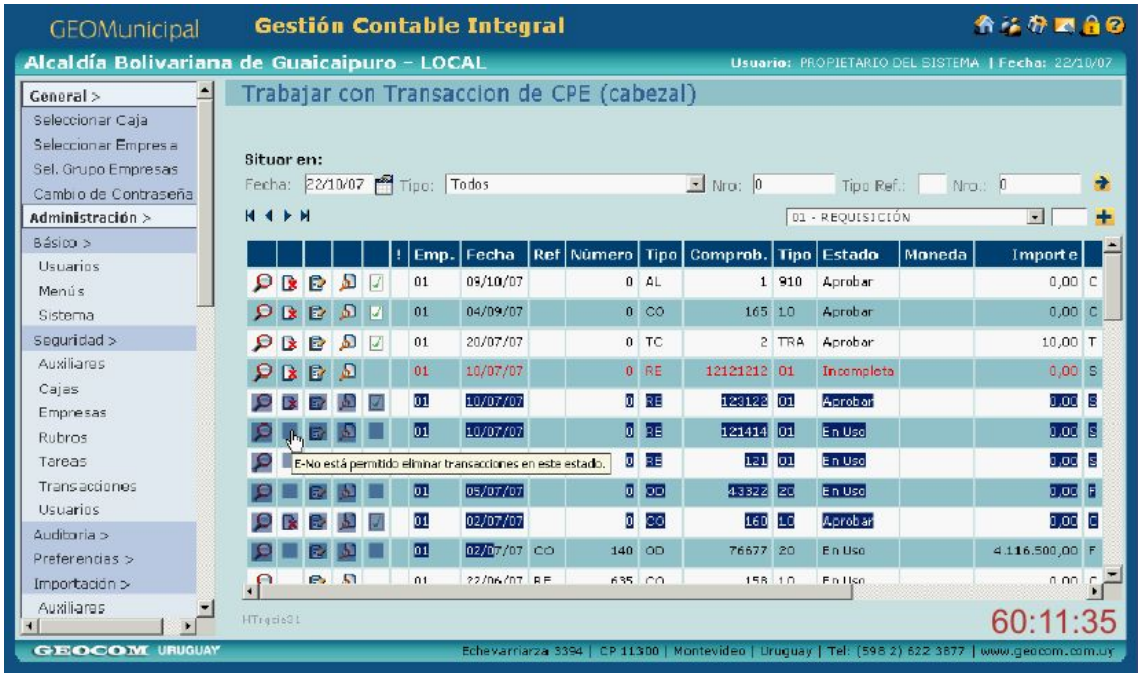
Aquí se encuentran:

- Delete, New y Update para la manipulación de la Transacción, asociados al “Trabajar Con”;
- First, Last, Next y Previous son los cuatro botoncitos de paginado de la Grilla.

Y arriba de todos ellos:

- ActionDisabled, que es una imagen totalmente transparente (es un .gif) y aplica cuando tenemos un “Trabajar Con” como el que sigue:

donde como señalo con el puntero, en la segunda columna pueden ver que la imagen de la Acción “Eliminar” no se ve, pero si marcamos el área:



podemos ver que una imagen sí está, pero ocurre es que se trata de un Gif transparente.

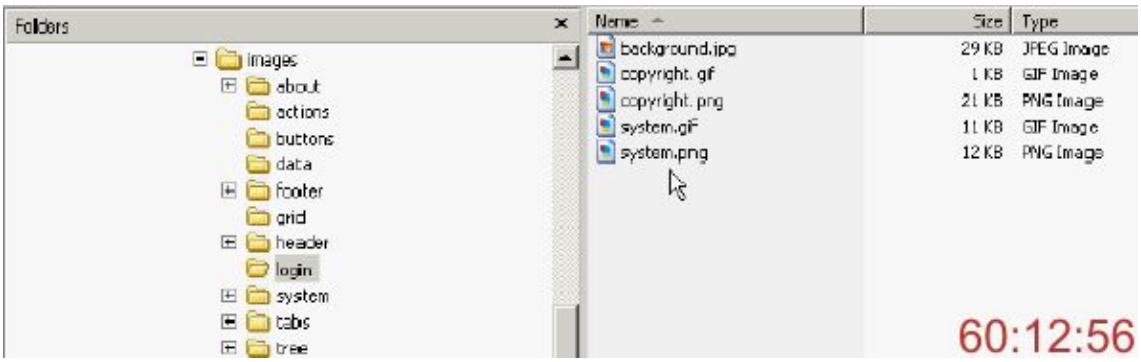
O sea, cuando la condición de la Acción InGrid impide que el usuario la aplique, como la columna tiene que mostrarse de todos modos por los otros casos en que la condición lo habilita y para no mostrar la imagen, la solución compatible con GeneXus (que en caso de invisibilizar la imagen genera problemas), es cambiar la imagen y poner la ActionDisable.gif.

Por lo tanto esta imagen transparente no deberían tocarla salvo para ajustar su tamaño al resto de las imágenes que usen en la columna donde la vayan a poner, para mantener fijo el ancho de la misma (este ancho se autoajusta al contenido).

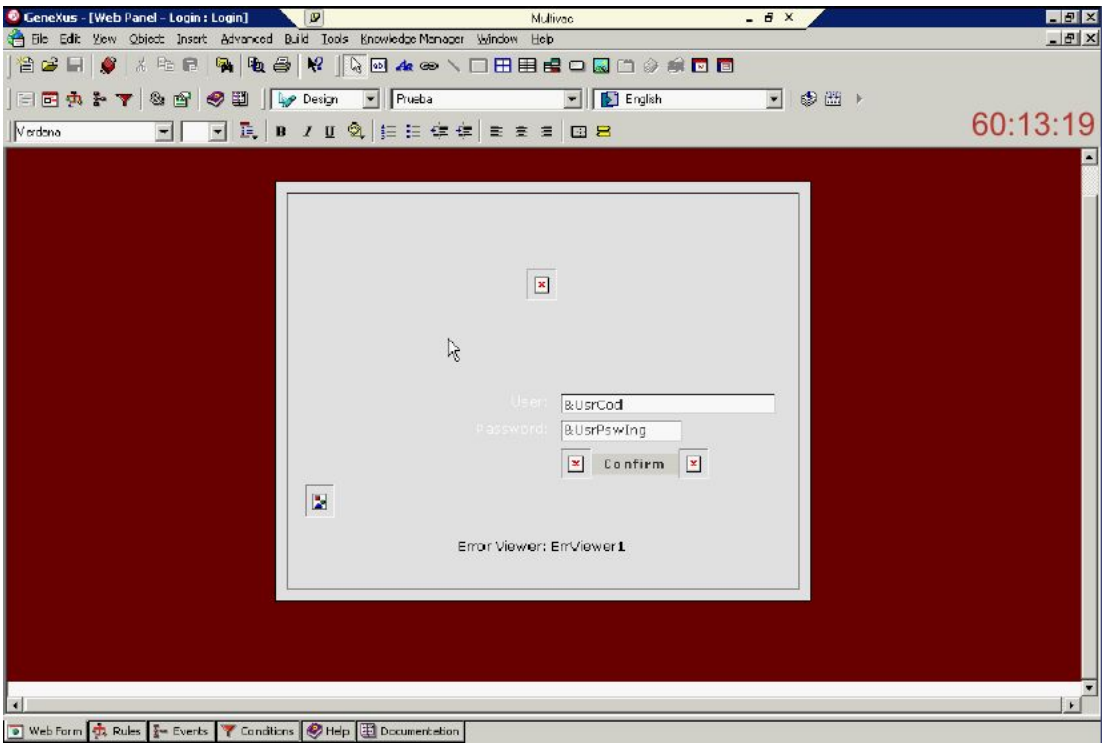
Login V 6 00:12:50

Estas imágenes no son usadas por el Pattern, simplemente el Login existe como valor por defecto y lo pueden manipular.

Esta carpeta contiene:



El Background asociado a lo que sería el diseño gráfico del Login:



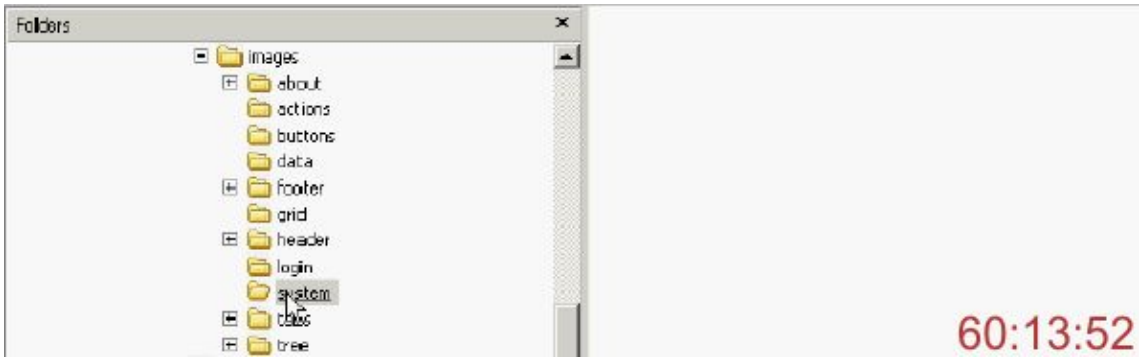
y que va adentro de este marco. En este sitio le pintamos de color bordeau la parte externa, pero no es lo que viene por defecto.

Después tenemos la imagen que representa al sistema y otra que representa al copyright como vimos en la figura 50:55:59.

Vamos a ver que podemos manejar las extensiones de estos archivos, por defecto creo que está con .gif y vamos a ver donde se puede cambiar esto.

System **V 6** **00:13:56**

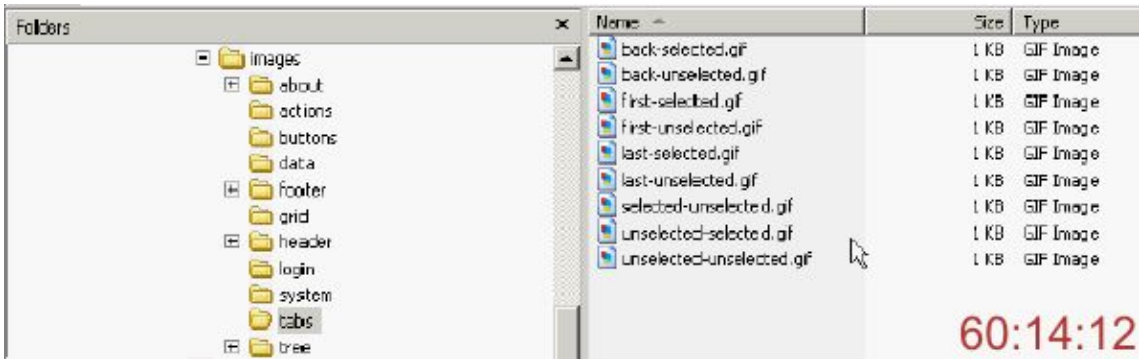
La carpeta System está, pero vacía:



reservada para que ustedes puedan almacenar las imágenes propias del sistema que van a desarrollar tal como vimos al explicar la entrada System Images Folder de los PXToolsParameters, en V 5 | 01:32:20.

Tabs **V 6** **00:14:12**

En la figura siguiente podemos ver que el juego de imágenes de Tabs:



es numeroso y para entenderlo vamos a ver un ejemplo de ventana de Tab:

en la que estamos marcando las distintas imágenes involucradas.

Ven que en este caso hay varias imágenes en juego:

- Comienzo de lengüeta de Tab seleccionado  ,
- Combinación de Tab anterior seleccionado con Tab siguiente no seleccionado  ,
- Combinación de Tab anterior no seleccionado con Tab siguiente no seleccionado  y
- Fin de lengüeta de Tab no seleccionado  .

En la figura 60:14:12, por el nombre de las imágenes vemos que está el juego completo:

- First-selected, comienzo de lengüeta de Tab seleccionado,
- First-unselected, comienzo de lengüeta de Tab no seleccionado,
- Last-selected, fin de lengüeta de Tab seleccionado,
- Last-unselected, fin de lengüeta de Tab no seleccionado,
- Selected-unselected, combinación de Tab anterior seleccionado con Tab siguiente no seleccionado,
- Unselected-selected, combinación de Tab anterior no seleccionado con Tab siguiente seleccionado,
- Unselected-unselected, combinación de Tab anterior no seleccionado con Tab siguiente no seleccionado.

Después tenemos los Backgrounds con las imágenes que van detrás del texto del nombre de la lengüeta:

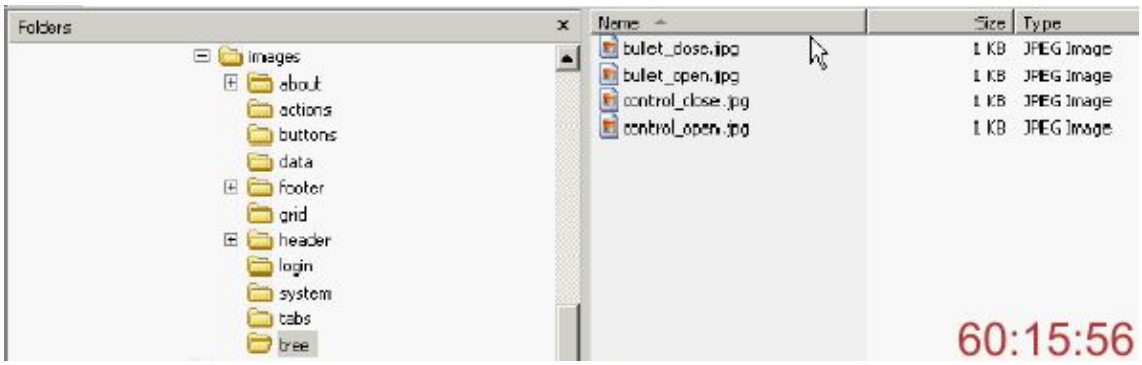
- Back-selected, para la lengüeta de Tab seleccionado y
- Back-unselected, para la lengüeta de Tab no seleccionado.

Al igual que el Background del botón, se trata de un segmento vertical de un pixcel.

Después veremos que todo esto también está parametrizado en el sistema y lo podemos manipular un poco.

Tree V 6 00:15:56

Por último la carpeta Tree para el TreeView:



que lo que tiene básicamente son las imágenes de los nodos:

- Correspondientes a nodo no seleccionado  (bullet_close) y

- A nodo seleccionado (bullet_open) y el
- Control de nodo cerrado (control_close) y
- De nodo abierto (control_open).

Es posible que más adelante agreguemos algunas imágenes más para distinguir, por ejemplo los nodos “hojas”, o los nodos que tienen “padre” de los nodos “raíz”, pero por ahora son éstas las que tenemos.

Ocurre que al TreeView lo hicimos lo más parecido al TreeView del Folders de GeneXus, donde todas son carpetitas, abiertas o cerradas, seleccionadas o no (lado izquierdo de la figura anterior).

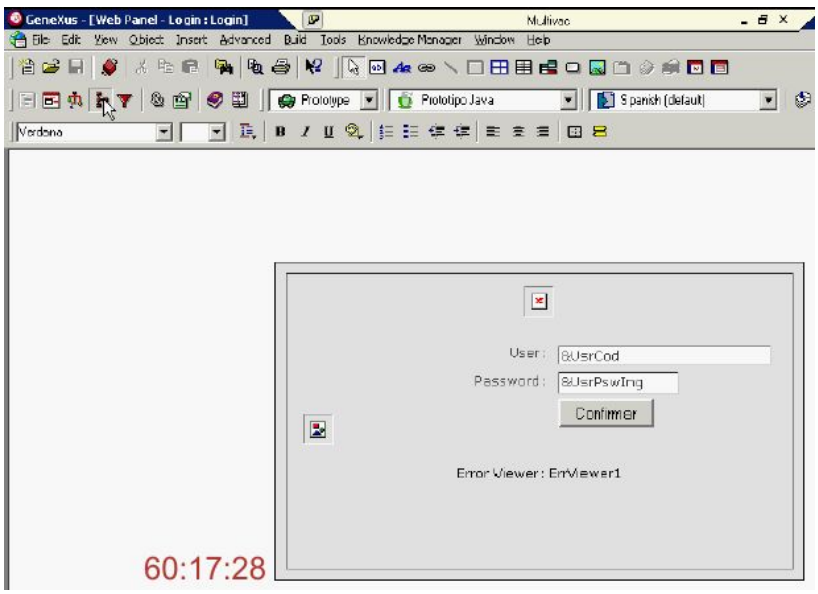
Diseño Gráfico - Login V 6 00:16:50

Ahora veremos como manipular algunos objetos, en particular el Login.

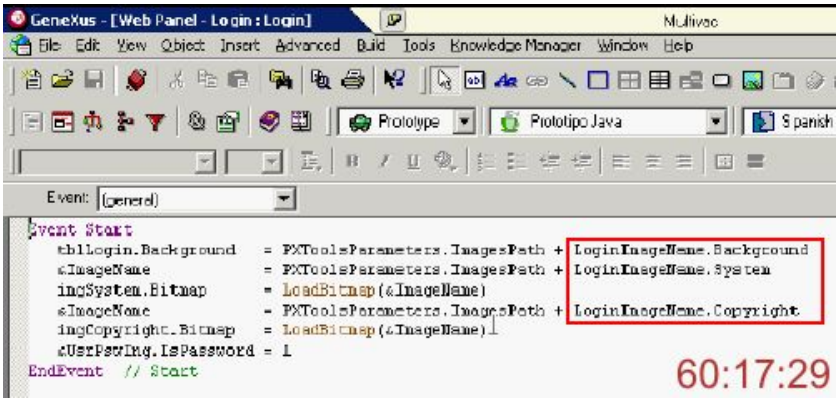
Todo lo que vamos a ver está en la Folder PXPersonalized que vimos en V 5 | 00:27:24, justamente porque se trata de funciones adaptables por ustedes a los requerimientos del sistema en el que trabajen.

Como explicamos en V 5 | 00:45:216, allí se encuentra la carpeta PXTools_System, en cuyo contenido:

está el Web Panel Login, que si lo abrimos:



y vamos a los Eventos:

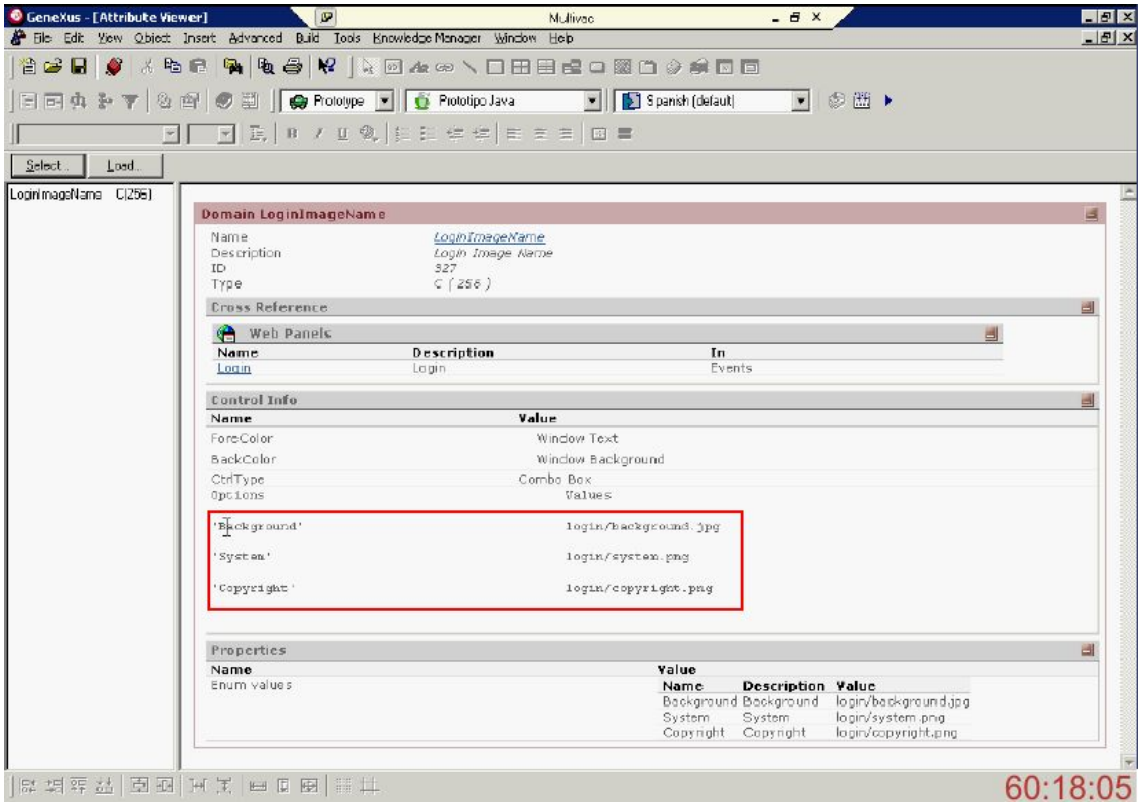


básicamente si se fijan, el Event Start del Login está referenciando a unos Dominios Enumerados LoginImagesName que es lo que les quiero mostrar cómo pueden manipular.

LoginImageNames **V 6 00:17:29**

Obviamente esto no quita que todo lo relativo al diseño gráfico lo pueden hacer todo de nuevo desde cero, entonces tomen esto sólo como un ejemplo.

Como estamos en Prototipo hacemos Menú superior Tools -> List Attributes/Domains, seleccionamos LoginImagesName y vemos el Enumerado:



que tiene tres valores: “Background”, “System” y “Copyright”, que representan las tres imágenes que juegan en este Web Panel como vimos en V 6 | 00:12:50

y que apuntan a sus correspondientes archivos en la carpeta “login” de la figura 60:12:56.

Si necesitan cambiar el tipo de imagen y su extensión, simplemente cambiando estos valores del Enumerado el sistema va a reconocer las nuevas imágenes y no tienen que modificar el Web Panel de Login.

Ahora, si quieren rediseñarlo ya es otro tema, pero la idea es tratar de que puedan comenzar el desarrollo sin complicarse con estas cosas menores.

Diseño Gráfico- otros Dominios Enumerados V 6 00:18:52

Hay otros Dominios Enumerados que fueron creados por las mismas razones que acabamos de exponer.

TabImageNames V 6 00:18:57

Las imágenes relativas a las pantallas de Tabs que acabamos de ver también tiene un Dominio Enumerado TabImageNames:

que tiene básicamente los valores que vimos en V 6 | 00:14:12 a los que se agregan las imágenes de “Next” y “Previous” que no vimos entonces.

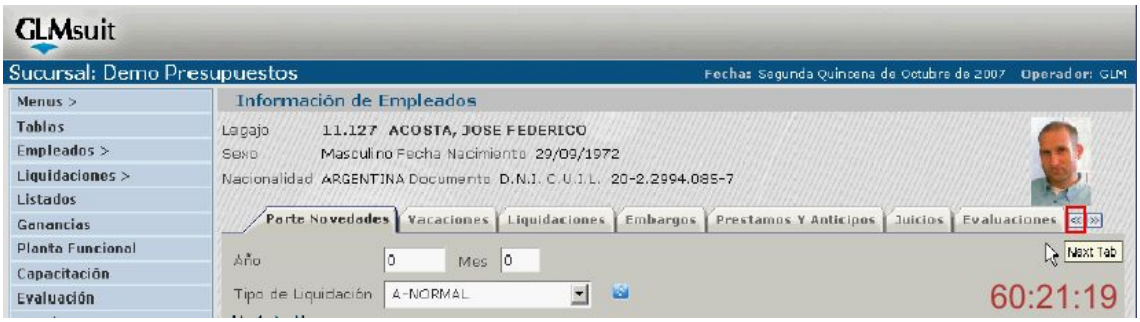
Estas imágenes son usadas en las pantallas de Tabs cuando el ancho del área no permite desplegar todas las lengüetas simultáneamente, sea porque son muchas, sea porque los textos de sus nombres son muy largos.

Para estos casos fue necesario implementar una tabulación horizontal de estas lengüetas para que el usuario pudiera acceder a las que no entraron en la pantalla, como se muestra en el siguiente ejemplo tomado de un sistema de Recursos Humanos que hemos visto anteriormente:

En este caso se están mostrando los primeros siete Tabs (esta cantidad es configurable en el sistema), seguidos del botón de “Next” (recuadro), cuya imagen “next.gif” es la que muestra la figura 60:19:11. Los Tabs que se ven debajo de estos siete, no son de la ventana de Tabs sino de la Transacción que se está mostrando en la lengüeta “Datos Generales”.

En todo caso deben tener cuidado al configurar la cantidad de Tabs que van a mostrar simultáneamente, pues que éstos quepan en el ancho depende también del largo de los textos con los nombres de las lengüetas. Por ahora el sistema no puede detectar automáticamente el exceso y va a generar un scroll horizontal local al área de Tabs.

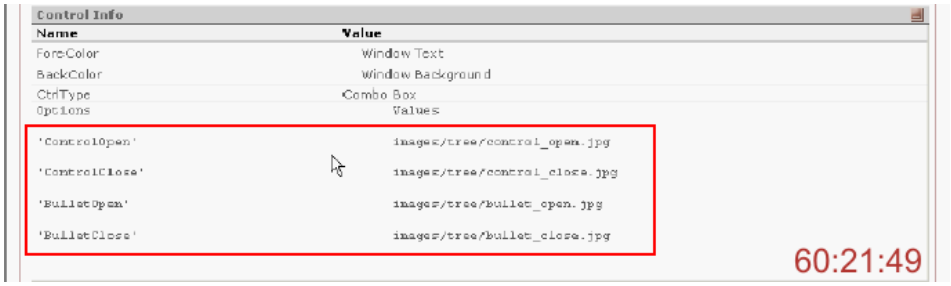
Se trata de dos botones, porque al tabular la muestra hacia la derecha presionando el botón “Next” para ver los siguientes siete Tabs:



aparece a su izquierda el botón “Previous” (recuadro) , cuya imagen “previous.gif” es la que muestra la figura 60:19:11 y hace la tabulación inversa permitiendo volver atrás.

TreeImageNames V 6 00:21:39

Para el TreeView también hay un Dominio Enumerado:

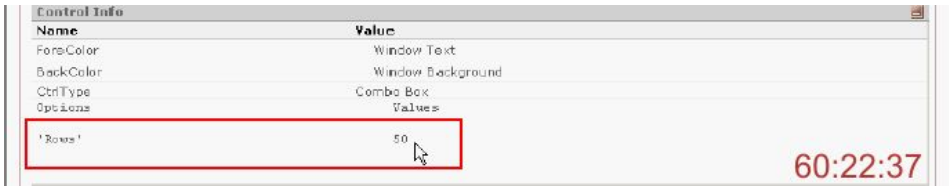


para representar las cuatro imágenes, de modo que no sea necesario meterse en el Web Panel para cambiar las referencias a estas imágenes sino que alcanza con hacerlo sólo dentro del Enumerado TreeImageNames.

Page V 6 00:22:06

El Page en esta versión todavía está separado del Dominio Enumerado PXToolsParameters, tal como venía en la versión original del Pattern WorkWith de Artech, pero creo que para la próxima lo vamos a incluir allí.

Actualmente este Dominio Enumerado:

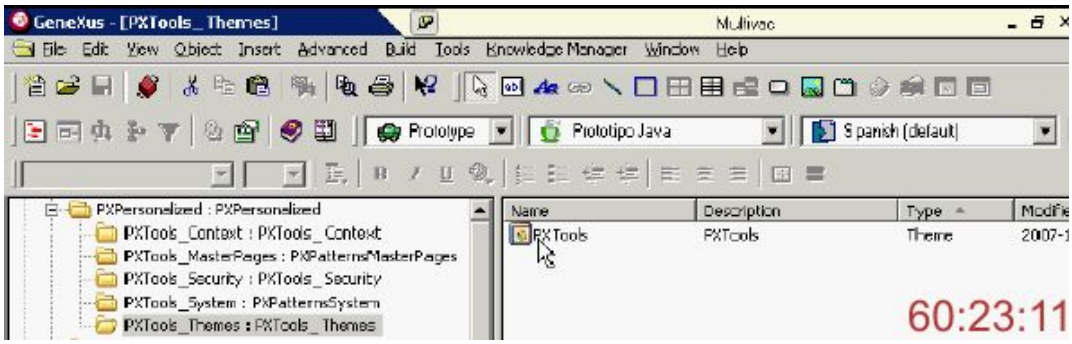


la única propiedad que tiene es el “Rows” que viene pre cargada con el valor 50 que indica el número máximo de registros que se van a mostrar en cada página de la Grilla.

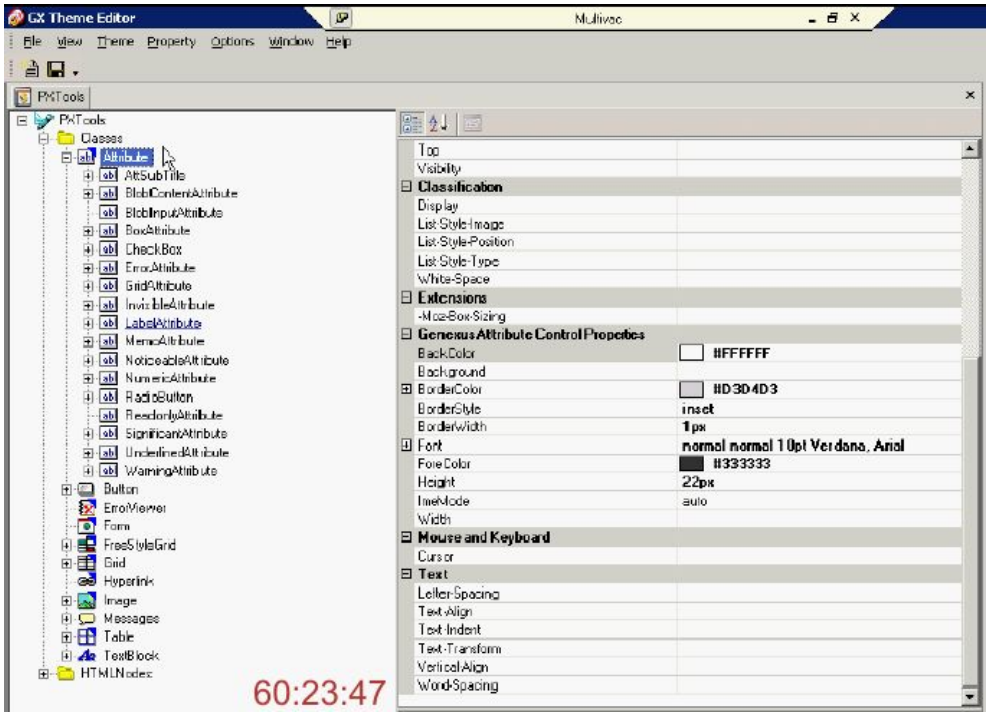
Diseño Gráfico - Tema

V 6 **00:23:00**

Comenzamos ahora el análisis del Tema GeneXus PXTools_Themes:



Hemos preferido implementar todo el diseño gráfico referido en las PXTools en un solo Tema, que si abrimos con el editor:



Clases relacionadas con el Área de Datos

V 6 **00:23:47**

Este análisis puede resultar un poco tedioso, pero es necesario hacerlo.

Para poder hacer algún cambio en el diseño gráfico que viene por defecto, se debe saber para qué sirve cada Clase.

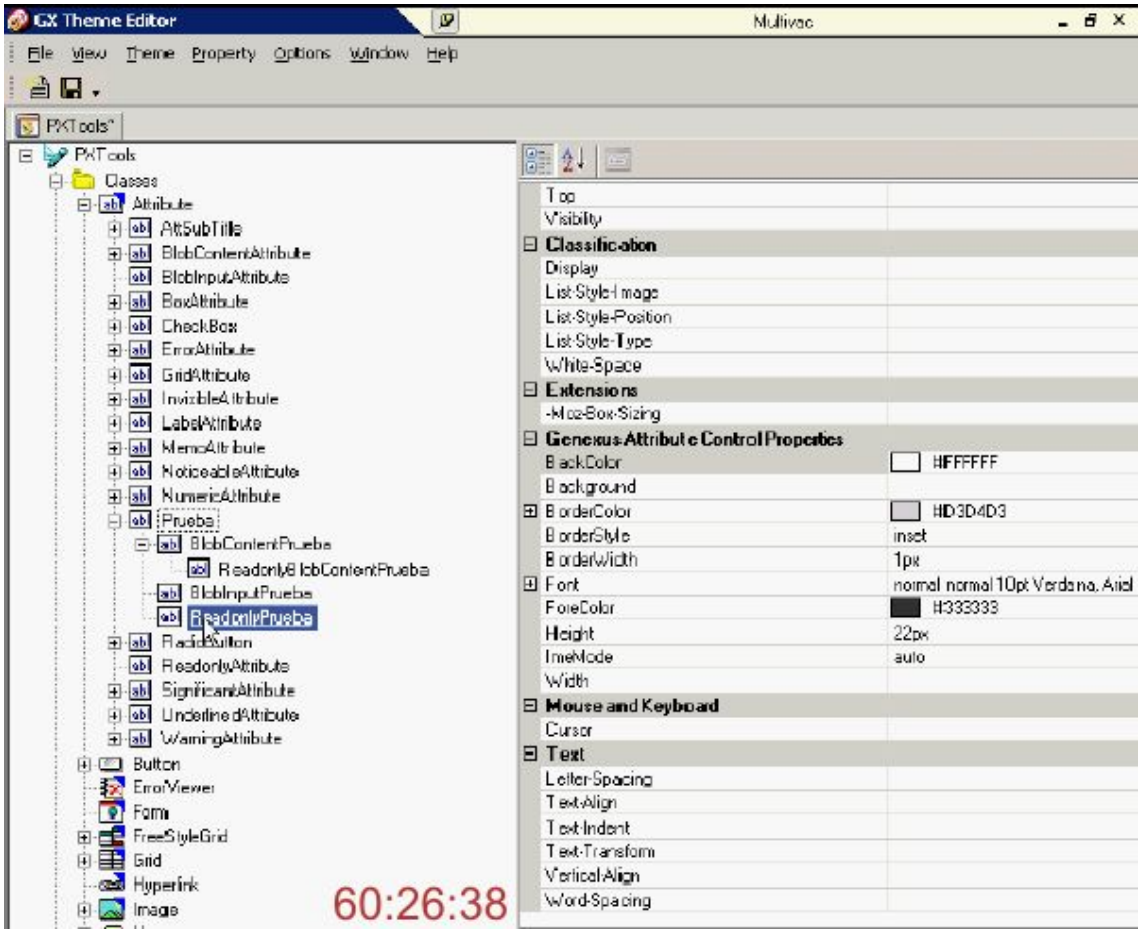
Vamos a tratar de diferenciar aquellas Clases que son requeridas por las PXTools de aquellas que están definidas por defecto para que ustedes las puedan aplicar y definir después conforme al diseño que deseen.

Comencemos con la Clase del nodo Attributes, que es la que aplicaría por defecto en todos los Atributos y Variables que definan en la pantalla. Esto es así porque esta Clase es la que está declarada por defecto, tal como lo indica la flechita azul encima del nodo:  .

La Clase por defecto puede ser cambiada haciendo botón derecho sobre otro nodo (por ejemplo GridAttribute) y “Set as GeneXus Default”. Entonces la flechita azul quedaría sobre este nuevo nodo en lugar del actual y de allí en adelante, cuando apliquemos el Pattern, la Clase por defecto que va a estar asociada será GridAttribute.

Es poco común que se cambie una Clase por defecto pero está la posibilidad. Tuvimos un caso de un cliente que quería que todas las variables o los atributos que estaban ReadOnly, tuvieran un recuadro, la caja que se muestra en Windows cuando se da esta condición y en ese caso en lugar del tradicional nodo Attribute, se pasó al BoxAttribute (cuarto nodo subordinado) como Clase por defecto.

Cuando definimos una nueva Clase (botón derecho sobre el nodo Attribute y New Class) el sistema agrega el nodo “NewClass” (para este ejemplo le cambiamos el nombre a “Prueba”) y vamos a ver que el Tema por defecto va a crear un montón de Clases asociadas, “hijas” de esa Clase:

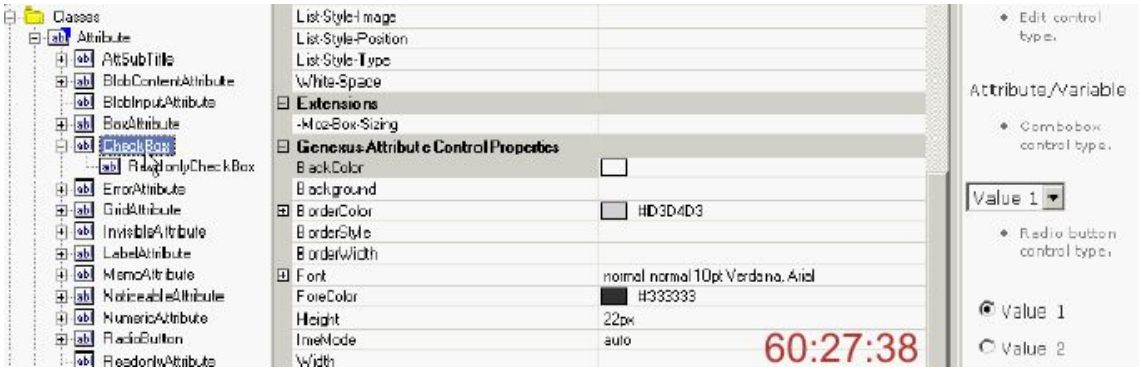


Entonces, nosotros generamos la Clase “Prueba” y GeneXus nos puso otras Clases relacionadas y la más importante es ésta que se llama “ReadonlyPrueba”. Cuando a una variable o a un atributo le pongamos la propiedad Enable, ReadOnly o NoAccept en las Reglas, GeneXus automáticamente va a utilizar esta Clase que tiene como nombre “Readonly” y el nombre de la Clase declarada en principio, en este caso “Prueba”. Está como “hija” pero podría estar en otro lado que él la va a buscar con ese nombre.

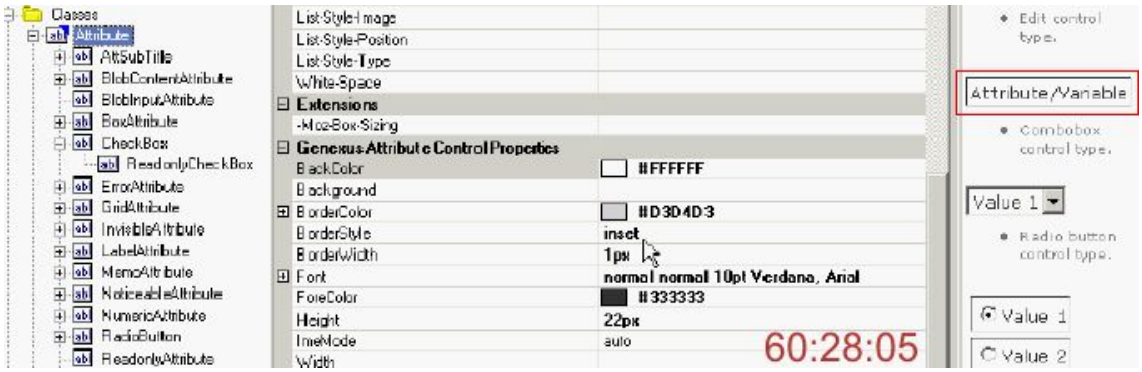
Las otras Clases “hijas” las crea, creo que a partir de la versión 9.0, porque podríamos definir un atributo de tipo Blob que tiene toda una estética visual para la cual asocia estas clases. Nosotros nunca las hemos usado, es más, por lo general estas clases las borramos para no hacer más pesado todavía el Tema.

Todo esto termina siendo información que va almacenada en un archivo css y cuanto menos información haya, mejor. Las dejamos sí en la Clase Attribute (segundo y tercer nodo subordinado) porque si en algún momento alguien define un Atributo de tipo Blob es necesario que estén las clases, pero en todas las demás las eliminamos.

Pasemos a describir las Clases más importantes debajo de este nodo Attributes, comenzando por la CheckBox:

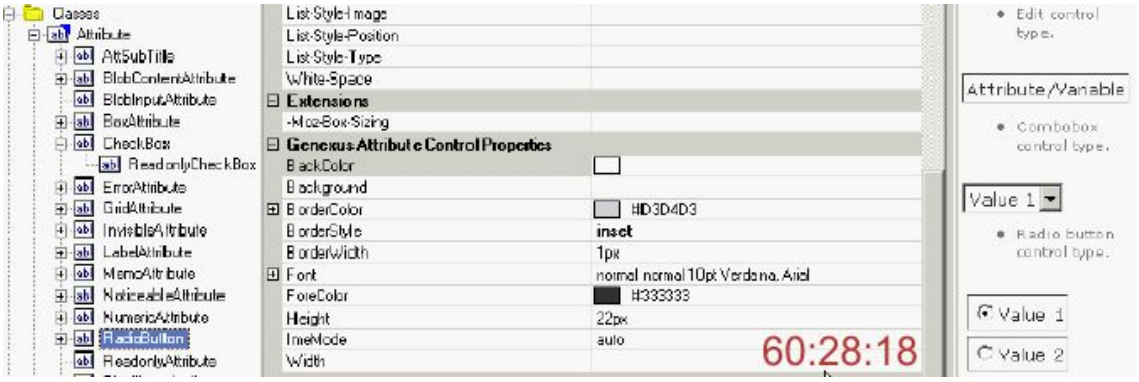


El Pattern, por defecto, siempre que tenga que mostrar una CheckBox le va a asociar esta Clase CheckBox. Esto es porque normalmente, cuando definimos una Clase de tipo Atributo común, el atributo en modo de edición:



tiene todo el tema de un recuadro, el Border, declarado en las propiedades BorderWidth y BorderStyle que es lo que le da la estética de cajita de texto (recuadro rojo) y en los CheckBox, este recuadro no queremos que se vea. Solo mostramos la cajita de CheckBox con el texto a la derecha.

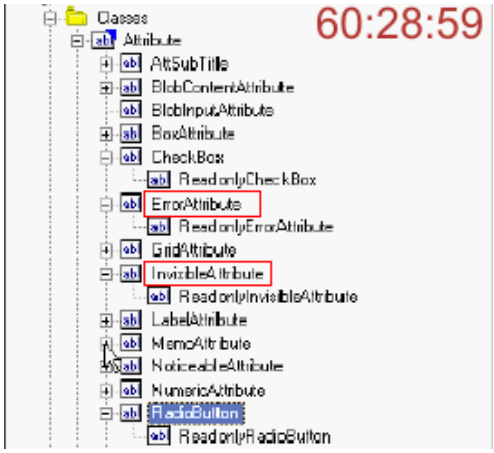
Lo mismo ocurre con la RadioButton:



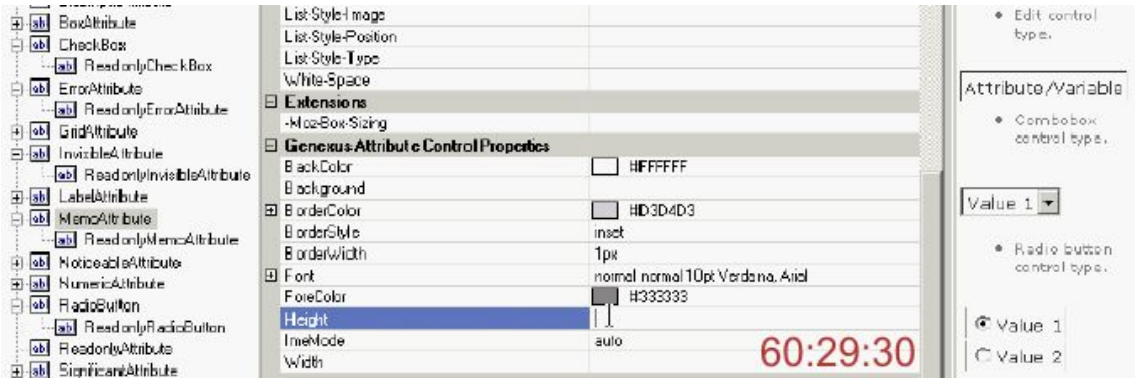
que el Pattern resuelve con esta misma lógica, que hace que cuando ponemos en la pantalla una RadioButton, automáticamente le va a asociar esta Clase RadioButton al definir el control de edición.

O sea que estas clases son las que se van a aplicar por defecto: La Attribute, CheckBox o RadioButton para la mayoría de los campos que nosotros definamos en pantalla.

Después tenemos algunas otras que son conceptos, como por ejemplo, la ErrorAttribute y la InvisibleAttribute:



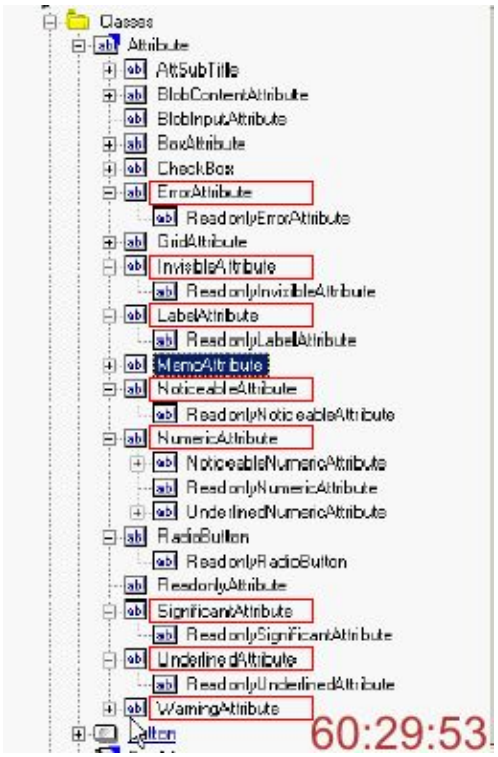
Al verla me doy cuenta que a la MemoAttribute (puntero), olvidé mencionarla entre las Clases que asocia automáticamente el Pattern. Si definimos un campo de tipo LongVarChar en el sistema, el Pattern le va a asociar esta Clase MemoAttribute:



Esto nos va a permitir estandarizar el alto de los campos Memo.

Los Memo por defecto, si los ponemos sin más en pantalla, GeneXus va a generar automáticamente un cierto tamaño de campo. Ahora con esta Clase podemos usar la propiedad Height (puntero en la figura anterior) y establecer un alto y también un ancho estándar para los campos de tipo Memo.

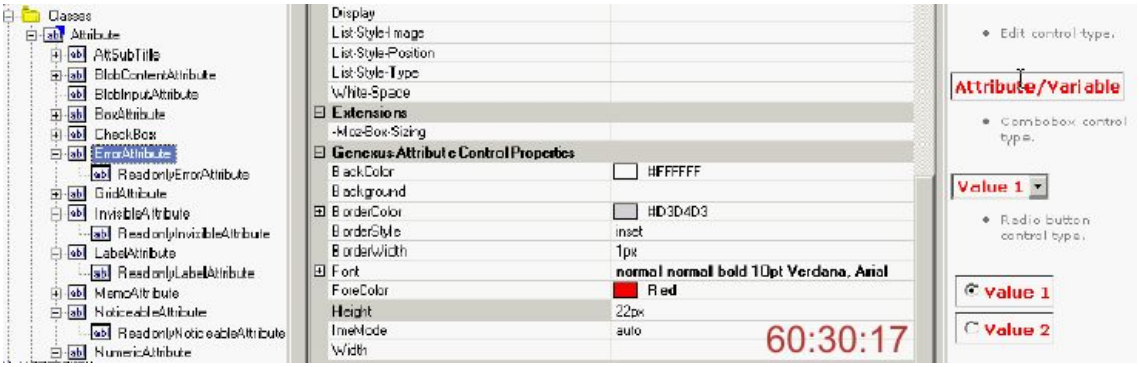
Volviendo a las clases que son conceptos, además de la ErrorAttribute y la InvisibleAttribute, tenemos la LabelAttribute, la NoticeableAttribute, la NumericAttribute, la SignificantAttribute, la UnderlinedAttribute y la WarningAttribute:



son todos conceptos que ustedes pueden manipular para adecuar su diseño a las especificaciones gráficas del sistema a desarrollar. Nosotros las tuvimos que ir haciendo para distintos proyectos y las fuimos dejando porque entendimos que podrían sernos útiles en otros casos similares.

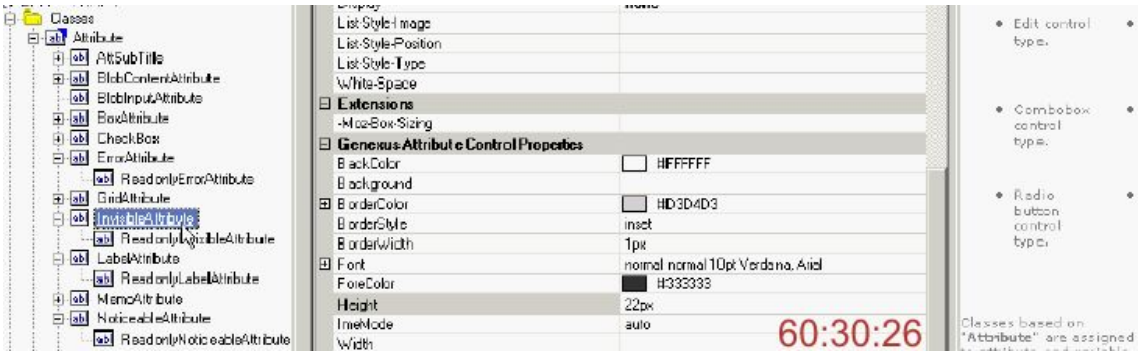
El proceso de creación de Clases es bastante tedioso y está bueno que ya estén creadas y podamos referenciarlas aunque luego debamos ajustar el diseño.

Todas representan conceptos, por ejemplo la ErrorAttribute:



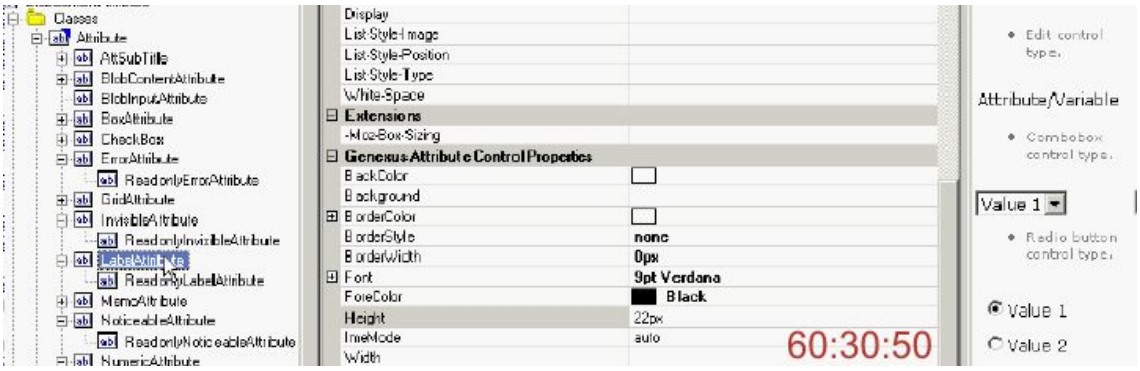
cuando se aplica, el atributo va a tener declarado el ForeColor en Rojo.

La InvisibleAttribute es una Clase que deja invisible al atributo:



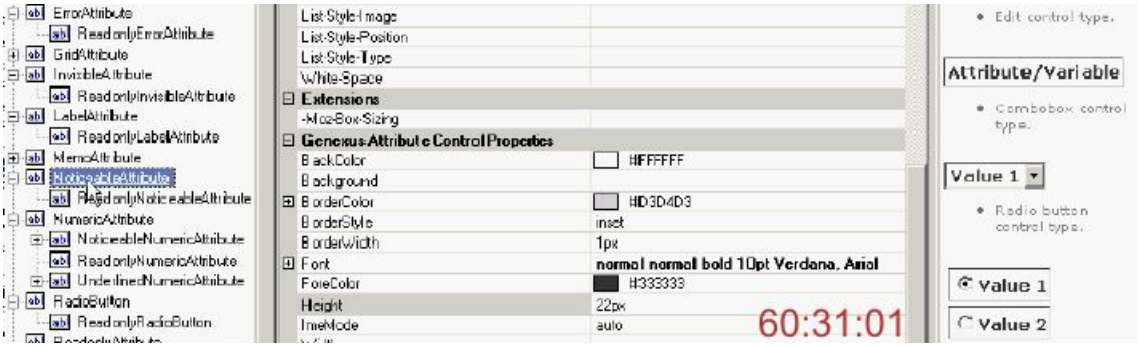
Vamos a ver que en algún caso se requiere que el atributo esté en pantalla pero esté invisible. Esto no es lo mismo que la propiedad ".Visible", tiene alguna diferencia que ahora les voy a comentar, pero para no perder el hilo de la exposición primero vamos a terminar de ver este tema.

La LabelAttribute es para definir típicamente variables ReadOnly que se conceptualizan como "Label":



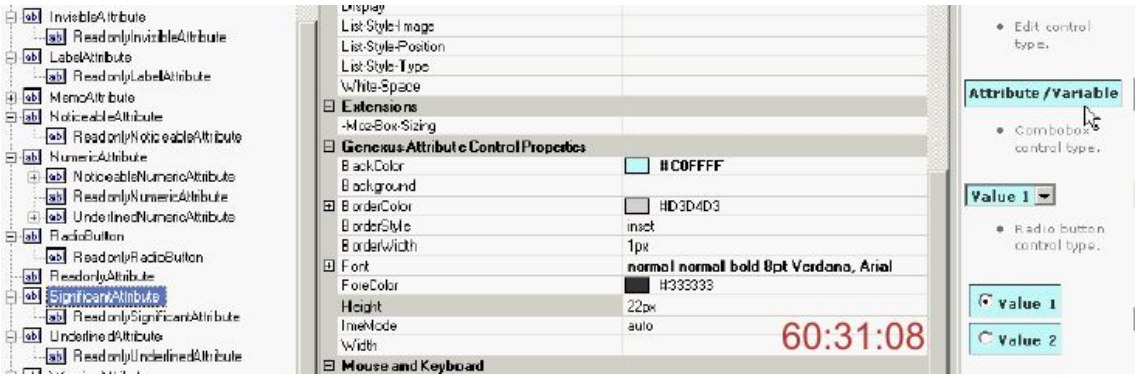
entonces se aplica la misma clase que se usa como Label de las variables.

La NoticeableAttribute es una Clase que pone al atributo en "negrita":



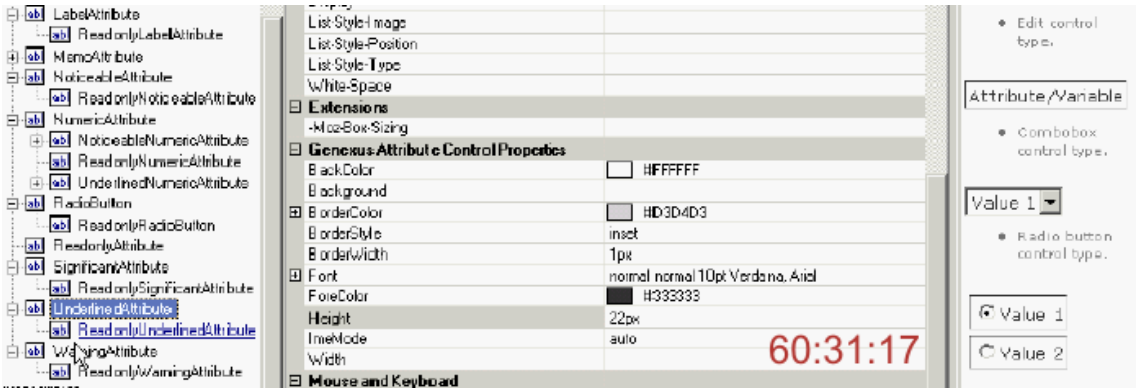
simplemente para remarcar o reforzar alguna información.

Algo similar ocurre con la Clase SignificantAttribute, con la variante que aquí juegan los colores, que ustedes tendrán que armonizar con los colores del resto de la pantalla según las especificaciones del diseño gráfico que se haya definido para la aplicación:



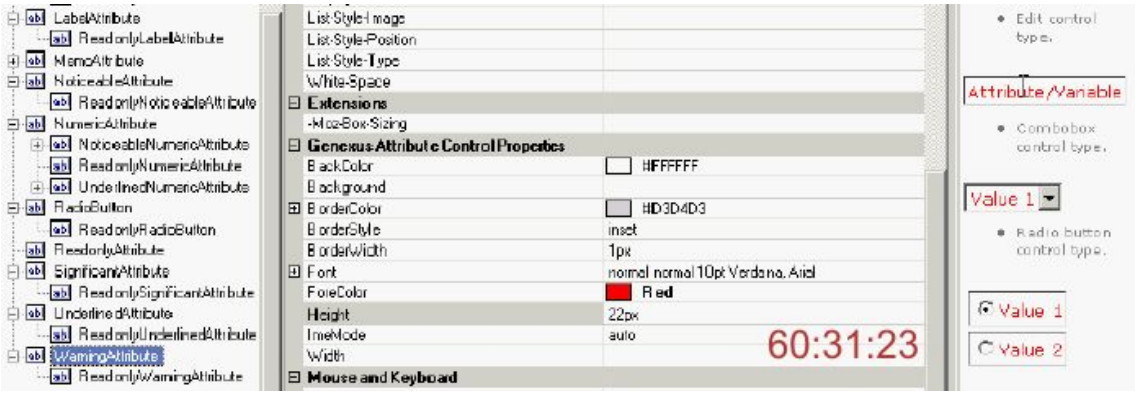
y la usamos cuando hay algo muy particular que hay que remarcar.

La UnderlinedAttribute:



para subrayar el valor del atributo.

Y la WarningAttribute que también declara el ForeColor en Rojo:



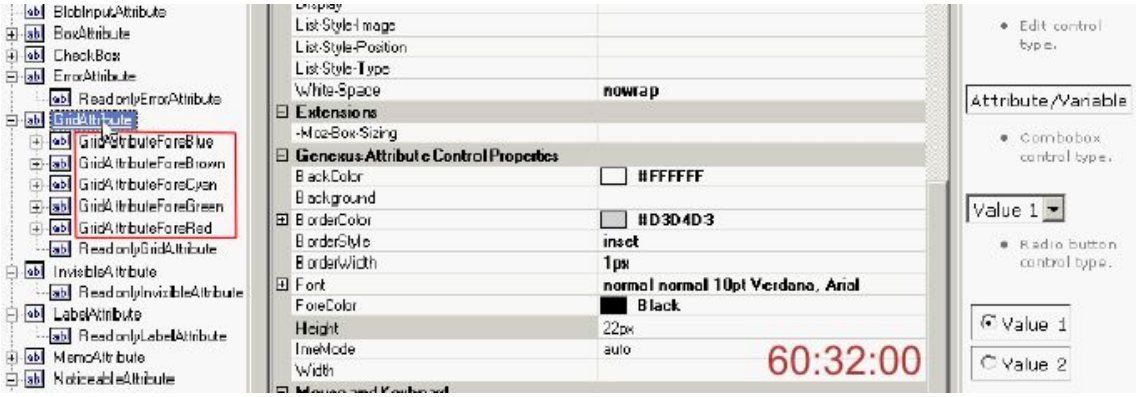
pero no en “negrita”, a diferencia con la ErrorAttribute que sí lo declara.

Son todos conceptos que ustedes los pueden usar o no, incluso las pueden borrar porque no las va a usar el Pattern ni se van a usar en el sistema a no ser que ustedes explícitamente indiquen que “tal” atributo lo quieren con “cual” clase de estas.

Las que sí son requeridas son:

- La que declaren como Clase por defecto para todos los Atributos o Variables que definan en la pantalla (originalmente la Clase Attribute),
- La CheckBox,
- La Memo,

- La RadioButton y
- La GridAttribute, que es la Clase que se aplica por defecto a todos los Atributos que están dentro de la Grilla:



y dentro de la GridAttribure también se crearon conceptos (recuadro) que pueden usar o no, que tampoco aplica el Pattern por defecto y constituyen Clases para aplicar colores en la Font de los atributos. Son los colores más convencionales (el nombre de cada Clase los indica) que podríamos estar usando para marcar diferencias en una Grilla.

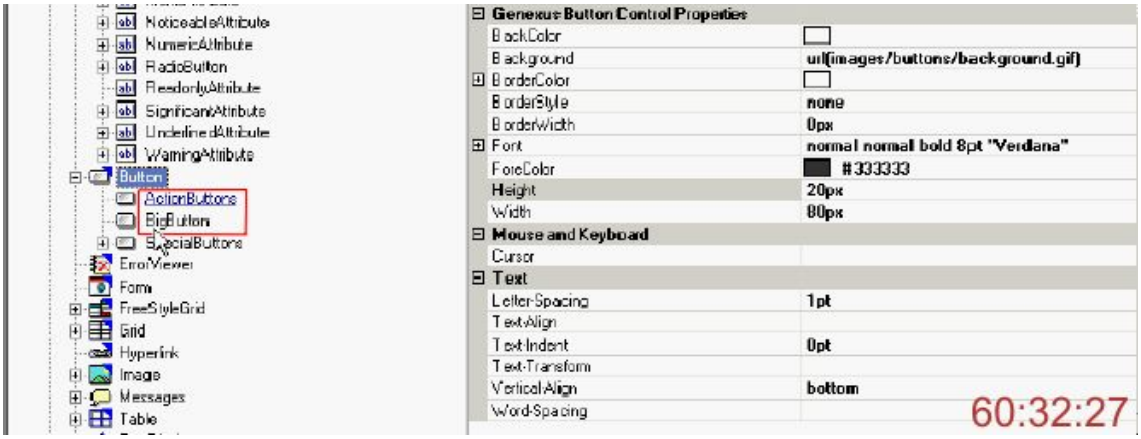
Se trata de las Clases:

- GridAttributeForeBlue
- GridAttributeForeBrown
- GridAttributeForeCyan
- GridAttributeForeGreen
- GridAttributeForeRed

Lo hacemos así porque en vez de aplicar hardcode (con el RGB o la Regla Color) en el Web Panel, es recomendable usar una Clase de GeneXus que nos permita corregir fácilmente los colores que se aplican sin tener que cambiar el programa.

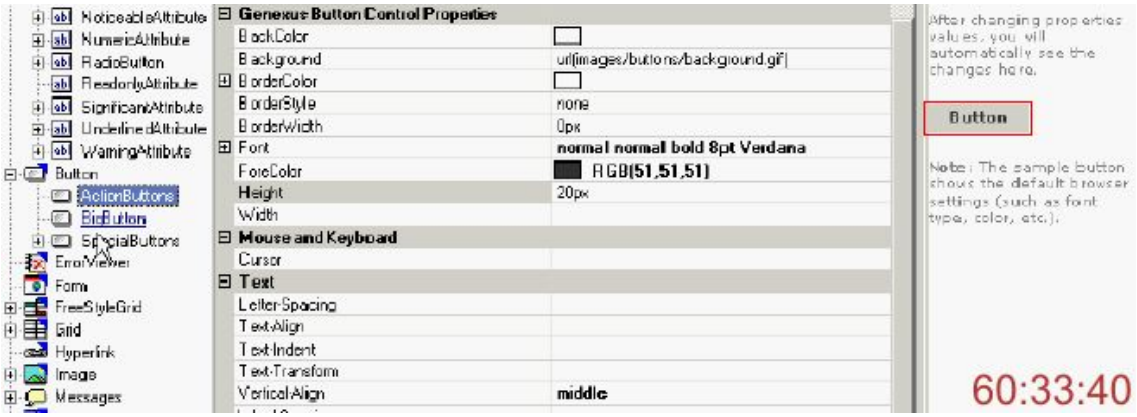
Bien, hasta aquí hemos recorrido todas las Clases relativas a Atributos principales y pasamos a las relativas a los Botones.

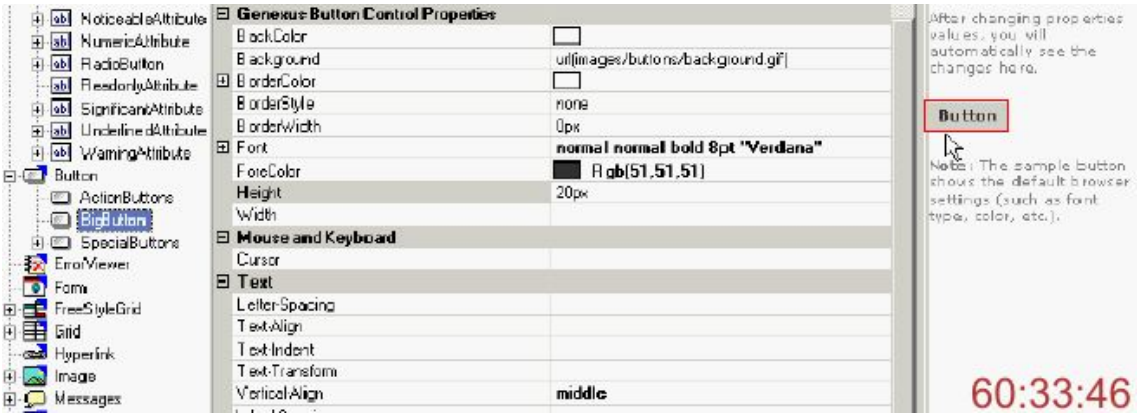
La Clase Button es bastante sencilla, tenemos sólo dos Clases subordinadas:



la ActionButtons y la BigButtons que son las que utiliza el Pattern, la SpecialButtons ya viene por defecto.

El objetivo de que haya dos Clases (que en realidad no se cumple muy bien) era usar la ActionButton para cuando teníamos un Texto muy chico en el botón:



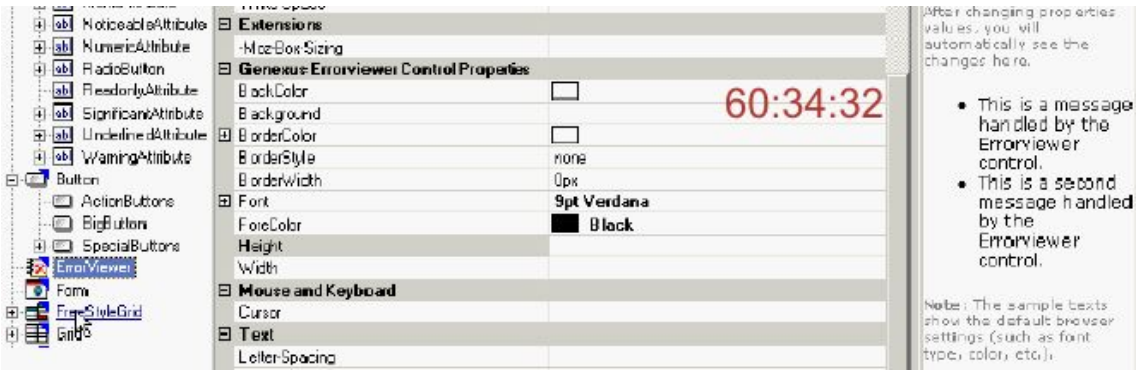


y cuando teníamos un Texto muy grande usar la BigButton.

Si se fijan en la imagen del botón (recuadro) en la figura 60:33:40, verán que los márgenes derecho e izquierdo del Texto (Button) son mayores que en la figura 60:33:46, precisamente porque se espera un Texto más grande.

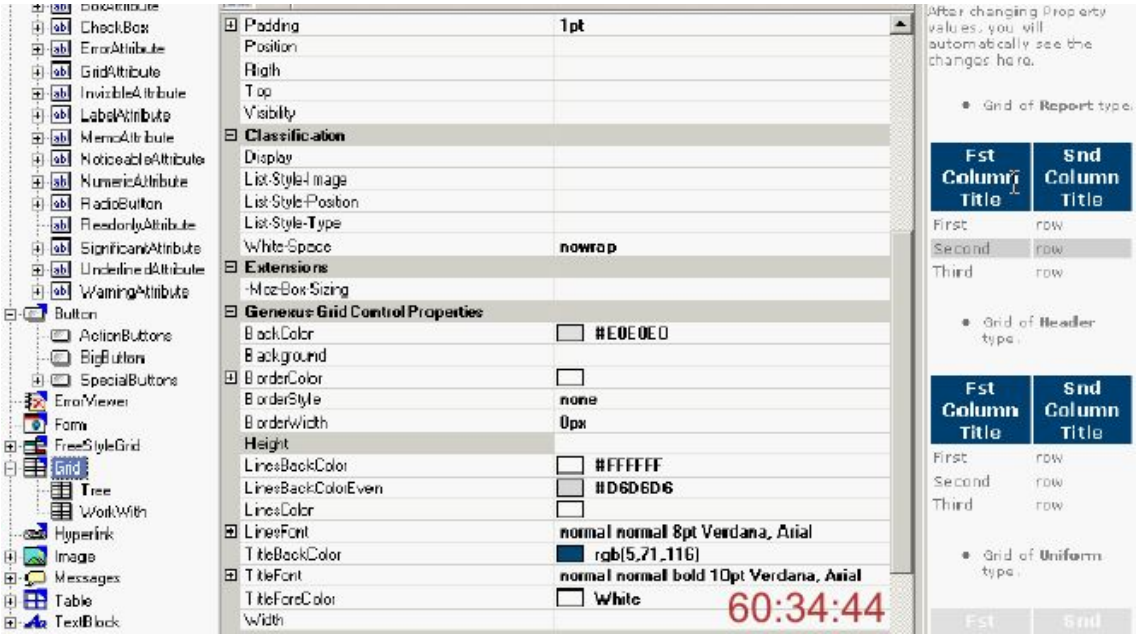
Esto lo determina el Pattern automáticamente dependiendo de la cantidad de caracteres del Texto de cada botón.

Debajo tenemos la Clase ErrorViewer que tiene un solo nodo y es el que viene por defecto para que la adapten porque el Pattern no la utiliza:



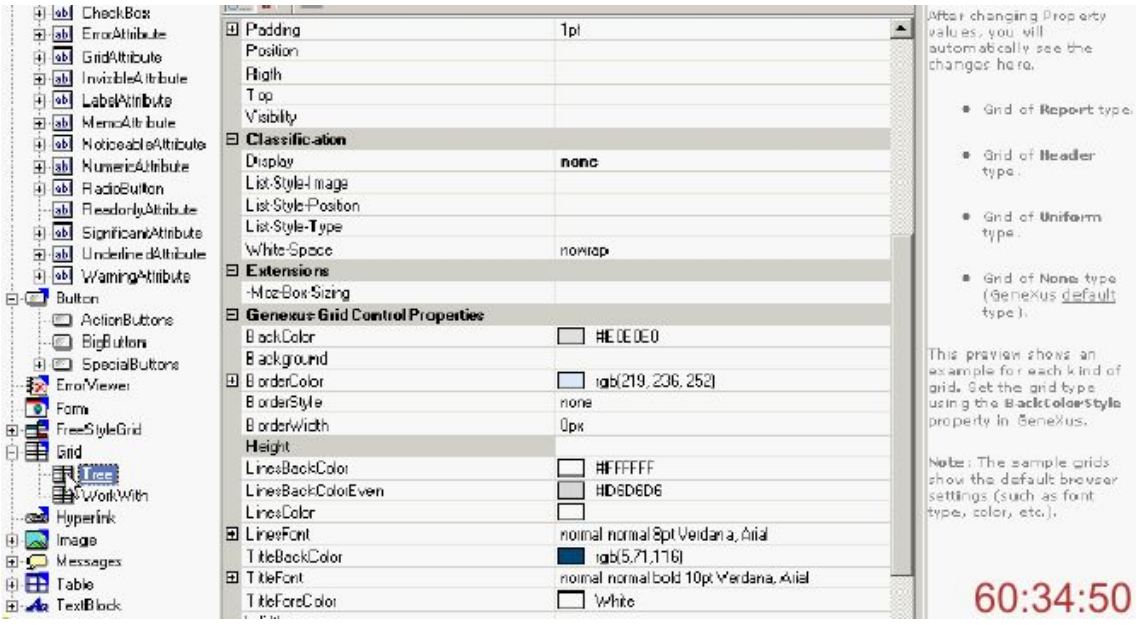
Lo mismo ocurre con la Clase Form y FreeStyleGrid que están debajo porque tampoco el Pattern las utiliza en ningún caso.

Sí utiliza la clase Grid que es la que aplica por defecto:



y tiene el color que vemos normalmente en los títulos y las líneas de las Grillas.

La Clase subordinada “Tree” es la que debemos asociar cuando se utiliza el TreeView:



y lo más importante que tiene es la propiedad de invisibilidad de la Grilla.

De lo contrario la Clase por defecto es Grid, pues la WorkWith que vemos debajo no aplica (en realidad no debería estar).

Por último vamos a profundizar en la Clase Table y en la TextBlock.

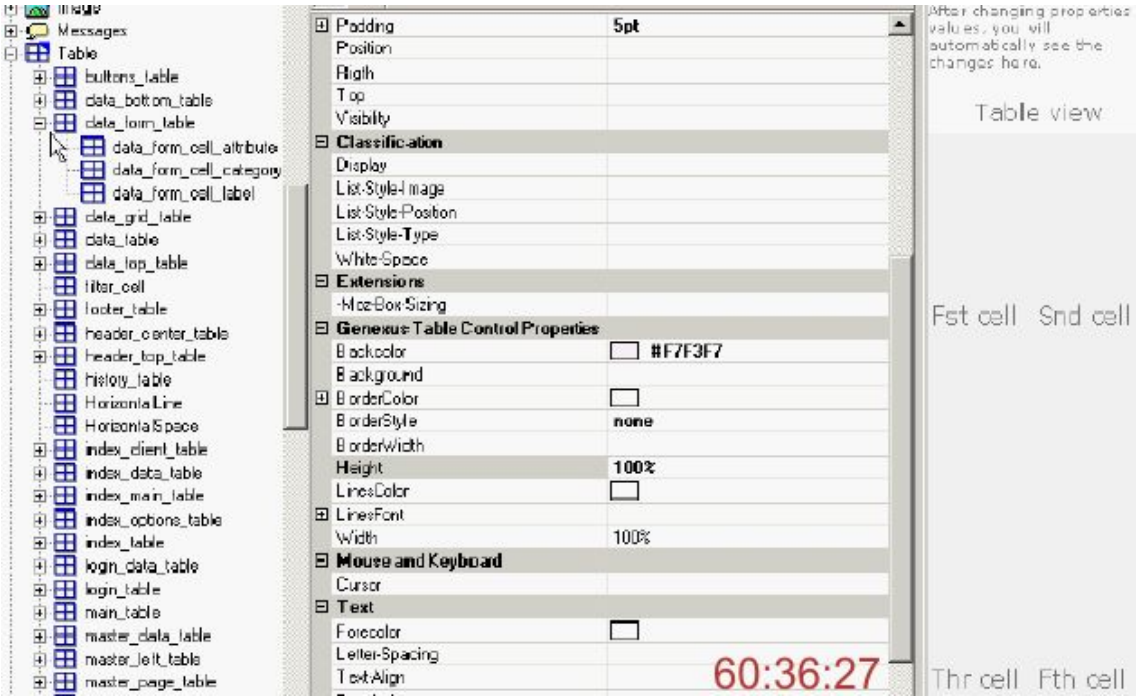
Para las primeras vamos a tratar de explicar un poco la metodología que usamos para la nomenclatura.

Las primeras letras de los nombres de Clases identifican a la ubicación principal en la pantalla.

Por ejemplo, si el nombre comienza con “data”, la Clase refiere a la estructura de pantalla del área de datos, “header” o “footer” refiere a las respectivas áreas de cabezal o pié.

Así tenemos otras para el “historial”, el “login”, la “master_page” además de las que venían por defecto en el WorkWith original que nosotros no las tocamos, de las cuales la única que va a aplicar es la TableBorder que se usa para los Tabs.

Vamos a profundizar en algunas de ellas y comenzamos por la data_form_table:

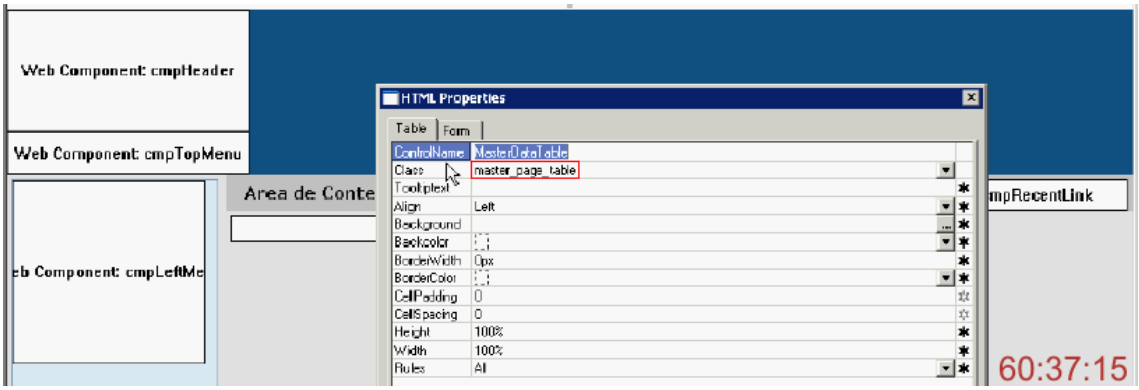


que básicamente representa a tres secciones comunes que se usan en el área de datos mediante sus respectivas Clases subordinadas en la figura anterior.

La sección del Category que es un Título y la celda donde está el Category está asociada a esta Clase data_form_cell_category.

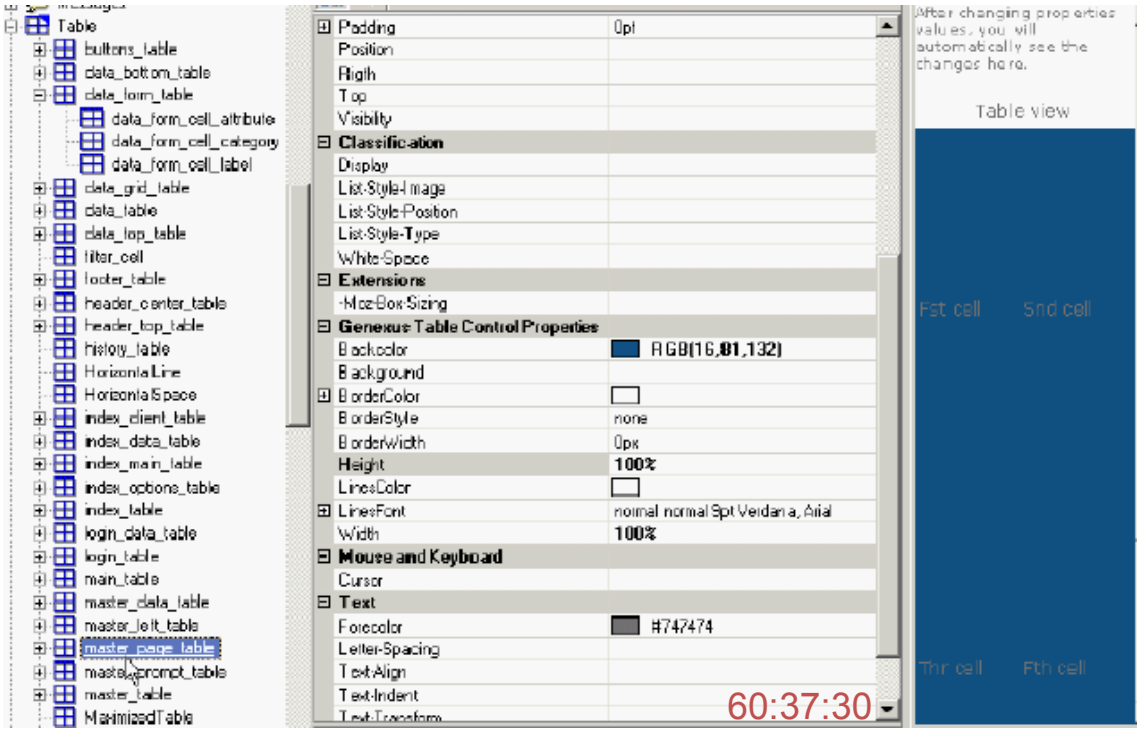
Veamos un poco este tema de cómo asociar una Clase una celda, al cual GeneXus no le aporta una solución directa pero nosotros sí lo hemos resuelto.

Cuando definimos una pantalla, GeneXus nos provee a nivel visual (botón derecho sobre una Tabla del Form del Web Panel y Properties):



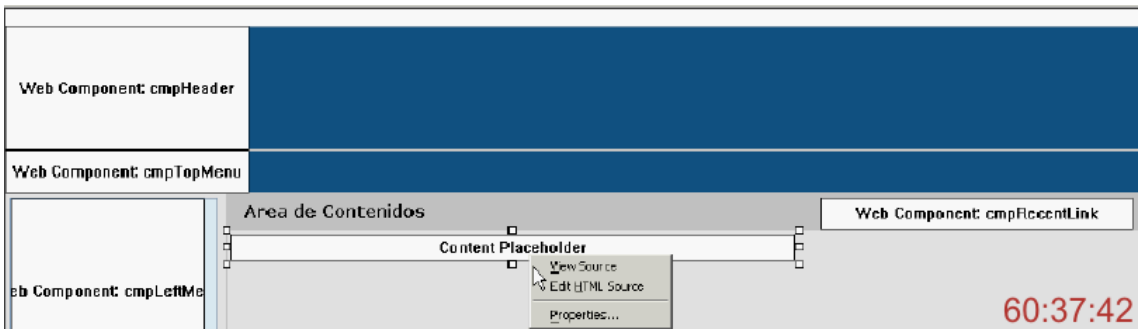
de una Clase (master_page_table en el recuadro) asociada a la Tabla.

O sea que tenemos una Tabla dentro del HTML y una Clase asociada, en este caso a la Tabla principal, la que comanda toda la Master Page y la Clase pertenece al Tema que estamos viendo:

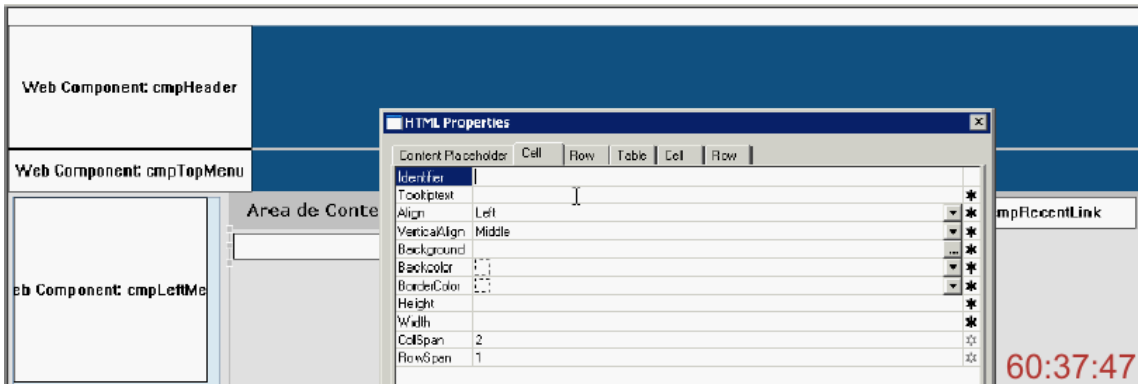


que es ésta, que pinta de azul el fondo de la Master Page.

Pero, si abrimos una Celda, por ejemplo la del Content Placeholder:



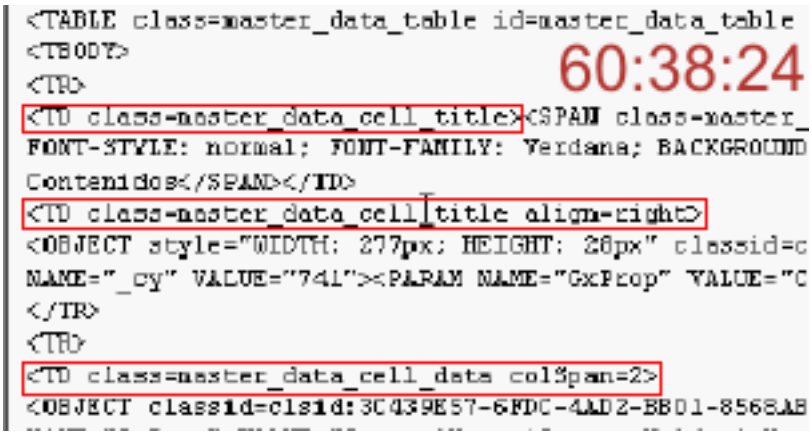
botón derecho y Properties y vamos al Tab Cell que representa una Celda:



vemos que no tenemos la posibilidad de definir una Clase a la Celda en donde está ubicado el Content Placeholder.

Y esto es bastante inconveniente, porque nos obligaría a estar creando muchas Tablas adentro de Celdas para poder aplicas una Clase. Si quisiéramos pintar de un color distinto que el resto a la Celda donde está el Content Placeholder, tendríamos que meter una Tabla para que GeneXus nos permita asociar una Clase.

Si vamos directamente al Source (botón derecho sobre el Form del Web Panel y Edit HTML Source):



vamos a ver casos de Celdas TD que tienen asociado forzadamente la Clase.

Esto no se ve reflejado visualmente con el editor de GeneXus, pero sí vemos que esta resuelto internamente cuando accedemos al editor del Source de HTML.

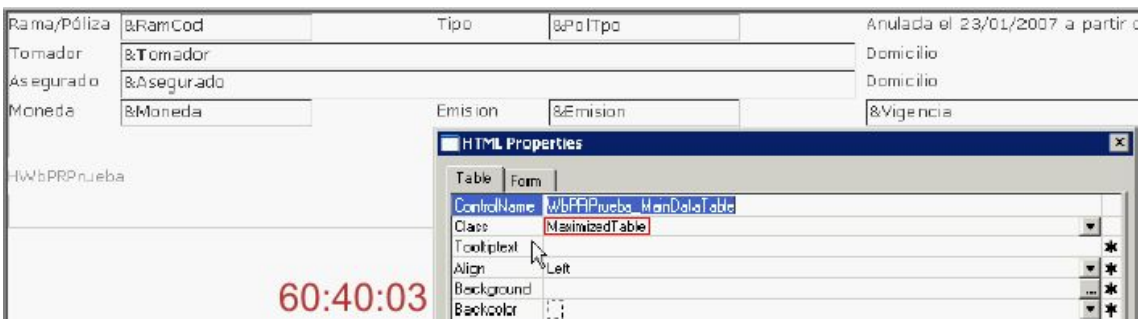
Toda vez que veamos el nombre de una Clase con un nombre que comience con master_data_cell, sabremos que esa Clase está referenciando a una Celda.

Si en cambio, comienza con master_data_table (primera línea de la figura anterior) está referenciando a una Tabla.

Tanto estos asuntos de la Master Page como del Login, sería ideal que lo trabaje alguien que conozca HTML porque toda la parte de diseño gráfico va a tener que aplicarlo alguien que tenga algunas nociones de HTML.

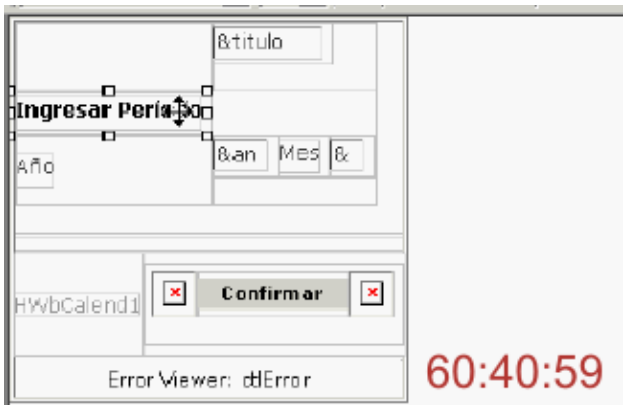
De todos modos quisimos darles algunas nociones de cómo se manejan tales asuntos.

Volviendo a la data_form_table de la figura 60:36:27, cuando definimos una sección del área de datos, arrancamos con la data_form_table y si vamos a un ParameterRequest, por ejemplo, y hacemos botón derecho sobre la Tabla principal y Properties:

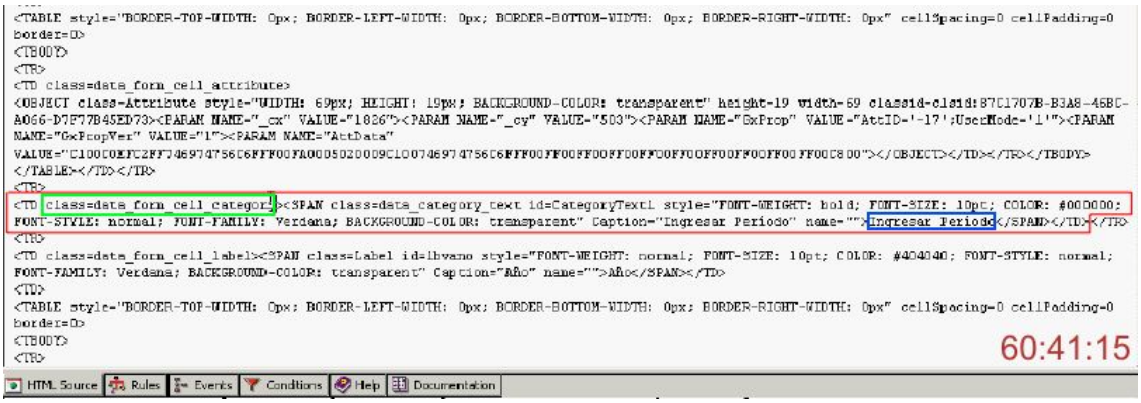


veríamos por defecto la data_form_table, pero justo en este caso está cambiada (a MaximizedTable en recuadro) por una propiedad de la Instancia que permite cambiar esta Clase.

En este otro ejemplo, en el Form, “Ingresar Período” es un Category:



Si hacemos botón derecho sobre el Form y Edit HTML Source:

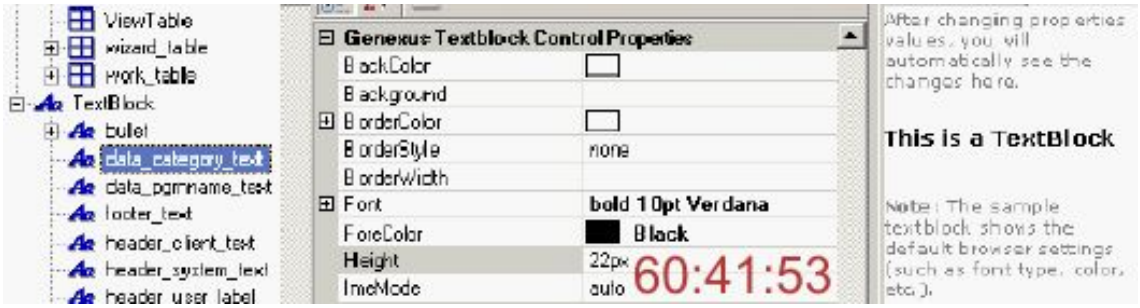


ubicamos en él la Celda (recuadro rojo) donde se encuentra el texto de esta Category (recuadro azul) y vemos que esta Celda tiene asociada la Clase data_form_cell_category (recuadro verde).

Esto está indicando que la Celda donde va a estar el Category se muestre con esa Clase asociada.

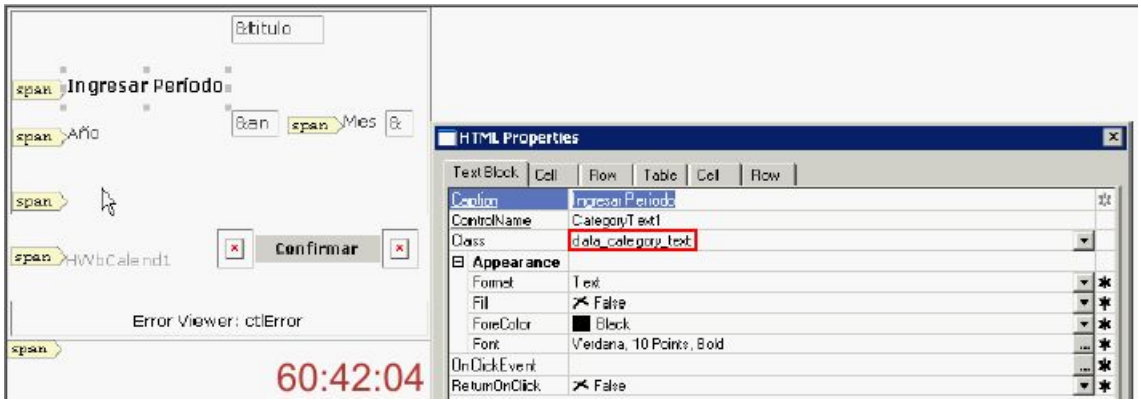
Entonces, por ejemplo, podemos hacer que todos los Category tengan como Background fondo azul. Luego declarando este Background en esta Clase, todos los Category van a tener ese fondo azul.

Hay que diferenciar dos temas, porque también hay un “Label” asociado al Category que es la propiedad “Description”, cuya Clase asociada está entre las subordinadas a la Text Block. Se trata de la Clase data_category_text:



declarada para el Label asociado al Texto de la “Description” del Category.

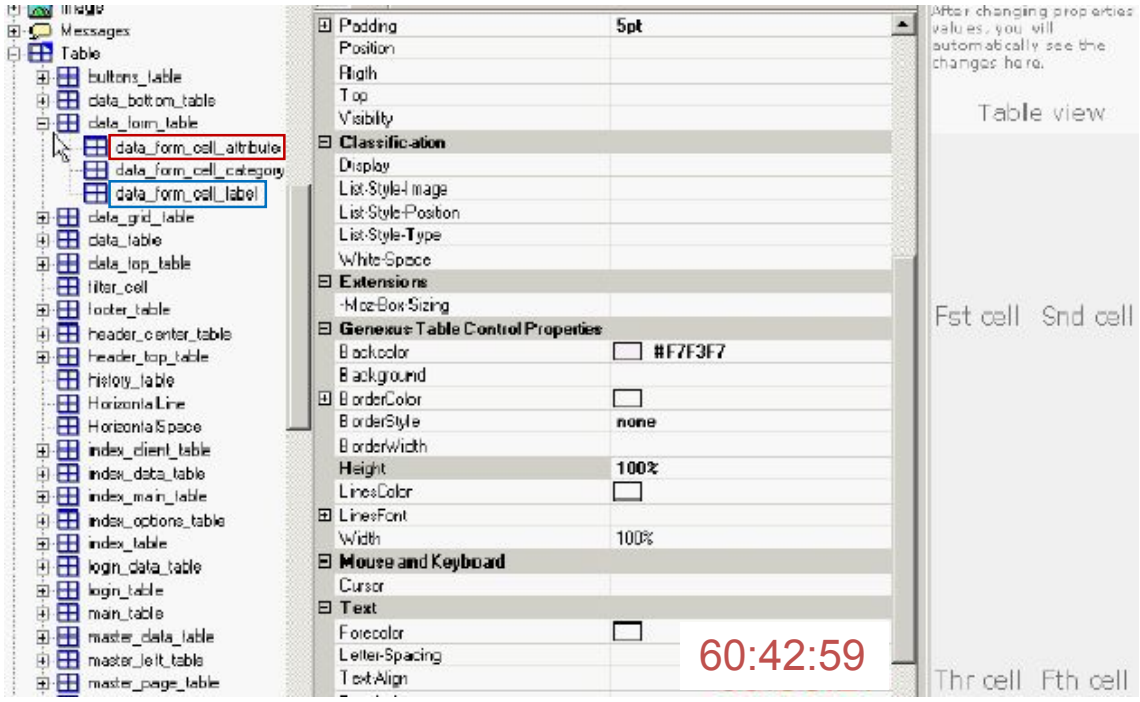
Si nos fijamos otra vez en el Form de este Web Panel y hacemos botón derecho sobre el Texto y Properties:



este Text Black tiene asociada la Clase data_category_text.

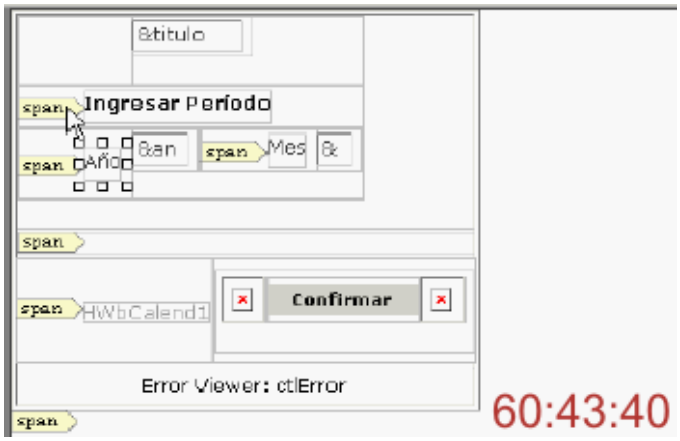
Podrían definir el Background sobre esta Clase, pero esto les va a generar un problema porque en FireFox no es compatible la definición al 100% del Background del Texto.

Entonces, si en la Clase asociada a un Texto definen un Background al 100%, esto es, al tamaño de su contenedor que sería la Celda donde está ese Texto, en Internet Explorer le va a andar bien y pinta el Background del Texto extendido al 100% de la Celda, pero en FireFox no, sólo queda extendido al borde del Texto y no al borde de la Celda, luego si quisieran pintar bien la Celda donde está el Category tienen que usar la Clase data_form_cell_category que vimos al principio:



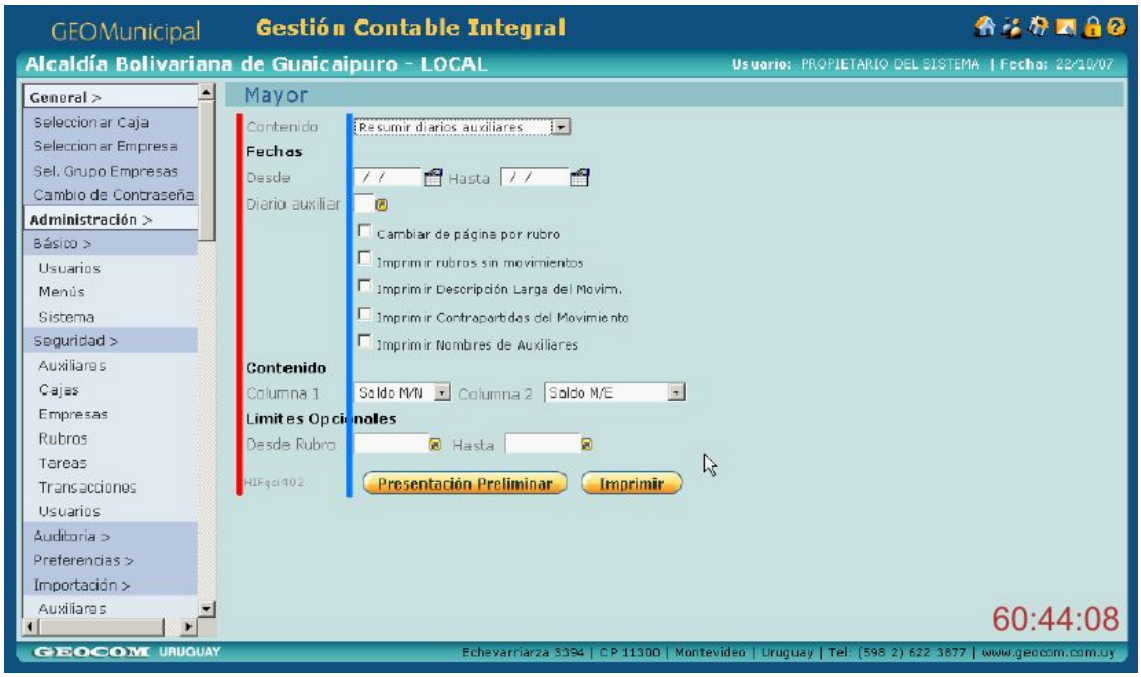
También existen allí otras dos Clases que son la data_form_cell_attribute (recuadro rojo) y la data_form_cell_label (recuadro azul).

Estos dos refieren a lo que tenemos en una pantalla de datos cuando definimos Atributos una arriba del otro, con Categories o sin ellos, donde siempre hay una sección del Form que representa la sección del Label:



Por eso cuando trabajamos con un Category siempre se ve la primera parte del Category alineado.

Si vemos un informe, por ejemplo:



ven que el sistema va a alinear en una primera Celda de cada altura todos los Labels (línea roja) y también va a alinear en una siguiente Celda de cada altura los valores de Atributos o Variables (línea azul).

Es algo parecido a lo que vimos acerca de la propiedad Category Columns Dependant, pero esto ya es más hardcode, por atrás, que determina que el primer Label que se defina siempre va a estar alineado a la izquierda con todos los otros primeros Labels de las demás alturas y el arranque de todos los datos también queden alineados a la izquierda.

Si a alguna altura aparece un Category, ya el resto de su alineación depende de los datos que estemos mostrando, pero el primero siempre queda alineado.

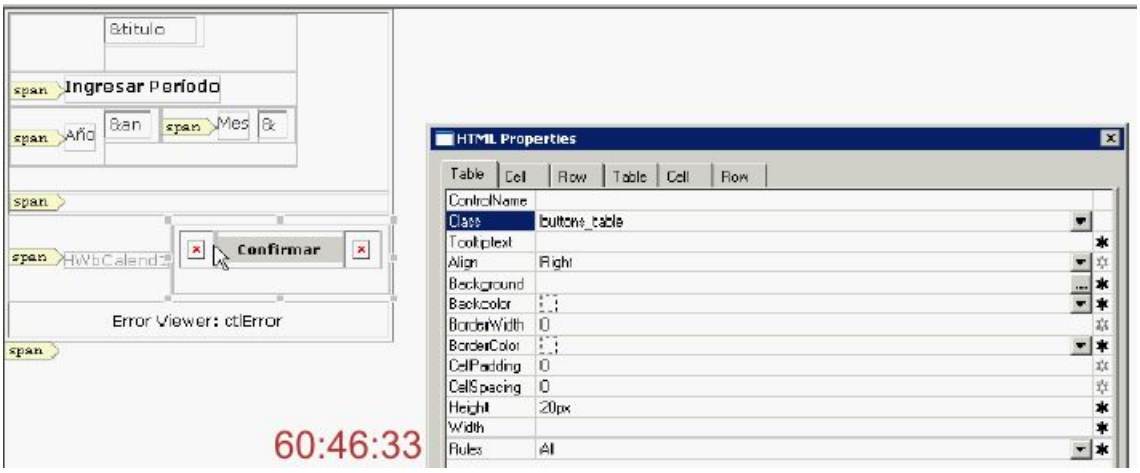
Entonces, esa primera Celda que representa la Celda relacionada con el primer Label (y si no tiene Label, igual se crea una Celda), tiene asociada esta Clase data_form_cell_label de la figura 60:42:59. Su nombre refiere al Cell relacionado a los Labels principales (primeros de la izquierda).

Después tenemos la Clase form_cell_attribute asociada a la siguiente Celda de cada altura con los valores de Atributos o Variables.

Siempre un Form se compone de dos columnas, la de la izquierda relacionada con el Label y la de la derecha relacionada con el Atributo. Si incluimos un Category, en realidad el mismo ocupa las dos columnas y puede abrir otras.

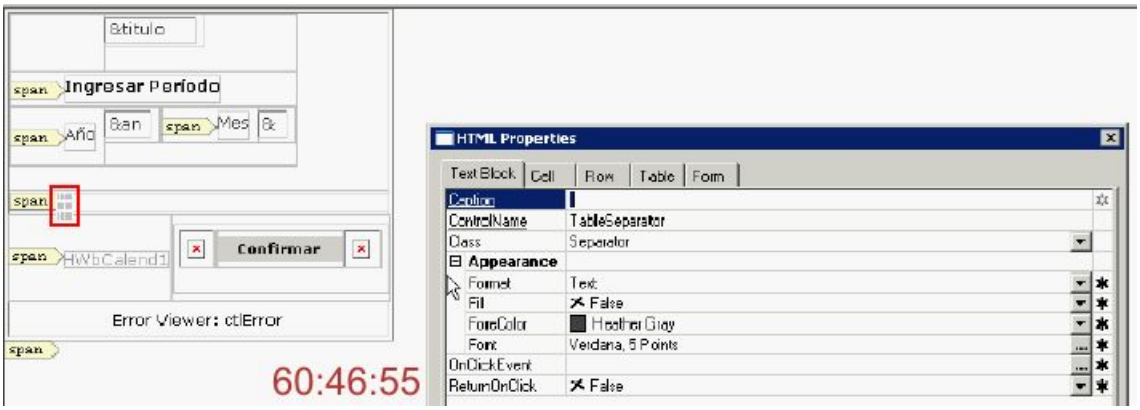
También dentro de la Clase Table que estamos viendo, tenemos otras Clases asociadas con otras secciones y al recorrer las propiedades en el Form de los Web Panels podemos ir descubriendo las que se están aplicando, porque ha sido una norma usarlas para que se puedan manipulas alturas, colores y otras características que se deban adaptar a la estética de los sistemas a desarrollar.

Así tenemos por ejemplo la Clase buttons_table (primera subordinada a la Clase Table en la figura 60:42:59) para asociar a los Botones:



para que toda la estética de los Botones esté resuelta en ese lugar.

Si no son Clases explícitas son Clases internas y adentro de la Tabla está el valor. Por ejemplo:

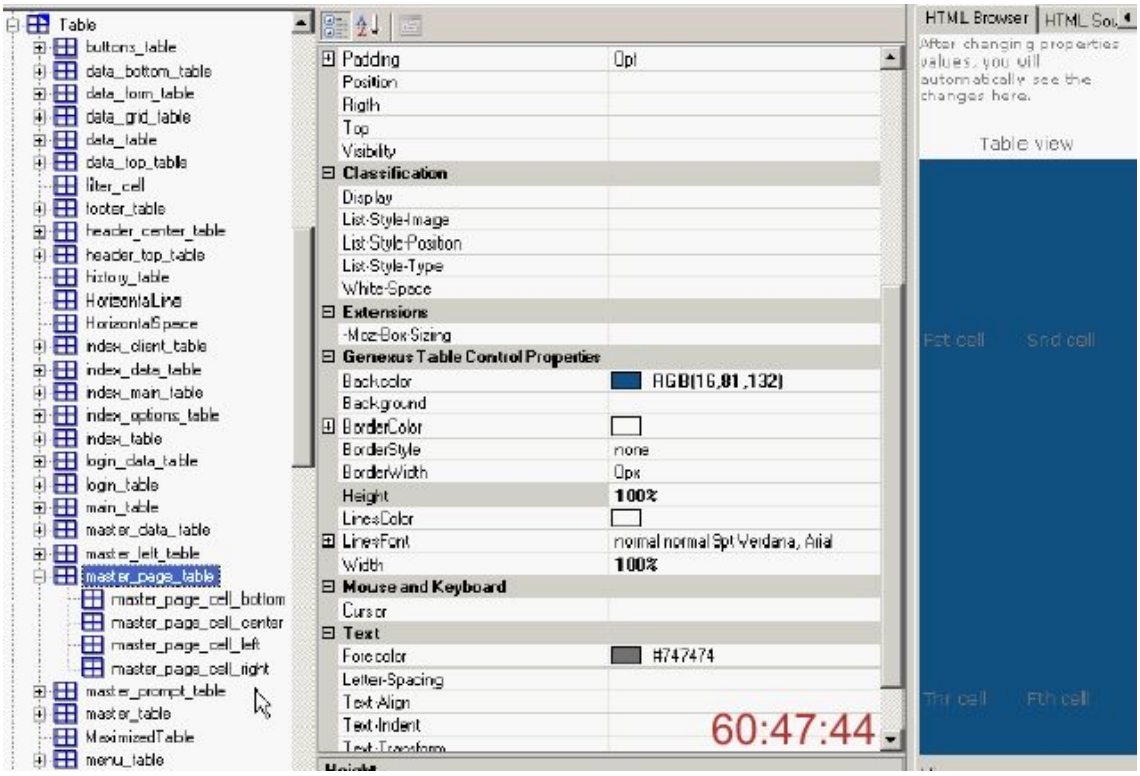


La Tabla seleccionada (recuadro) es un separador, por eso se llama TableSeparator, que tiene asociada la Clase interna “Separator”.

Esto ya venía en el WorkWith original que en los Filtros, por ejemplo, tenía la Tabla de ingreso de datos separada de la Grilla mediante un Span llamado TableSeparator, al que podíamos ajustarle el alto para distanciar adecuadamente la sección de Filtros de la Grilla o, como en este ejemplo, la sección de solicitud de datos del ParameterRequest de los Botones de abajo.

Clases relacionadas con la MasterPage **V 6** **00:47:44**

Por razones de tiempo vamos a mencionar sólo las Clases más importantes del Tema, siguiendo entonces por la master_page_table a la que ya nos hemos referido anteriormente:



que como otros casos tiene debajo las Clases que están relacionadas.

En este caso se trata de Clases para Celdas dentro de la Tabla principal de la Master Page, que representan ciertos elementos que vamos a pasar a ver.

Como vimos, la Master Page cuenta con secciones bien definidas:

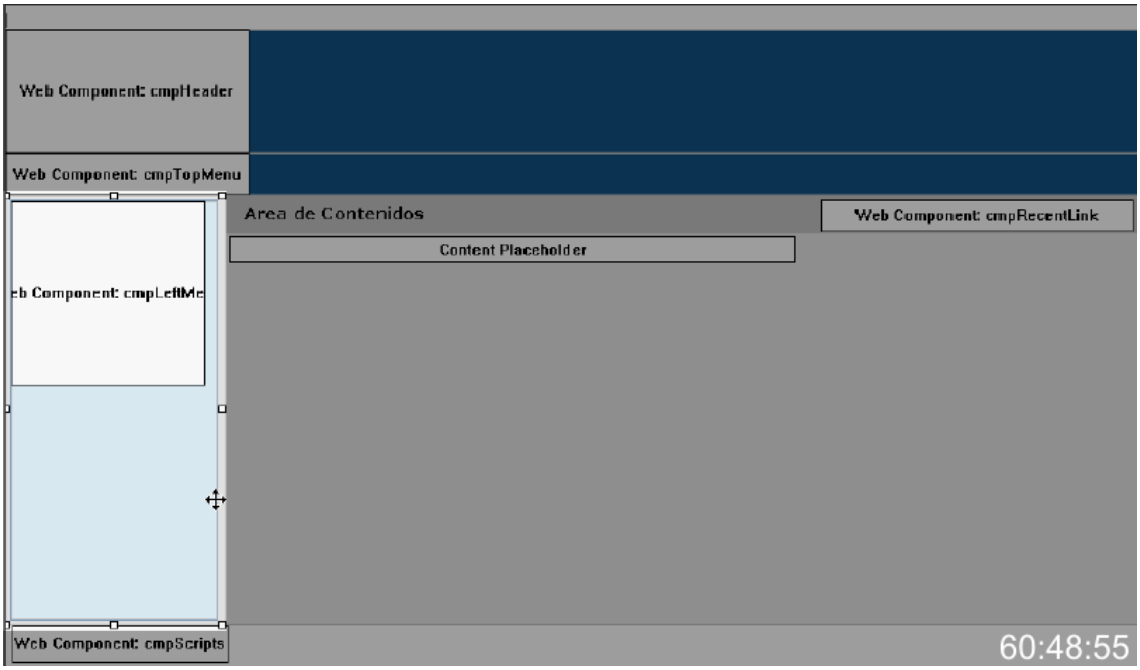
- Una Sección de Cabezal,
- una Sección Izquierda,
- una Sección de Datos,
- una posible Sección Derecha y
- una Sección de Pié.

En la figura siguiente ubicamos estas secciones en el Web Form de una Master Page que no cuenta con Sección Derecha ni Sección de Pié, pues en este caso no las habilitamos:

Y básicamente estos son los elementos que se representan las Celdas de la Tabla principal de la Master Page, a los que les podemos asociar las Clases subordinadas a la master_page_table en la figura 60:47:44.

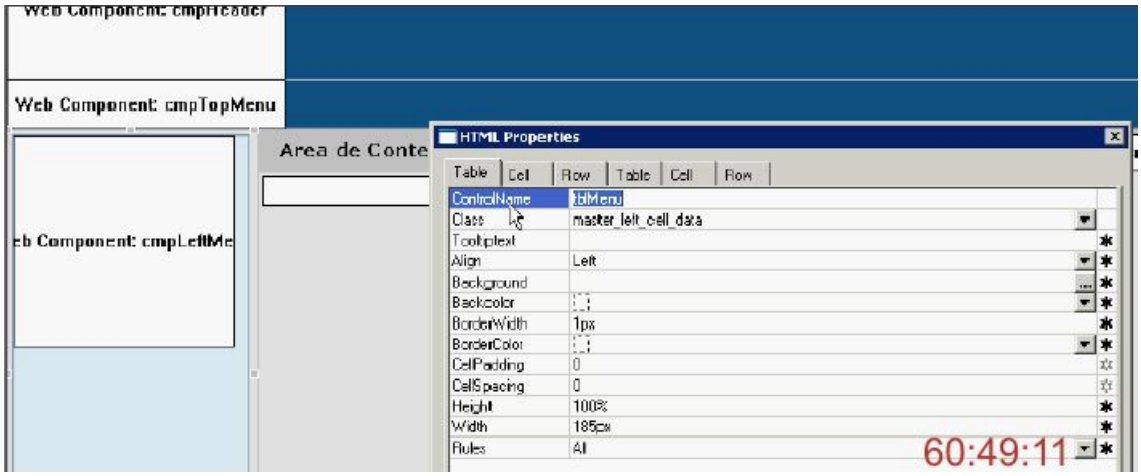
Tenemos la master_page_cell_bottom para la Sección de Pié, la master_page_cell_center para la Sección de Datos, la master_page_cell_left para la Sección Izquierda y la master_page_cell_right para la eventual Sección Derecha.

Esto es para las Celdas de la Tabla principal, después puede haber casos en que dentro de ellas haya a su vez Tablas, por ejemplo:

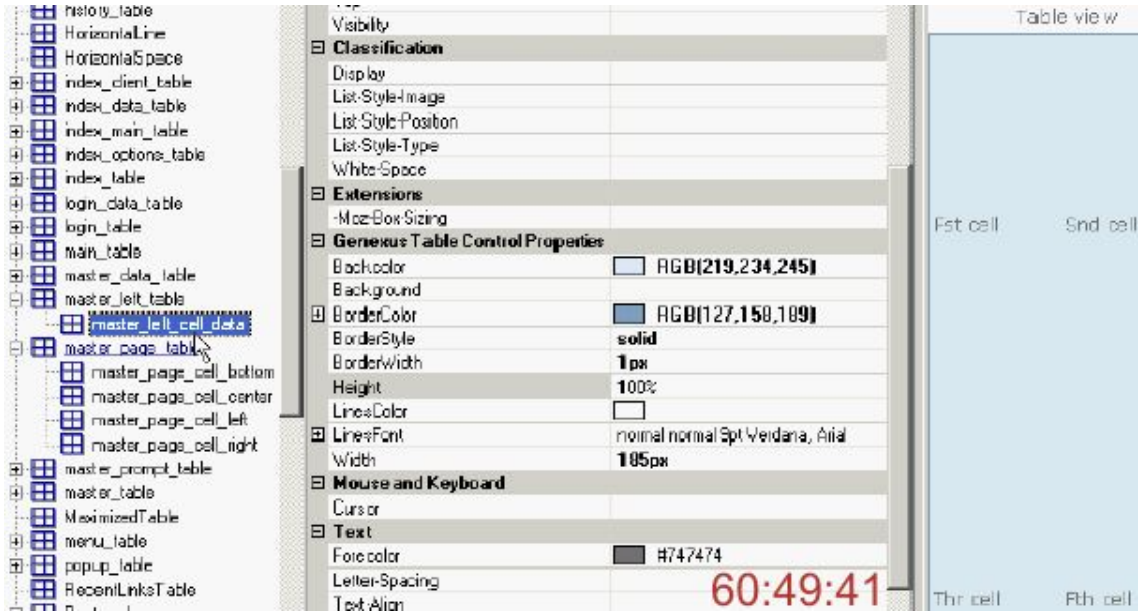


aquí hay una Tabla que es la que le da un bordecito al área de Menús, que no se le dio en la Celda por si es requerido cierto margen interno en esta área.

En este caso se trata de una Tabla con una sola Celda (botón derecho sobre la Tabla y Properties):



cuya estética está resuelta en la Clase master_left_cell_data, subordinada a la master_left_table, encima de las que estábamos viendo:



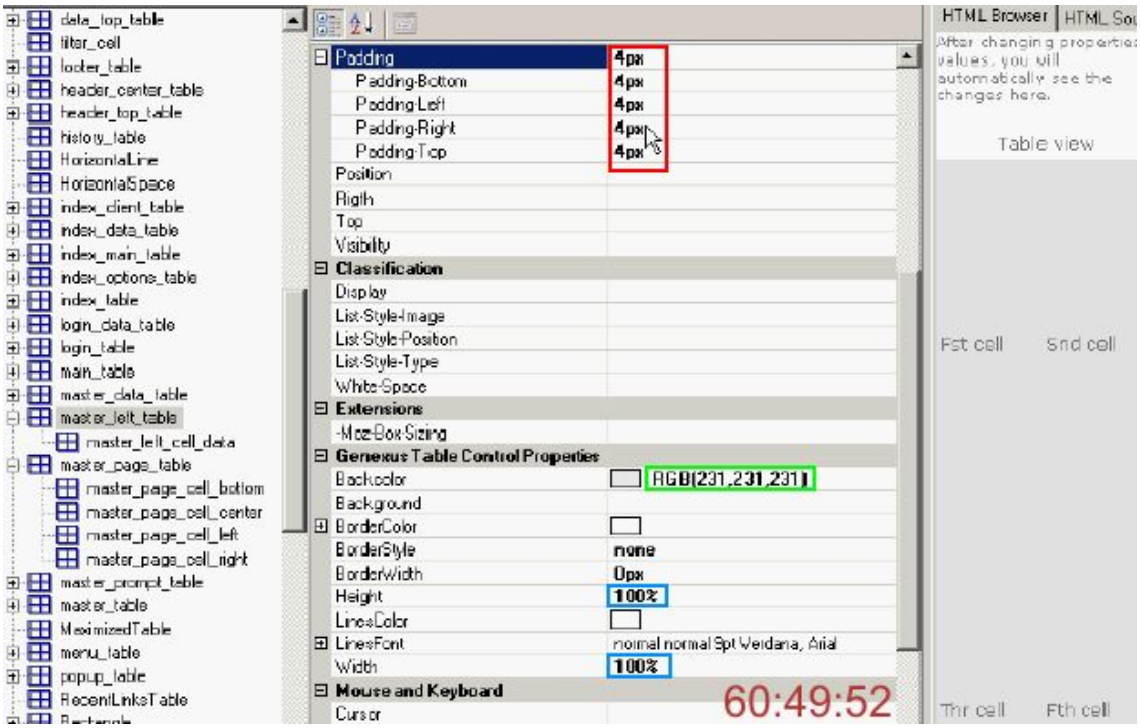
que es la que provoca el efectito del borde en los Menús.

Pero la propia Tabla también tiene juego si le asociamos la Clase "padre" master_left_table porque por ejemplo, en este caso tiene un padding de 4 pixeles que es lo que le da un margen interno (recuadro rojo en la figura siguiente).

Este Padding hace que la Celda esté contenida en la Tabla, enmarcada con un margen de 4 pixeles.

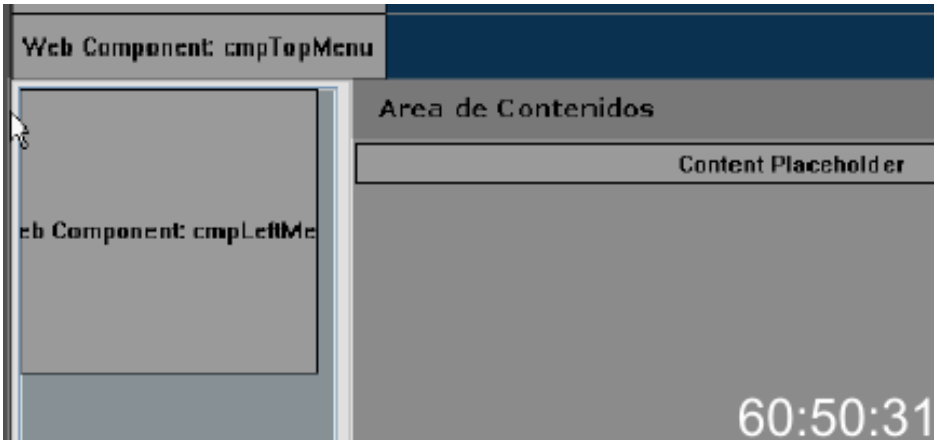
No vamos a ver a fondo cómo funcionan todas las propiedades de cada Clase, eso lo van a tener que leer en el Manual de GeneXus, pero sí vamos a mencionar alguna de las que estamos aplicando en estos casos que estamos mostrando.

Por ejemplo, si se fijan en el resto de las propiedades de esta Clase master_left_table, esta Tabla exhibirá las siguientes características:

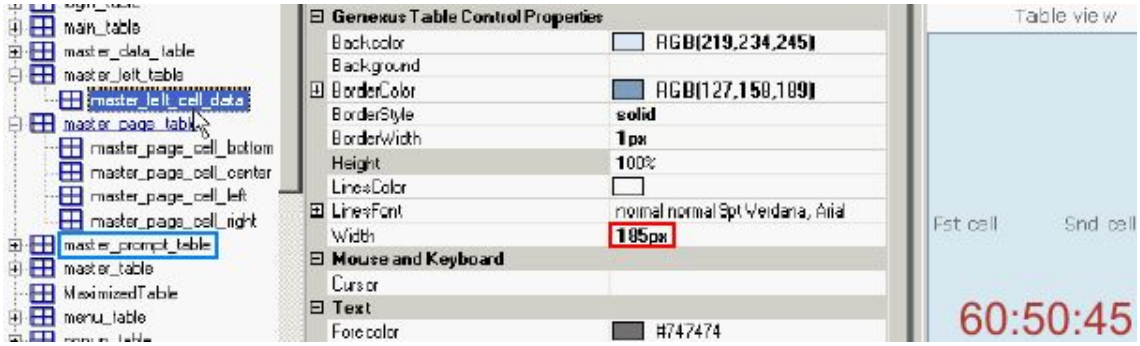


está definida al 100% de ancho y al 100% de alto para que se extienda lo máximo posible dentro de su Celda(recuadro azul), con un Background color grisecito (recuadro verde) que se muestra a la derecha de la figura anterior.

Es el mismo gris del marco de 4 pixeles:



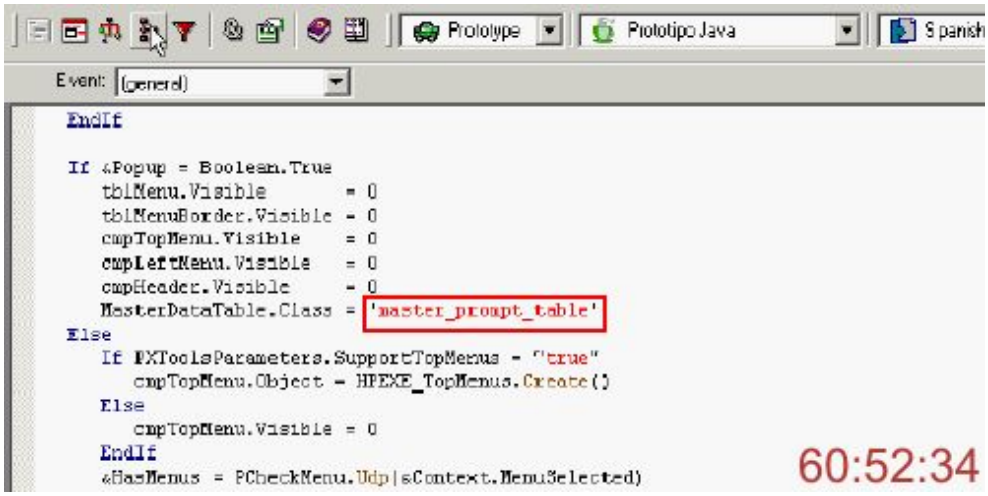
que vemos rodeando el área de Menú Izquierdo en la Master Page, por fuera del bordecito que genera la Clase que ya vimos, asociada a la Celda:



que también está al 100% en el alto y en este caso, ésta es la que está comandando el ancho que se va a ver en la Sección Izquierda (recuadro rojo).

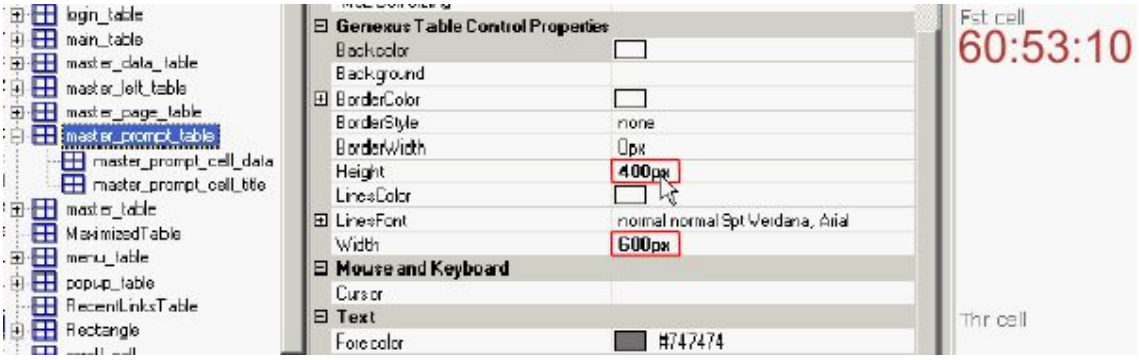
Ya dijimos que anteriormente existía una Master Page principal y una Master Page especial para el Prompt. Ahora como se unificó en una única Master Page, no es que haya perdido vigencia la Clase master_prompt_table que vemos debajo de la master_page_table (recuadro azul en la figura anterior) sino que el sistema automáticamente la aplica en la medida que se da cuenta que se trata de un Prompt y que debe retirar todas las Secciones que rodean a la de Datos.

Si vamos a los Eventos del Web Panel de la Master Page:



vemos que si estamos en estado de Popup, para la Tabla principal, la Clase es la master_prompt_table (recuadro) y si no, la Clase asociada es la que estamos viendo ahora por defecto, que es la master_page_table.

Al analizar las propiedades de la Clase la master_prompt_table:



surge un tema interesante que hay que tener en cuenta.

Las Popups están sometidas a un sistema de auto dimensionamiento que cambia el comportamiento estándar de GeneXus, según el cual, si usamos un Prompt, la Popup se abre sin tamaño predefinido, al mismo tamaño de la última pantalla Popup cerrada por el usuario.

Nosotros preferimos incorporar en todas las Popups una lógica mediante un Javascript que lo que hace es auto dimensionar el tamaño de la Popup al 100% de lo que el contenedor principal de la pantalla permita.

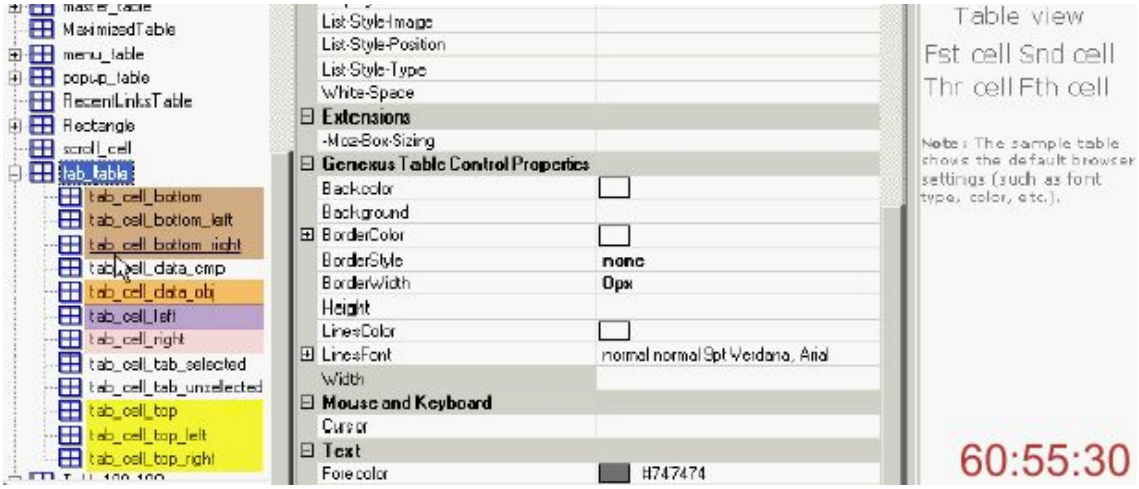
En este caso el contenedor principal de la pantalla es justamente la Tabla a la que está asociada esta Clase master_prompt_table.

Entonces si quieren cambiar el tamaño de las Popups en vuestro sistema, pues deberán cambiar los 400 pixeles de alto y los 600 pixeles de ancho declarados en esta Clase (recuadro) en las propiedades Height y Width respectivamente.

Los Javascripts lo que van a hacer luego es extender el tamaño de la Popup al 100% de lo que permita la Tabla que esta Clase dimensiona.

Clases relacionadas con Tabs en el Área de Datos V 6 00:55:27

Pasemos a ver la Clase tab_table:

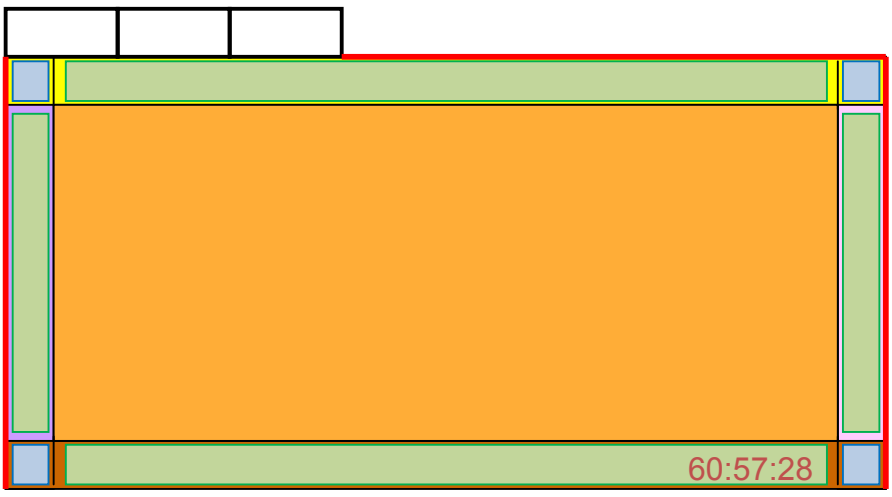


que está definiendo básicamente la representación de los Tabs en Transacciones.

Los Tabs en Transacciones tienen toda una lógica de implementación que nos permite definir ciertos valores de la pantalla.

En particular, todas las Clases que tiene subordinadas se ocupan de dar la estética visual al área de datos de un Tab, motivo por el cual es probable que en algún momento las pasemos al Tab del View.

Vamos a hablar un poquito de esto y a ilustrarles gráficamente partir de la siguiente representación de un Tab:



Las lengüetas superiores serían los Tabs, cuyo alto recuerden que se resuelve con las imágenes que vimos en V 6 | 00:14:12 y debajo tendríamos la representación del área de datos del Tab.

Tuvimos un cliente que quería mostrar un recuadro fuerte en los bordes laterales y superior del Tab (marcado en rojo). Ahora, ¿Como hacer para aplicarle diseño gráfico a esos sectores del área de datos?

En los Tabs del View no existe esa posibilidad, pero en los Tabs en Transacciones sí. Lo que hace el sistema es dividir el área de datos del Tab en nueve sectores:

- En las cuatro esquinas (cuadro azul) es posible aplicar imágenes cuyos tamaños determinan el ancho de los sectores exteriores.
- En los cuatro laterales (cuadro verde) aplica un Background, que como ya vimos en V 6 | 00:04:48 respecto a los botones, no es más que un segmento vertical u horizontal de un pixcel, que después se extiende a todo lo largo del sector por el efecto mosaico de las imágenes de Background.

Así se hace posible jugar con estos recursos para darle un poco de estética al diseño gráfico y por ejemplo, ponerle un marco al área de datos, que es lo que quería este cliente. Desde luego este juego va a requerir la intervención de un diseñador gráfico para el armado de todas las imágenes que se requieren.

Pero el asunto es dónde se asocia todo esto. Para empezar pueden ver que hemos pintado el fondo de las tres alturas de los sectores en la figura anterior, de amarillo en los sectores superiores, de violeta, naranja y rosado para los medios y de marrón para los inferiores, los mismos colores que habíamos elegido para marcar las Clases subordinadas a la tab_table en la figura 60:55:30.

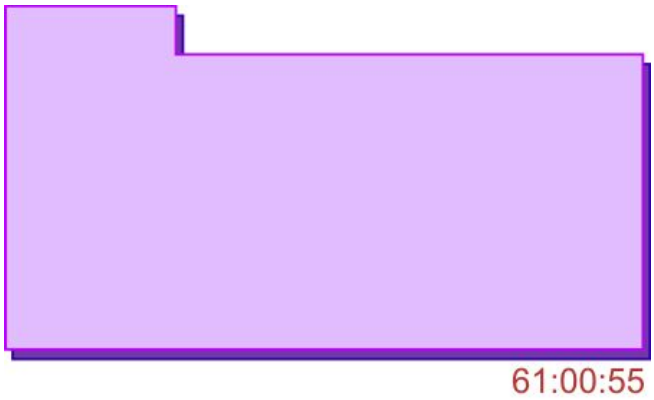
Lo hicimos así porque en cada una de estas Clases así marcadas es donde debemos establecer las propiedades de los sectores que se corresponden con la altura y/o posición que sus nombres indican.

Una de las Clases que nos quedó sin marcar es la tab_cell_data_cmp, porque en realidad originalmente esto era muy genérico (para hacerlo a mano) pero ahora aplica nada más que la de tipo Object para asociar al sector central.

Y en cada una de ellas aplican Clases, aplican Background, aplican Width, aplican Heigth y así pueden manipular el diseño de los márgenes laterales de los Tabs en Transacciones.

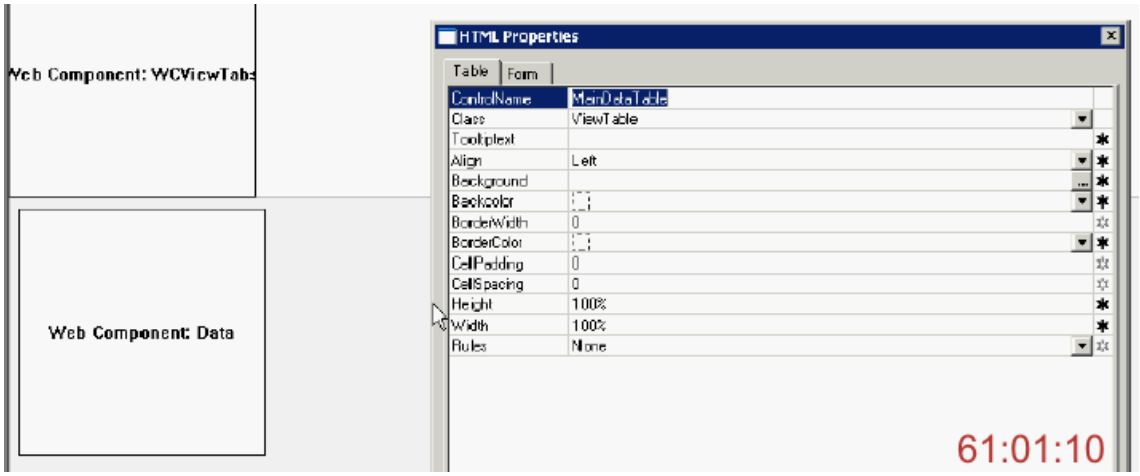
En los Tabs del View está un poco distinta la lógica porque se mantiene como vino originalmente y simplemente se divide en dos secciones:

Aquí se puede definir un borde pero la idea es que podamos tener la posibilidad de definir efectos de sombreado:

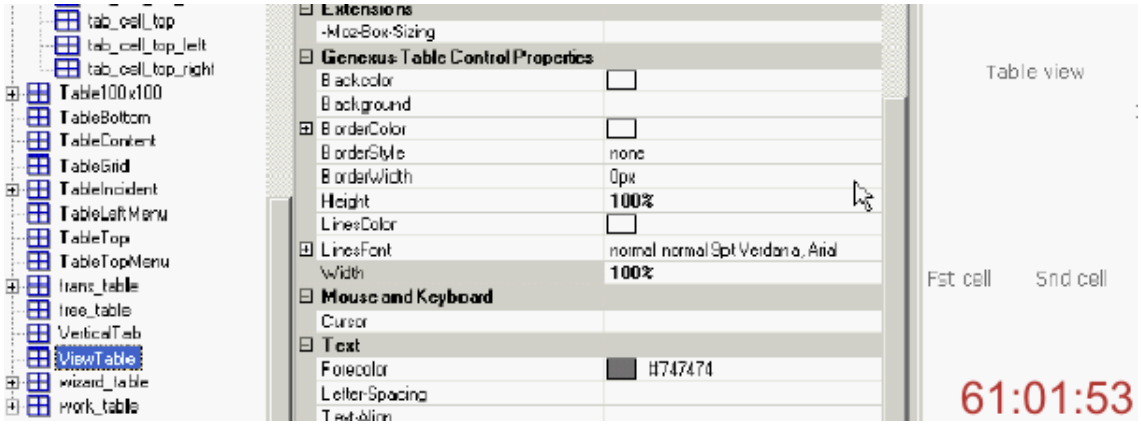


Si quisiéramos hacer un efecto sombra para darle profundidad al Tab, deberíamos tener la posibilidad de definir un Background para el borde derecho e inferior del Tab.

En el caso del View, lo único que tenemos es una Tabla en la sección del Web Component Data y si hacemos botón derecho sobre esa Tabla y Properties:



vemos que le podemos asociar la Clase ViewTable:



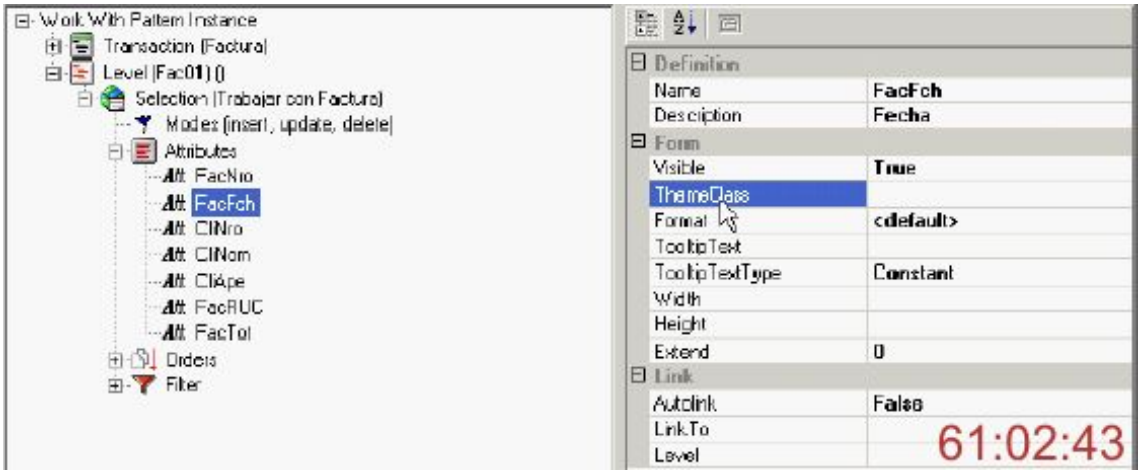
pero hay que tener cuidado, no puede ponerse aquí un Background para esa

Tabla porque obviamente el View puede achicarse o agrandarse, dependiendo de los contenidos que tiene dentro.

Esta Clase es la que pinta normalmente el área de datos de los Tabs del View. No de los Tabs de la Transacción, del Form, sino del nodo View de la Instancia.

Modificación al tema PXTools **V 6 01:02:20**

A nivel de la Instancia, ustedes pueden incorporar algunas Clases “a mano”, pudiendo aplicar, no las que brinda el Pattern por defecto, sino alguna Clase que hayan definido especialmente. Yeso se puede hacer normalmente en bastantes lugares, por ejemplo:



en los Atributos mismos hay siempre una propiedad que se llama “ThemeClass”.

Entonces, si tenemos un Atributo o una Variable en la pantalla y queremos forzar a que ese Atributo o Variable tenga una Clase asociada, como suele ocurrir por ejemplo con el Error Attribute o el Warning Attribute o cualquiera de esas Clases que ya hemos definido y que se pueden utilizar, aplican acá y le ponen esa Clase de su preferencia.

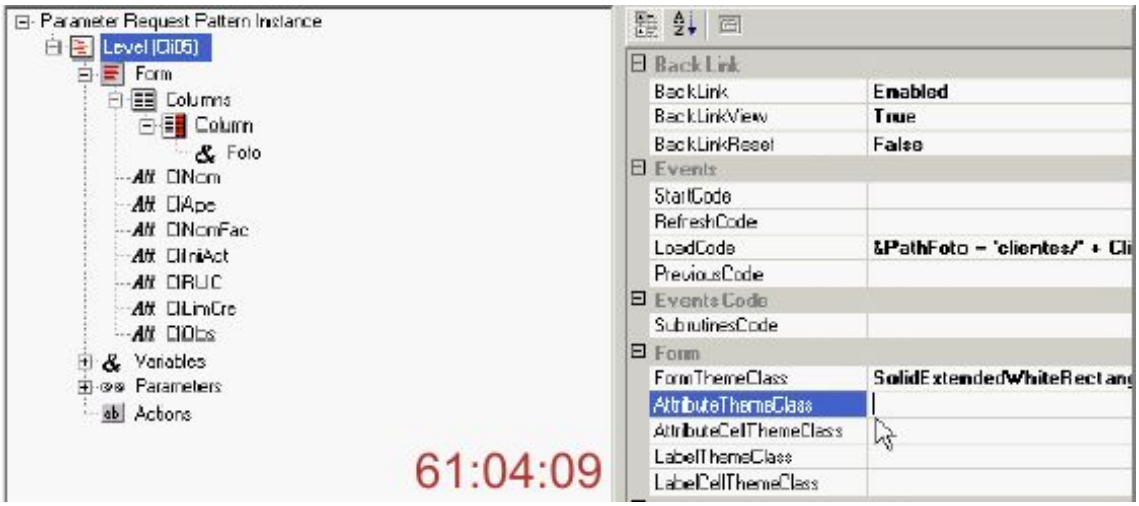
De lo contrario el sistema va a asociar la Attribute, que es la Clase por defecto, o la CheckBox o la RadioButton si definen un control de edición.

A nivel del ParameterRequest se agregó también la posibilidad de que definan en el propio nodo Level algunos valores por defecto.

Por ejemplo nos pasaba que como a los Parameter Request en muchos casos los habíamos empezado a poner como elemento de un componente principal, queríamos que todos los Atributos que se vieran en esa sección tuvieran una Clase distinta a la normal.

Entonces, en lugar de tener que recorrer cada uno de los Atributos que teníamos en cada una de las secciones del Form del ParameterRequest, hay en el nodo principal, en el Level, una propiedad “AttributeThemeClass” en donde se puede asociar una Clase por defecto a todos los Atributos que están en el Form.

De modo que aplicando una Clase al “AttributeThemeClass”:



se logra el mismo resultado que cargando dicha Clase en cada uno de los Atributos que vemos debajo en la figura, por el procedimiento anterior.

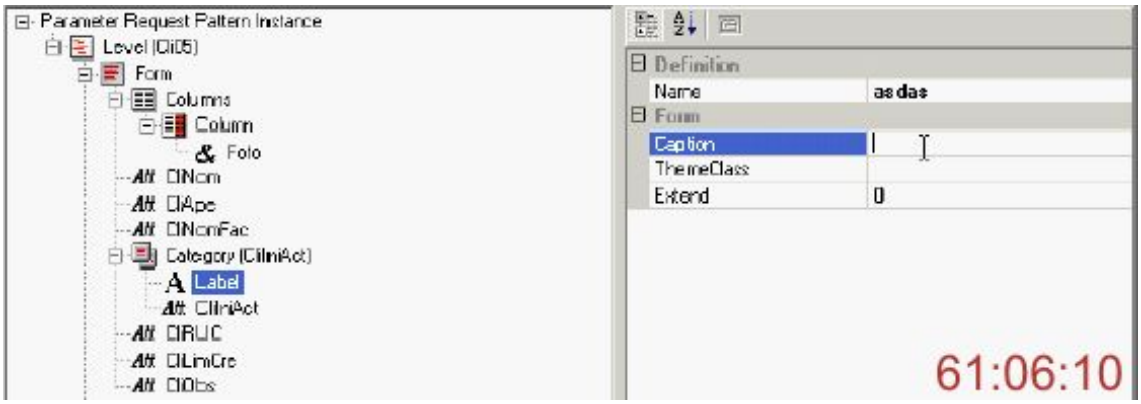
Lo mismo ocurre con el “Label”. Mediante la propiedad “LabelThemeClass” debajo de la AttributeThemeClass, se puede asociar una Clase por defecto a todos los campos tipo Labels que están en el Form de un ParameterRequest.

El ParameterRequest por ahora es el único Pattern que puede aplicar estos campos tipo Label en el sistema.

Los Labels se diferencian de la “Description” del Atributo (figura 61:02:43) que también son labels, precisamente en que a estos últimos no es posible asociarles una Clase distinta de la que le asocia el sistema por defecto a la Instancia, que es una Clase del TextBlock que vamos a ver en seguida.

Si ustedes quisieran que una Description de un Atributo en particular tuviera asociada una Clase distinta a la asociada por defecto, lo que normalmente hay que hacer es agregar un Category donde va a estar el Atributo, meterle dentro al Atributo (al que no le declaramos la Description), agregarle un Label adelante y como éste va a ser el primer label antes del Atributo, automáticamente va a sustituir a la Description ausente.

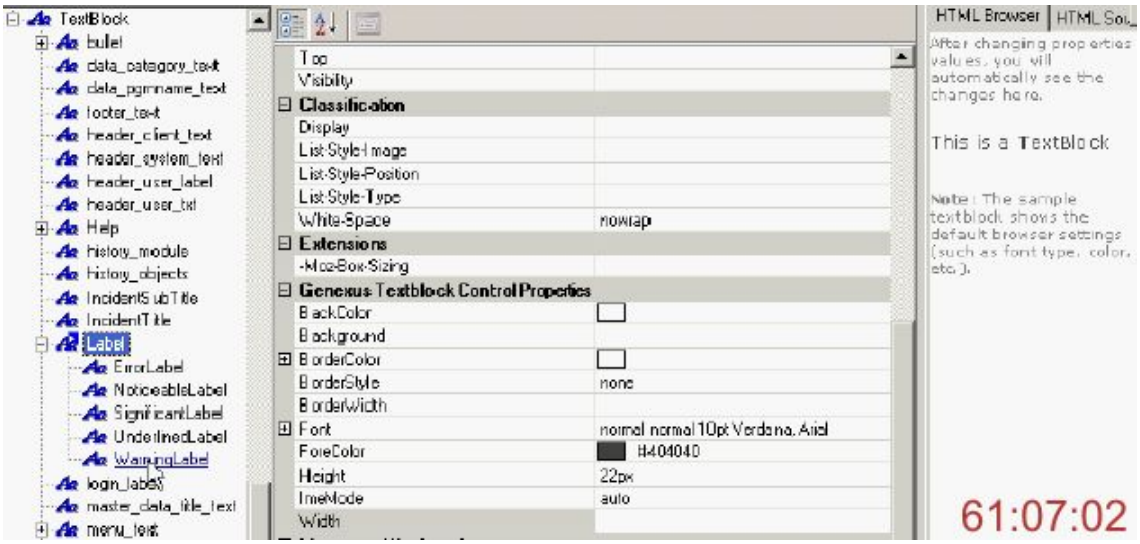
A este Label le podemos poner un Caption:



que va contener el valor que no pusimos en la Description y debajo, en la propiedad “ThemeClass”, ahora sí podrían aplicar una Clase particular.

El objetivo del Label fue justamente éste, darle al programador la posibilidad de asociarle una Clase en espacial a un label de un Atributo.

Esa Clase del TextBlock a la que nos referimos recién, que asocia por defecto el sistema a los Labels de la Instancia:

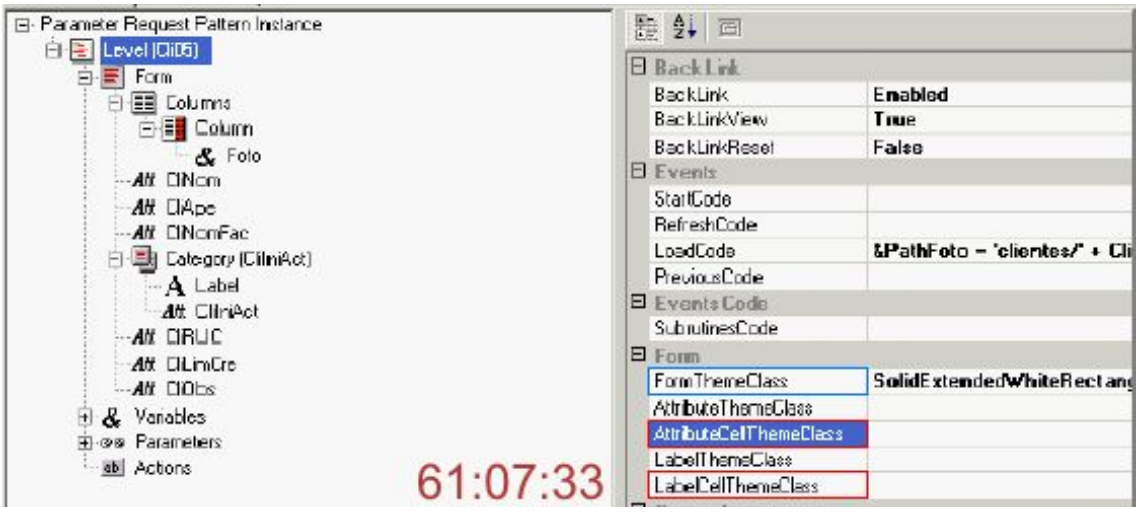


es la Clase “Label” y al igual que la Clase Attribute tiene subordinadas una serie de Clases que son conceptos que ustedes pueden manipular para adecuar su diseño a las especificaciones gráficas del sistema a desarrollar.

También al igual que vimos en la Clase Attribute, a estas Clases subordinadas no las usa el Pattern ni se usan en el sistema a no ser que ustedes explícitamente apliquen alguna de ellas haciendo lo que les acabamos de mostrar.

Entonces tenemos tanto para los Atributos como para los Labels la posibilidad de indicar en el nodo Level una Clase por defecto en esa Instancia.

Lo mismo pasa con las Clases por defecto para las Celdas y para el Form:



propiedades (las relativas a las Celdas) que fue necesario implementar por las razones de compatibilidad entre el Internet Explorer y FireFox que vimos al analizar la figura 60:42:04 con relación la label “Description” del Category.

Entonces vimos que si en la Clase asociada a un Label le definen el Background o el BackColor (el color del Background) al 100%, o sea, al tamaño

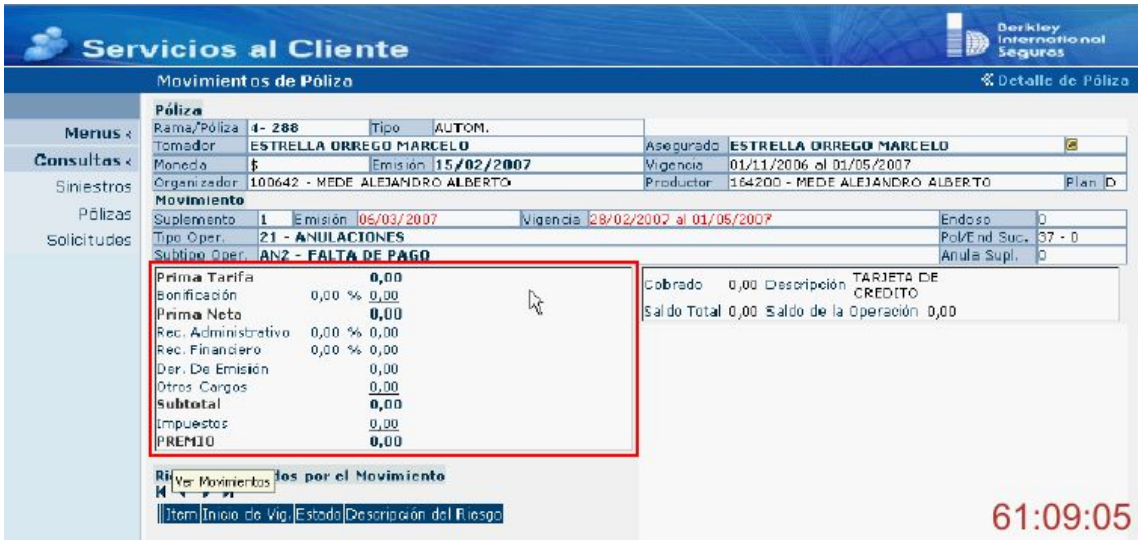
de su contenedor (que sería la Celda donde está el Texto del Label), en Internet Explorer le va a andar bien y pinta el Background del Texto extendido al 100% de la Celda, pero en FireFox no, sólo queda extendido al borde del Texto y no al borde de la Celda.

En FireFox es necesario aplicar la Clase a la Celda donde está el Label o a la Celda donde está el Atributo, cosa que se resuelve con las dos propiedades recuadradas en rojo en la figura anterior:

- AttributeCellThemeClass
- LabelCellThemeClass

También vemos allí a la propiedad FormThemeClass (recuadro azul) que brinda la posibilidad de definir una Clase para todo el Form.

A la Tabla principal del ParameterRequest ustedes le pueden definir una Clase (SolidExtendedWhiteRectangle, en el ejemplo), que en el caso de este cliente genera una figura:



de un rectángulo con marco de efecto Inset y fondo blanco donde se visualizan los elementos del ParameterRequest dentro del Composer (recuadro rojo).

En este caso además, está cambiada la forma establecida por defecto para visualizar los Labels, que deben verse sobre fondo blanco y no celeste como en otras partes de la pantalla y por tanto tienen ForeColor negro en lugar de azul. También podemos ver que los valores de algunas variables encima de los totales se muestran subrayados, cerrando el área de subtotales.

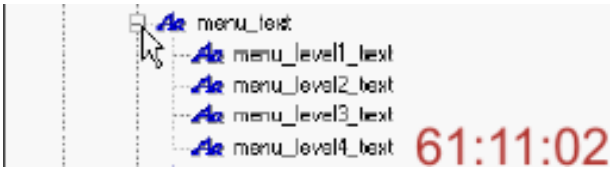
El ParameterRequest que se muestra en la parte superior de esta pantalla nos sirve para ilustrar el comportamiento de las propiedades AttributeCellThemeClass y LabelCellThemeClass que acabamos de ver.

La Clase aplicada a la propiedad LabelCellThemeClass determina el fondo celeste con marco y Texto en azul que lucen los Labels y la aplicada a la propiedad AttributeCellThemeClass determina el fondo blanco con marco azul y Valor en negro para los Atributos.

Obviamente debajo de la Clase Table hay un montón de Clases más, pero normalmente uno va a entrar a la Master Page y va a empezar a ver que cosa quiere cambiar, va a ver que Clase tiene asociada y allí mismo crea y aplica la nueva Clase.

Dentro de la Clase TextBlock para los Labels, ya mencionamos algunas, entre las cuales la Clase Labels que vimos en la figura 61:07:02 es la principal.

Después tenemos las Clases de TextBlock relativas al Menú de la izquierda:



con los correspondientes subniveles.

Si se fijan, cada nivel de apertura de Menús tiene una Clase distinta:

Los de nivel 1 tienen un color de fondo y un tipo de letra (recuadro rojo), los de nivel 2 tienen otros (recuadro azul) y los de nivel 3 otros (recuadro verde). En realidad es posible definir tantos niveles como el sistema requiera.

¿Cómo se representa esto a nivel del Tema?

Esto involucra dos cosas, la primera es la Clase que define el color, tipo y tamaño de letra del Menú a nivel del Texto, que es lo que muestra la figura 61:11:02.

Y después tenemos también a nivel de la Clase Table:

las Clases relativas a este Menú con sus correspondientes subniveles y opciones de Mouse. En la figura 61:11:18, al posar la manito del Mouse sobre

el primer Menú de nivel 1 cambió el fondo de la Celda para indicar su posible selección.

Lo mismo ocurre en los demás niveles, para todos los cuales tenemos la posibilidad de definir estos dos efectos de Background que se identifican con el sufijo “out” y “over” según no esté o sí esté el Mouse sobre el Menú.

Y en esto está presente una vez más el problema de la compatibilidad con FireFox, debido al cual estos Backgrounds no se asocian al Texto sino que se asocian a estas Clases que estamos viendo, que aplican a la Celda de cada opción de Menú.

En el ejemplo que estamos mostrando están definidos cuatro niveles de Menús. Si ustedes requieren más niveles sólo deben agregar las Clases correspondientes y el sistema los va a implementar en forma automática pues está programado para cargar la Clase dinámicamente.

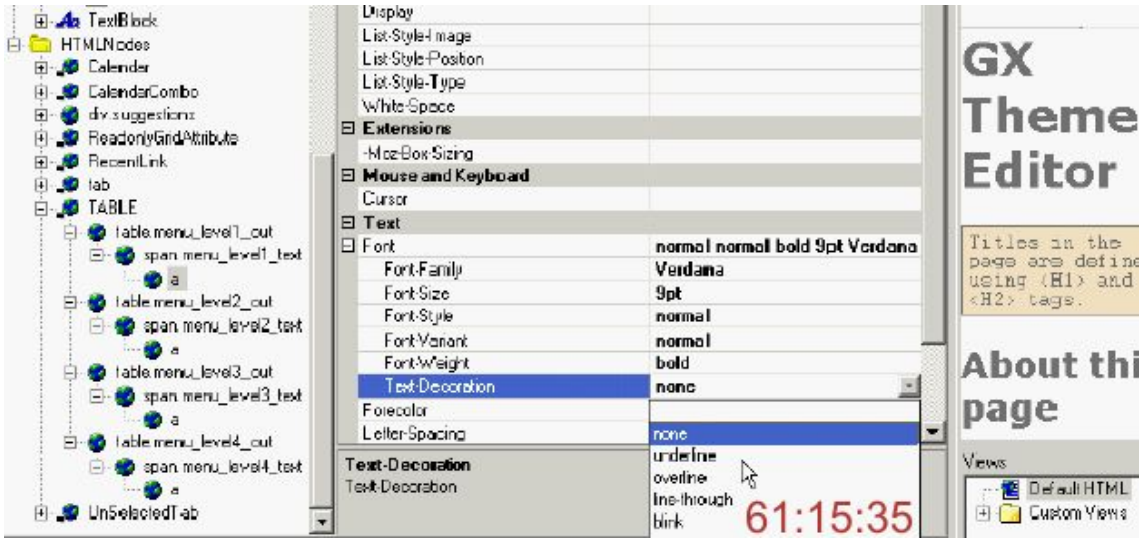
Tenemos que hacer una acotación más al tema este. Si ustedes programan un hipervínculo para cada opción de Menú, normalmente el navegador pretende poner sobre el Texto seleccionado un efectito de subrayado.

En el mismo ejemplo que estamos mostrando, si seleccionamos la opción “Trabajar con 4”:

vemos que en el Menú el Texto no ha quedado subrayado.

Para lograr esto hay un mecanismo que consiste en usar la sección HTMLNodes debajo de las Clases TextBlock:

y vamos a utilizar específicamente el nodo TABLE donde hemos definido la cadena de accesos para llegar al hipervínculo:



Aquí se define, para la estructura armada (que si tienen que ampliar lo pueden hacer de la misma manera) el formato de Tablas que se están generando para acceder al hipervínculo y cuando se define un Tag del tipo “a”, lo que se está definiendo es el comportamiento del Tag “anchor” (que es el Tag del hipervínculo para el navegador), allí se declara la Font y en particular en la propiedad “TextDecoration” que por defecto está en “underline”, lo que tenemos que hacer es ponerla en “none”.

Lo mismo hay que hacer para los demás niveles.

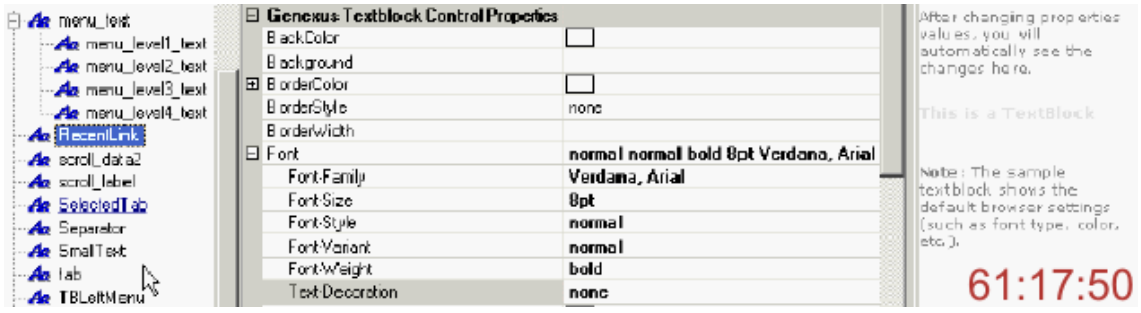
Y deben tener cuidado, porque el aspecto de la Font también es aplicado por el hipervínculo. Pueden manejar las dos opciones, la primera es optar por la Clase que define el color, tipo y tamaño de letra del Menú a nivel del Texto, que como dijimos, es lo que muestra la figura 61:11:02.

Pero este Texto siempre está metido bajo un hipervínculo, entonces podemos estar definiendo el aspecto del Texto a nivel de esta Clase por defecto, pero por arriba va a estar aplicando la Clase del hipervínculo, la del Tag “a”.

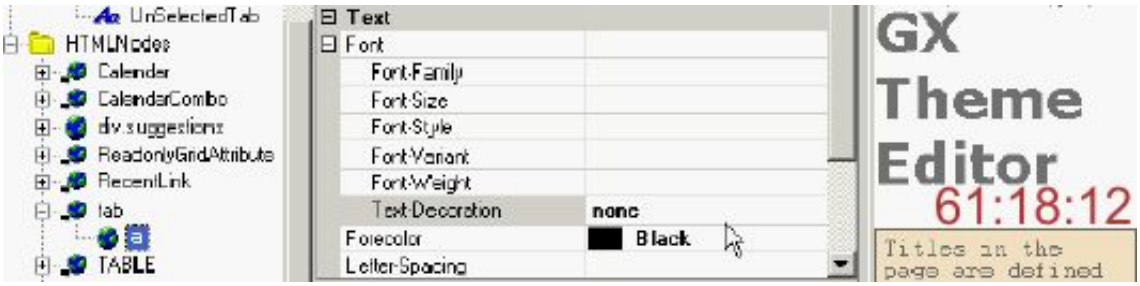
Lo que se hace normalmente es aplicar en las Clases de la menú_table los valores por defecto y si queremos diferenciar algo en el Tag “a” lo declaramos en la Clase de la figura anterior.

Lo que hay que tener presente es que la Clase que en última instancia comanda el aspecto de la Font es la del Tag “a” y al igual que podemos declarar “TextDecoration” en “none” también podríamos declarar en ella todo el resto de las propiedades de la sección “Text”, tal como se hizo en el ejemplo.

También tenemos entre las Clases del TextBlock la relativa al Recent Link:



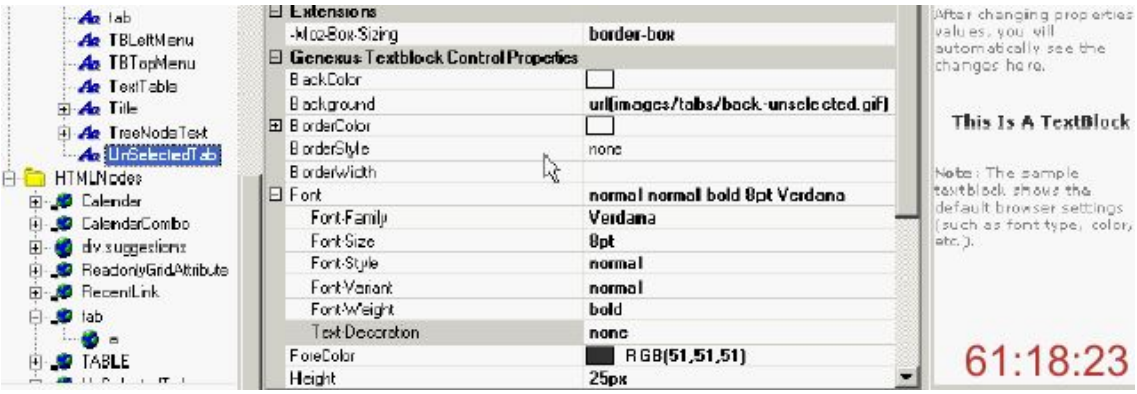
Lo otro que también tiene algo parecido a lo que acabamos de ver acerca de los hipervínculos del Menú, es el Tab:



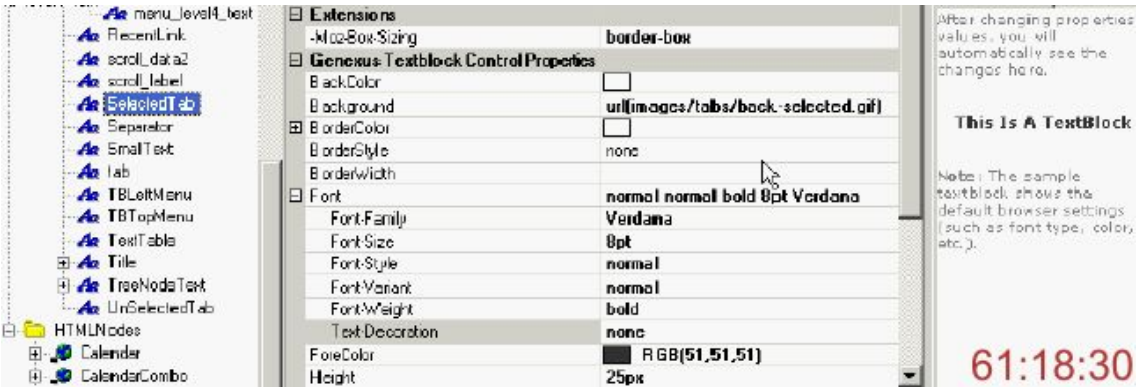
El Tab también tiene hipervínculos y también le aplica lo del efecto subrayado del navegador.

Para evitarlo debemos poner la propiedad “TextDecoration” en “none” como vemos en la figura.

Aquí no tiene definido el Font, pero sí está definido sobre la “UnSelectedTab”:



y el “SelectedTab” que está más arriba:



y en la próxima versión están unidas en una más general.

Lo importante es que acá pasa lo mismo que en los Menús: Como los Tabs tienen hipervínculos para activarlos, la Clase que en última instancia comanda el aspecto de la Font es la del Tag “a” y podríamos declarar en ella todo el aspecto del Font en la sección “Text” de la figura 61:18:12, aunque en este caso sólo tiene el “TextDecoration” en “none” y la que determina el aspecto de la Font es la Clase del TextBlock.

Me queda por aclarar el tema del “InvisibleAttribute” que vimos en la figura 60:30:26.

Entonces dijimos que en algún caso se requiere que el atributo esté en pantalla pero esté invisible y que esto no era lo mismo que la propiedad “.Visible”.

Imaginemos que tenemos que implementar un Prompt que está relacionado con una clave compuesta, por ejemplo en una Transacción: Tipo, Fecha y Número.

Y supongamos que ya sabemos que la Fecha siempre va a ser la del día, de modo que no hay que mostrarla en pantalla y convengamos además en que el Prompt está programado de modo que tenga que devolver los tres valores: Tipo, Fecha y Número.

Para que el Botón de Prompt funcione correctamente y cree el hipervínculo y aparezca la consabida “manito”, los elementos que están en los valores de Out de los Parámetros del Prompt tienen que estar en pantalla.

Si no están en pantalla no nos va a funcionar el Prompt y el inconveniente es que GeneXus no nos advierte con ningún Warning y recién nos vamos a dar cuenta del problema al probar la pantalla en tiempo de ejecución.

Cuando les pase que no aparece la “manito” sobre el Botón de Prompt, normalmente es porque entre los Parámetros de Out declarados en el Prompt, correspondientes a los parámetros que ustedes le están pasando en la invocación, alguna de esos elementos no está en pantalla.

Y no sólo tienen que estar en pantalla, sino que tienen que estar editables para que funcione el Prompt, por eso es que no nos sirve para esto la propiedad “.Visible”.

Porque si la ponen con la propiedad “.Visible” en cero, si bien en la pantalla está, como podemos comprobar accediendo al código HTML donde vamos a ver la declaración de la variable para que cuando se vuelva a cargar la pantalla se recargue su valor, pero no está editable y como dijimos, para que el Prompt funcione tiene que estar editable.

Para estos casos en que no queremos que alguna de estas variables se muestre en pantalla, la única alternativa es ponerla visible (visible real de GeneXus), pero aplicarle la Clase InvisibleAttribute.

En el ejemplo que pusimos la Fecha no debería ser editable, entonces la ponemos en pantalla y le asociamos la Clase InvisibleAttribute. Esto va a permitir que el Prompt funcione correctamente sin tener que poner la Fecha editable en pantalla.

Nosotros usamos mucho esta propiedad, especialmente para casos de Prompts con Parámetros de salida compuestos.

Otro caso que a veces nos ha pasado refiere no sólo a los Parámetros de Out sino a los de In. A veces también hay problemas con los Parámetros que se le pasan, porque si la variable no está en pantalla tampoco funciona bien el Prompt.

Hay casos en que ese Parámetro es muy afín a la pantalla y conocemos su valor, pero también tenemos que hacer lo mismo, definimos la variable invisible asociándole la Clase InvisibleAttribute para que no se modifique y la pasamos como Parámetro.

Vamos a entrar ahora en el último tema previsto para hoy.

Configs - Archivos V 6 01:22:57

Se trata de profundizar un poco en lo que son los Archivos Config que podría llegar a ser lo primero que tengan que hacer.

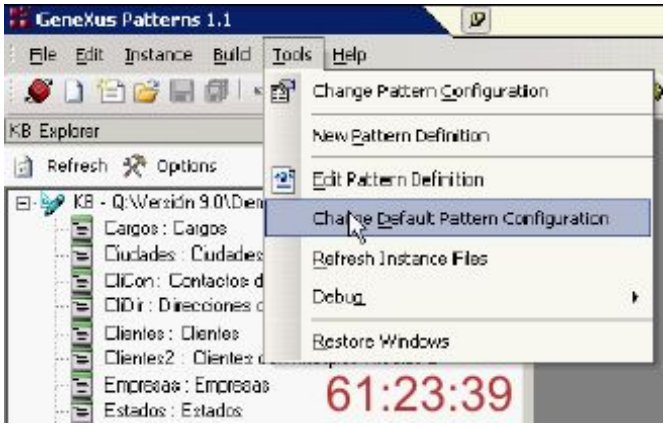
Cuando arrancan un sistema quizá lo que tengan que hacer es primero setear bien la configuración de los Archivos Config.

Si empiezan a trabajar sin hacerlo, bueno, está todo predefinido y pueden hacerlo rápidamente, pero ustedes verán si prefieren comenzar por ajustar esta configuración por defecto.

Default Config vs. Personalized V 6 01:23:31

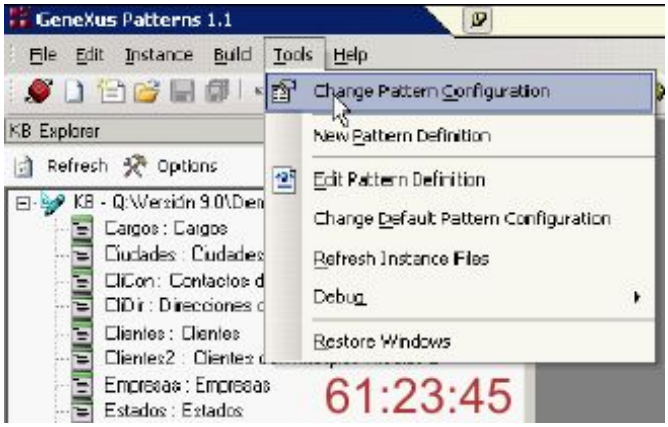
Ustedes pueden setear los Archivos Config de dos maneras:

- Entrando por el Menú superior Tools -> Change Default Pattern Configuration:



Para cambiar el seteo por defecto del archivo de configuración que está en el Directorio de Instalación del Pattern, de modo que el cambio es para cualquier KB que estén aplicando

- Entrando por el Menú superior Tools -> Change Pattern Configuration:

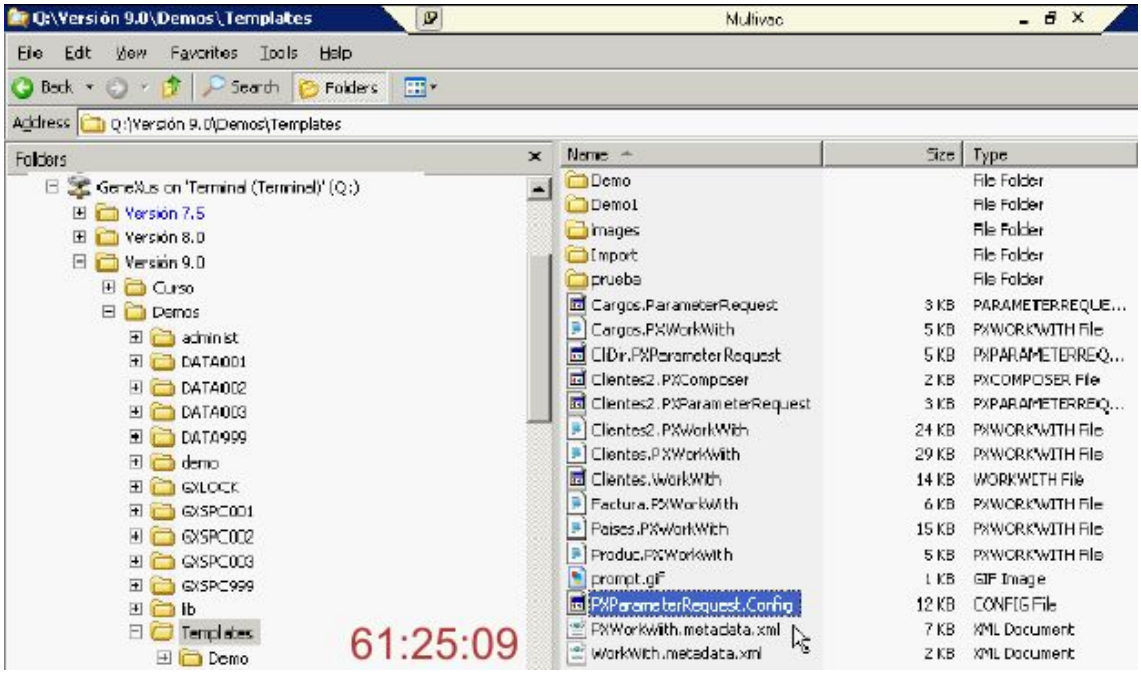


Les salta una ventana de alerta que les advierte que cuando pongan “Yes” se les va a copiar el archivo de configuración por

defecto a la KB en la que están trabajando, para que los cambios les queden locales a esa KB.

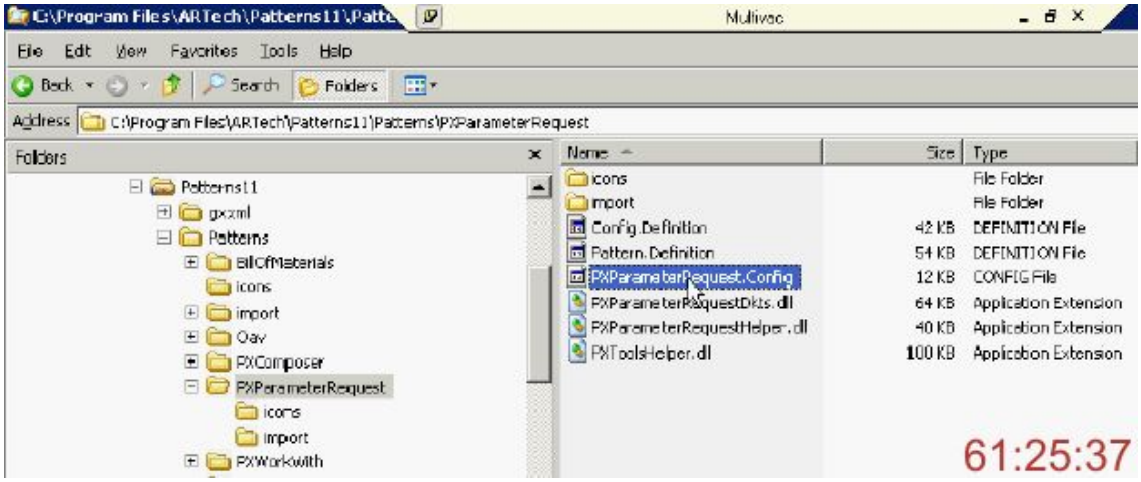
Entonces ustedes vean como lo quieren manejar, si se trata de un seteo que van a aplicar a todos los sistemas o es algún seteo puntual y personalizado para una KB en la que están trabajando.

Si optan por la segunda y presionan “Yes”, lo que el sistema está haciendo es copiando el archivo de configuración por defecto al Directorio “Templates”:



Aquí se acaba de copiar el PXParameterRequest.Config y basta borrarlo si quieren volver a aplicar el Archivo Config genérico que viene por defecto. De lo contrario, la próxima vez que accedan a través de esta opción ya no les salta la ventana de alerta porque este archivo queda local a la KB.

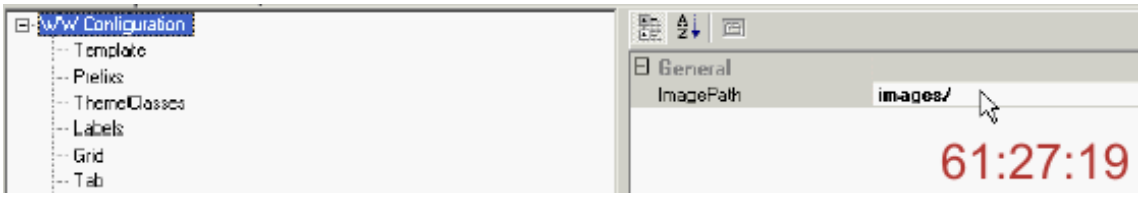
Si optan por la primera, ustedes directamente están modificando el Archivo Config genérico que está en el Directorio:



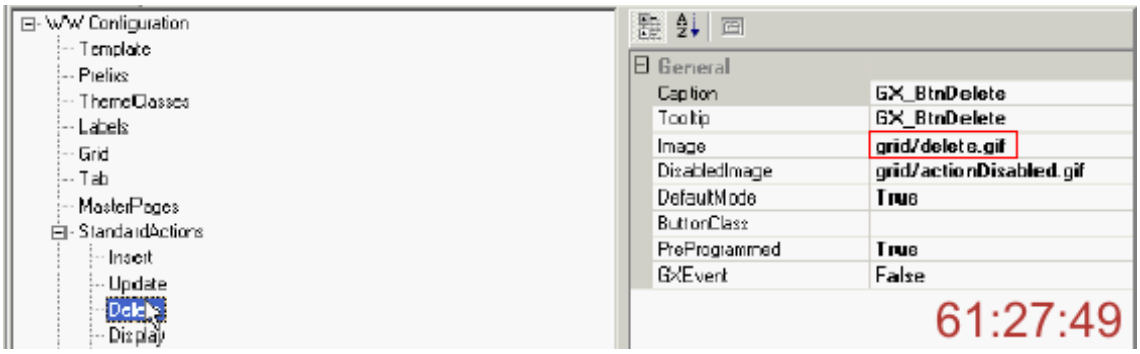
que es el Directorio de Instalación del Pattern.

Nosotros ahora vamos a entrar por esta primera opción para el Pattern PXWorkWith que es el más completo.

Los datos principales que se están manejando son los siguientes:



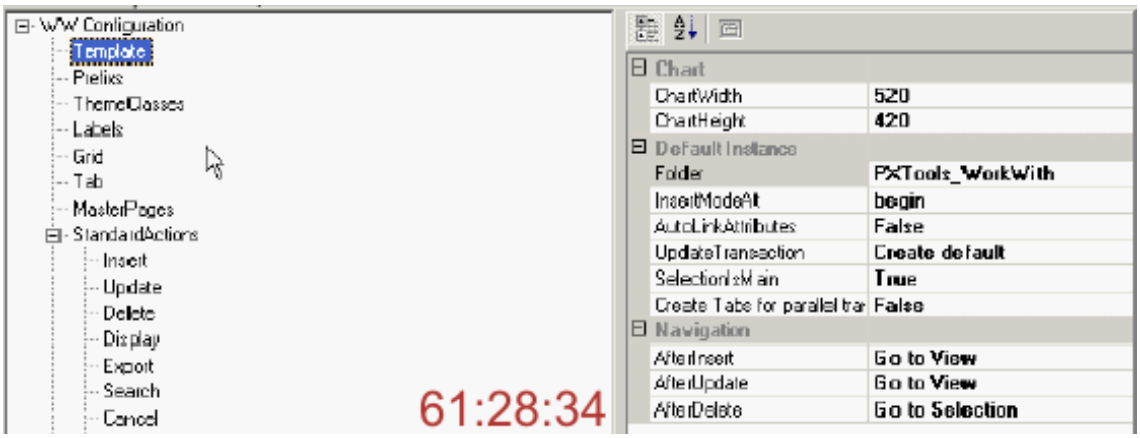
En “WWConfiguration” se setea el Directorio “images” que es el directorio de imágenes por defecto, que es el mismo que debería estar en el enumerado PXToolsParameters que vimos en V 5 | 01:31:35, lo único que en este caso se aplica para todas las imágenes que están definidas en el DefinedImages de la figura 20:41:22 y aquí están declaradas debajo del nodo StandardActions:



el Insert, el Update, el Delete (seleccionado), el Search, el Export, etc., porque la forma de referenciarlas está resuelta a través del Config.

Para cuando ustedes definan por ejemplo la Acción Delete, aquí es donde se declara donde hay que ir a buscar esa imagen (recuadro). Nosotros hemos declarado donde estaba el Directorio, pero el Pattern ubica la imagen gracias a esta propiedad del archivo de configuración.

Tratando de seguir el orden de la estructura, tenemos después el nodo “Template”:



que tiene el la sección Chart los valores por defecto para el armado de Gráficas y en las propiedades “ChartWidth” y en “ChartHeight” se declaran, en pixels, el ancho y el alto de la Gráfica.

En la sección “Default Instance”, en la propiedad “Folder”, se declara el Folder por defecto donde va a estar todo lo generado por el Pattern al que corresponda el Config (en este caso el PXWorkWith).

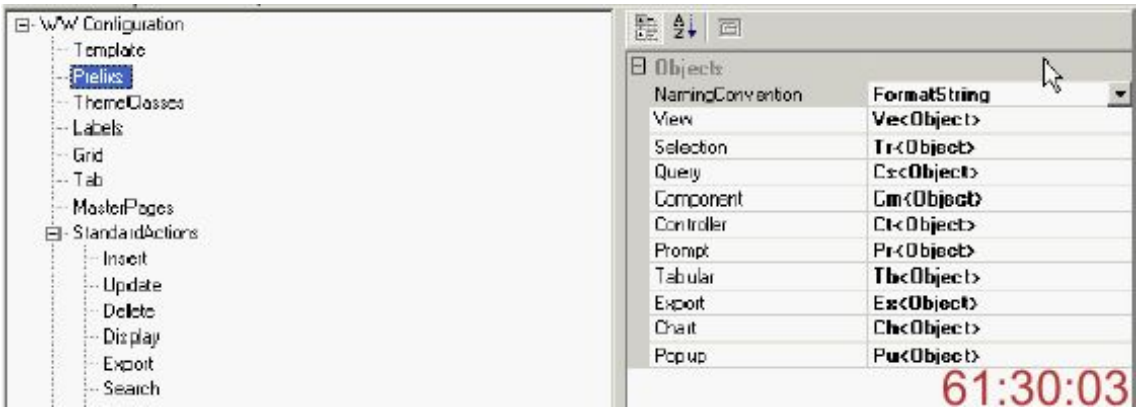
Recordemos que en el nodo raíz de cada Instancia podríamos indicar esto también con la propiedad “Folder” que aplica sólo a lo generado por la misma y habitualmente la dejamos en <default> para que aplique lo declarado aquí, en el Config del Pattern correspondiente.

Después comportamientos por defecto, incluso relativos a la navegación, como vemos en la sección “Navigation”, que pueden ir configurando conforme a los requerimientos del sistema a desarrollar.

Por ejemplo, la propiedad “SelectionIsMain” permite establecer que el Objeto “Tr” del Selection sea Main para poder compilarlo. También esto se puede declarar localmente en la Instancia, pero estos son los valores por defecto a setear.

Los comportamientos de control de secuencia que queramos declarar en las propiedades “AfterInsert”, “AfterUpdate” y “AfterDelete”, propiedades que ya hemos visto a nivel de la Instancia y que también habitualmente la dejamos en <default> para que aplique lo declarado aquí.

Después tenemos los Prefijos:



y acá básicamente se está definiendo toda la nomenclatura de Objetos del sistema, que ya hemos visto al analizar la figura 11:11:49.

Esta es una de las razones por las que puede ser necesario comenzar el desarrollo por los Archivos Config, porque si quieren cambiar el criterio con el que se resuelve la nomenclatura deberían hacerlo al principio.

Como ya hemos visto, en el caso del Selection, le va a poner un “Tr” (de Trabajar Con) seguido del nombre del Level. Donde dice <Object> en realidad va el nombre del Level.

Al nodo View le va a poner un prefijo “Ve” (de Ver), los tipo Query en el Selection “Cs” (de Consulta), a los Componentes “Cm”, a los Controladores “Ct”, al Prompt “Pr”, al Tabular “Tb” (de Tab), al Export a Excel, que son los procedimientos que se hacen específicamente para armar el Export a Excel, le pone “Ex” (de Export), a los de armado de Gráficas “Ch” (de Chart) y a los Popup “Pu”.

A propósito de este último, cuando se implementó toda la lógica de llamados a ventanas PopUps le agregamos al sistema la posibilidad de definir un prefijo específico para esas PopUps.

Esto porque cuando estamos trabajando con muchos programadores, es mejor poder determinar fácilmente, si un Objeto es una PopUp, el prefijo con el que otro programador tiene que invocarla dejando claro al primero que tiene que ser una PopUp.

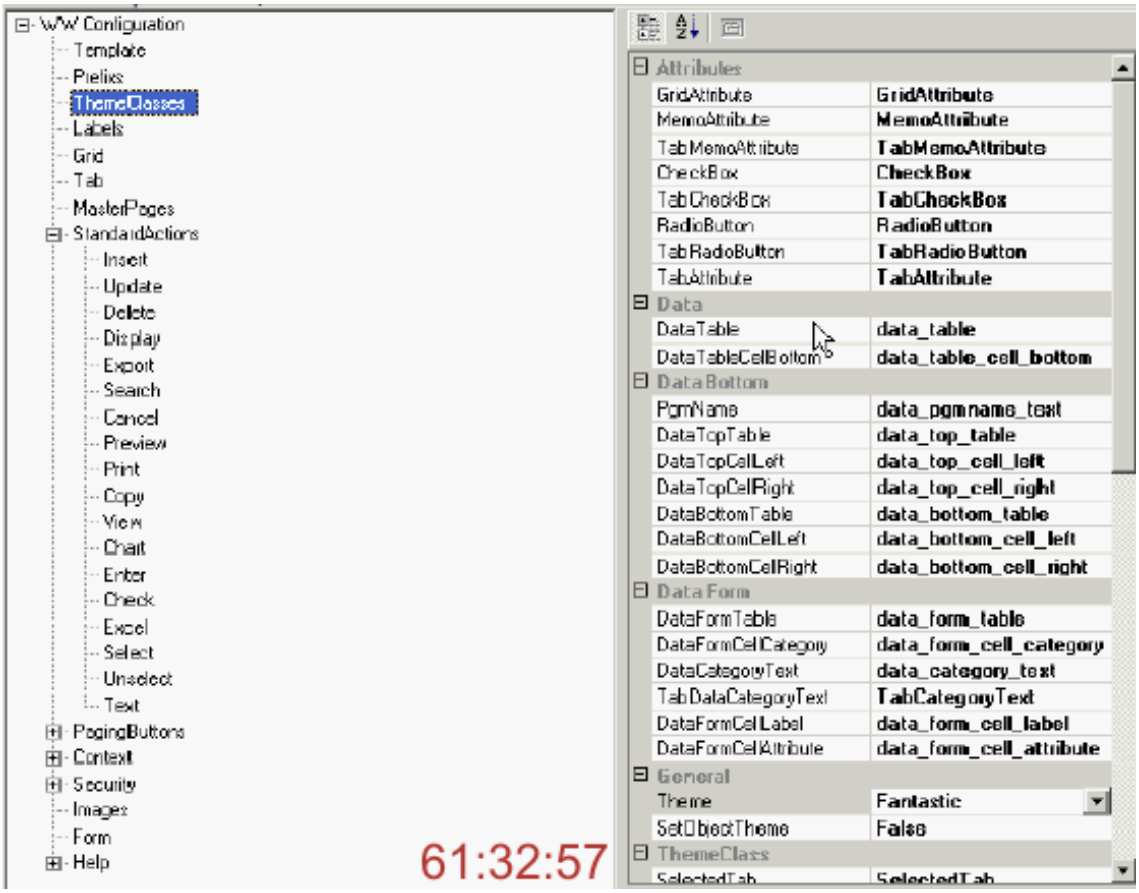
De modo que si tenemos que llamar a un Pu<Object>, ya sabemos que tenemos que hacer un Link con Target a “New”, en cambio si estamos llamando a un “Trabajar Con” común, tenemos que hacerlo con “Self”.

Si creen que esto los complica, no hay inconveniente en cambiar este prefijo “Pu” a “Tr” y listo. Van a quedar todos iguales.

Lo mismo puede ocurrir con las Consultas, que si les resulta complicado distinguirlas de los “Trabajar Con” comunes, basta cambiar el “Cs” por “Tr” y ya no es necesario diferenciarlas. Por esto decíamos que esto lo deben resolver al principio.

Después tenemos las Clases por defecto. Esto está muy relacionado con todo lo que estuvimos viendo acerca del Tema.

No está directamente hardcoded la relación, sino que todo lo que vimos está puesto a nivel del archivo de configuración:



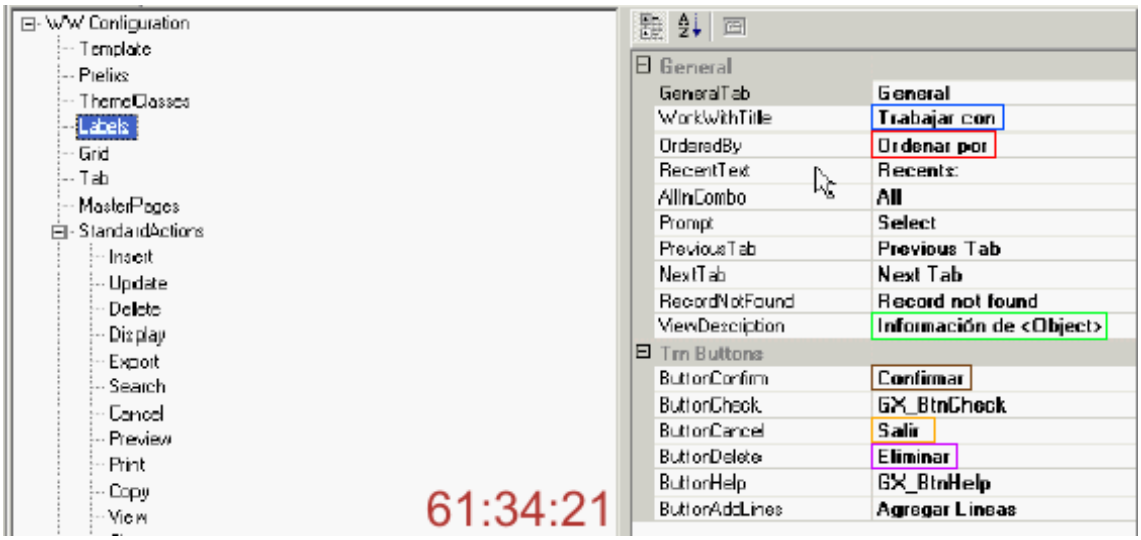
cada concepto está asociado a una Clase y el nombre de la Clase lo podríamos cambiar si no nos gusta, cambiándolo también aquí, en el Config.

Todo lo que el Pattern usa en el sistema, está todo declarado a nivel del Archivo Config.

Aquí está todo lo que vimos a nivel del Tema PXTools_Themes pero esto es específicamente lo que usa el sistema, acá no está lo que les mostramos acerca del ErrorAttribute, el SignificantAttribute y los estándares que definimos para que los puedan utilizar pero si no los quieren usar directamente los borran.

Por ejemplo en la figura anterior ven que están las Clases para el Atributo de la Grilla, Atributos de tipo Memo, los CheckBox, los RadioButton, acá está todo lo que está realmente relacionado y es utilizado por el Patrón.

Después tenemos los Labels por defecto del sistema:



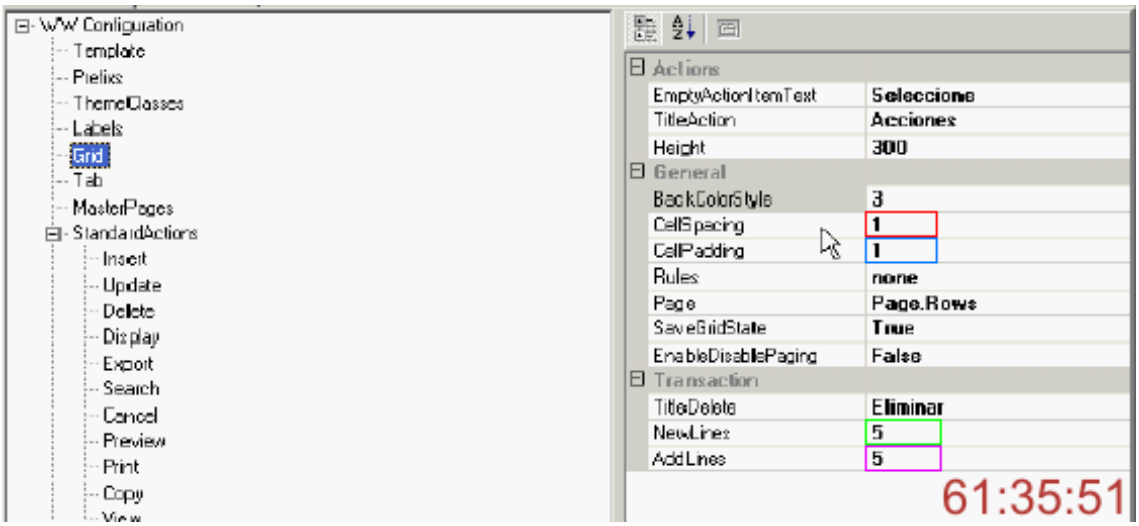
y si por ejemplo manejaran múltiples órdenes, les dijimos que les aparecía una Combito que nos permitía seleccionar los Caption de cada orden, bueno, el Label del OrderBy está definido acá (recuadro rojo). Si fuera necesario cambiarlo, lo hacen acá y ya se les va a cambiar en todos lados donde lo usen.

Son conceptos generales como es el caso del título “Trabajar con” (recuadro azul), allí va a decir “Trabajar con” y la descripción de la Transacción que están manejando, en este caso en español o, “Work with” si lo quieren en inglés.

“información de <Object>” (recuadro verde) se está aplicando en el View y allí va a decir “Información de” y la entidad que estén manejando, lo mismo que en la Transacción.

Y los Botones: El Botón de “Salir” (recuadro naranja) que normalmente está en los Prompts, donde automáticamente se les pone este Botón y los de las Transacciones, el de “Eliminar” (recuadro violeta), “Confirmar” (recuadro marrón), etc.

Después a nivel de la Grilla, tenemos el nodo “Grid” que vemos en la figura siguiente y es otro que tienen que tener en cuenta, especialmente para declarar el CellSpacing (recuadro rojo) y el CellPadding (recuadro azul) que son los más importantes.



Con estas dos propiedades ustedes definen el efecto que quieran lograr en la Grilla del “WorkWith”.

Se trata del espacio que vemos entre celdas y que estoy marcando con el puntero:

el que se define con el CellSpacing y el Cell Padding.

Este efecto de espaciado entre celdas no se puede resolver por Temas, o por las Clases de GeneXus (css) y hay que específicamente definirlo a nivel de estas propiedades de CellSpacing y CellPadding de la Tabla (que son atributos del HTML) que define el comportamiento de todos los elementos “hijos” de esta Tabla.

El CellSpacing determina el espesor de todas las líneas que forman la cuadrícula de la Tabla (en este caso de color gris que es el color de fondo) y el CellPadding determina el margen interno que va a haber alrededor de cada elemento que hayamos metido dentro de la Tabla, más allá de su alineación.

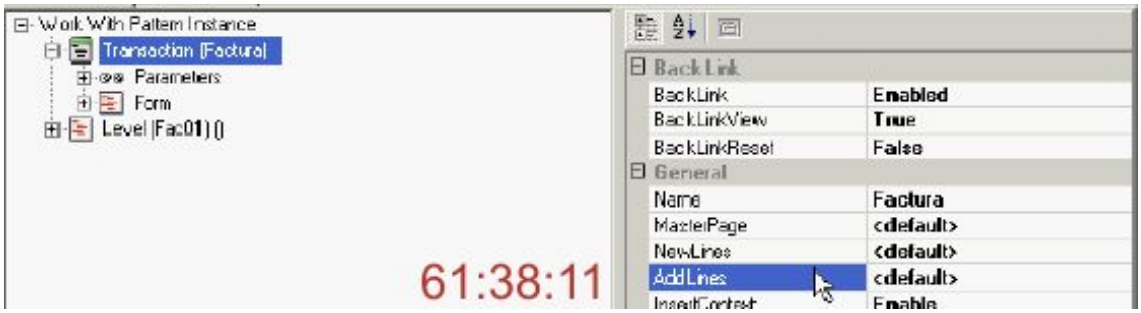
Estas dos cosas dependen de lo que se haya establecido en cuanto al diseño gráfico de la aplicación y si lo cambiamos después tenemos que regenerar todas las Instancias.

Esto, a diferencia de los cambios que puedan ocurrir a nivel de las propiedades de una Clase (css), que podemos cambiarla al final del desarrollo y su efecto es inmediato.

El NewLines y el AddLines (recuadros verde y violeta en la figura 61:35:51) son las dos propiedades que aplican cuando tenemos Transacciones de dos niveles, donde el segundo nivel es una Grilla que tiene todos los campos editables para poder ingresar los datos.

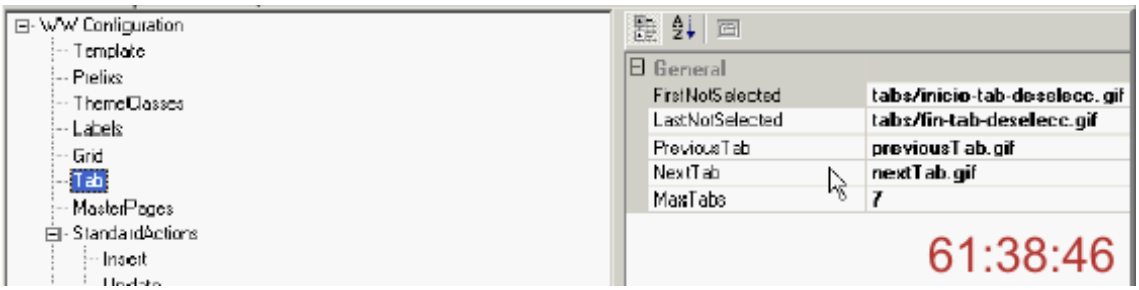
En NewLines se determina cuantas líneas por defecto van a tener para ingresar y el AddLines es un Botón que siempre tienen asociado a este tipo de Transacciones con dos niveles precisamente para agregar una cantidad de líneas que aquí se declaran, cuando el operador lo requiere.

Esto lo pueden configurar también a nivel de cada Transacción:



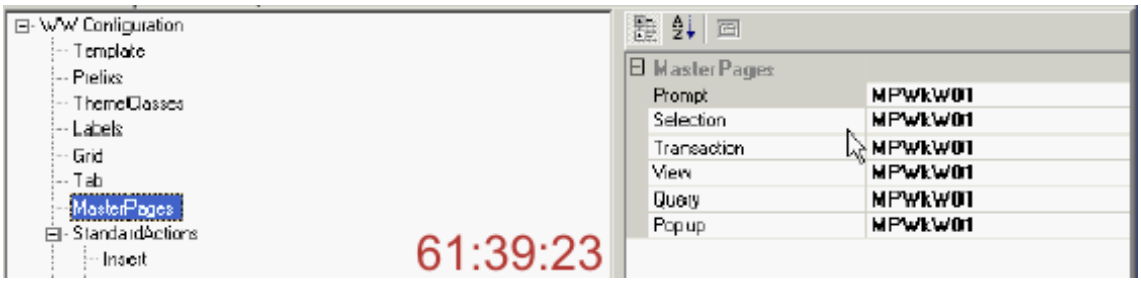
donde también tenemos las propiedades NewLines y AddLines para cambiar eventualmente para esta Transacción precisamente estos valores declarados por defecto.

Después tenemos el nodo “Tab”:



donde no aplican las dos primeras propiedades porque se está usando el enumerado que vimos en V 6 | 00:18:57 y sí aplican las propiedades PreviousTab, NextTab y MaxTabs donde se declara la máxima cantidad de Tabs antes de paginarlos, tal como vimos en esa misma oportunidad.

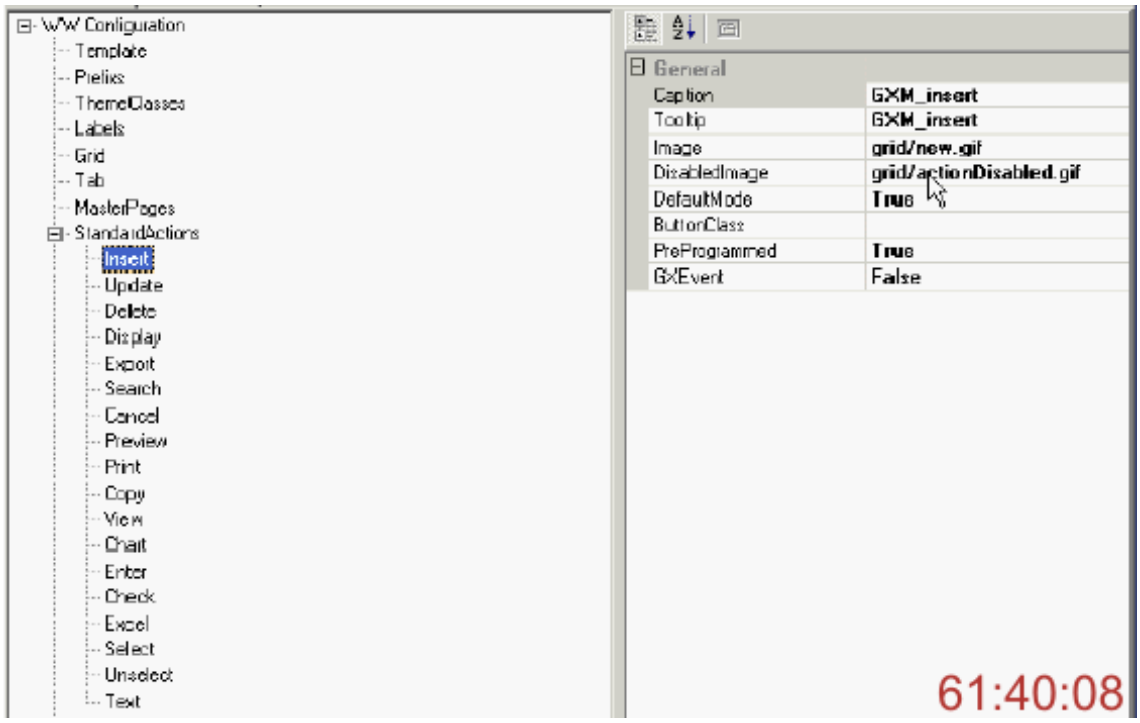
Por último tenemos este nodo “MasterPages” que antes tenía más utilidad, ahora no tanto porque ahora todos los tipos de Objetos se asocian a la misma Master Page, como puede verse en la figura siguiente:



pero podría haber un comportamiento diferencial por cada tipo, para cada concepto de nodo de la Instancia al que se le podrían asociar Master Pages distintas.

Hasta hace poco el Prompt tenía asociada una propia Master Page.

Después las “StandardActions” son todas las que se llaman DefinedImages en las Acciones de tipo Imagen y algunas más (las de los Modos):

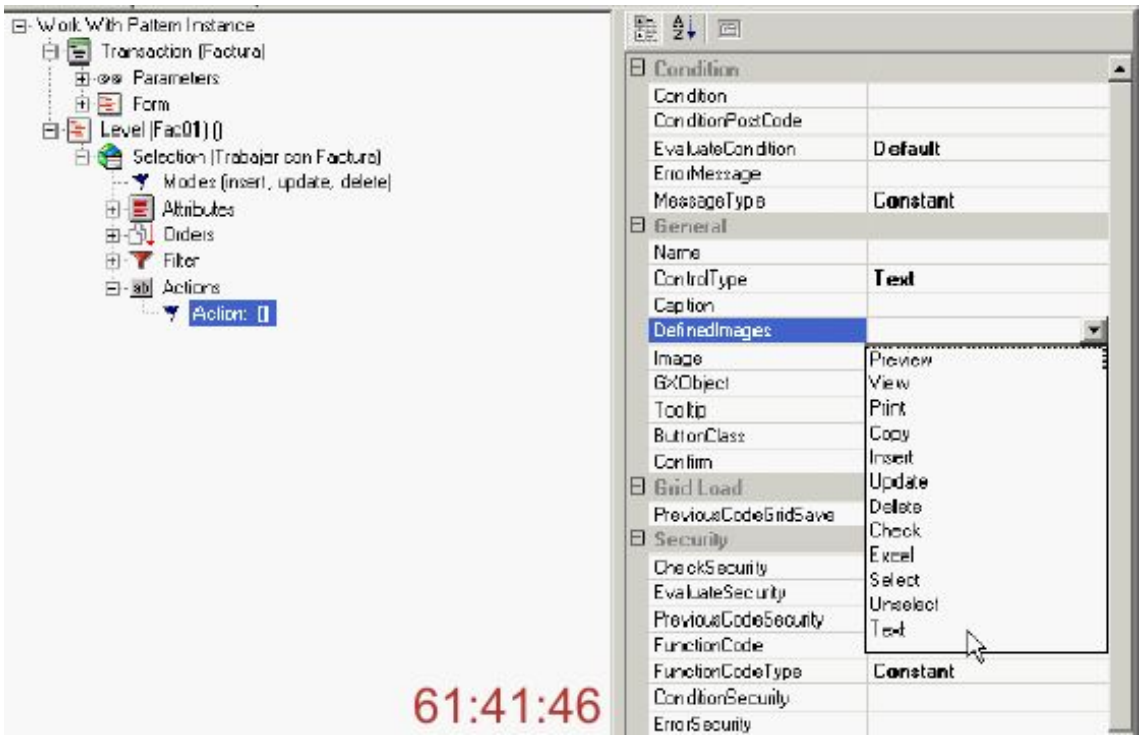


Aquí básicamente se apunta al repositorio de los directorios que les mostramos para cada una de estas Acciones.

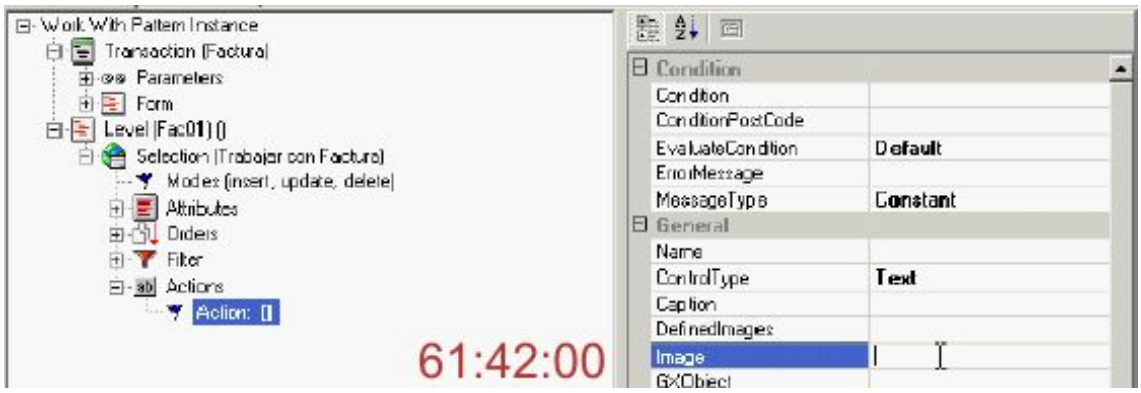
Si hubiera que agregar alguna el problema es que esto depende del generador, esto está medio hardcoded. Cuando se define una DefinedImage el sistema tiene que saber que debe ir a buscar esa DefinedImage y eso está programado por adentro del generador. Por tanto no les va a ser posible a ustedes agregar una DefinedImage, pero si se les ocurre algún otro tipo de Acción que no fue contemplado, será bienvenida la sugerencia de incorporarlo para generalizar más el concepto.

Justamente la idea es tener disponibles todas las Acciones que habitualmente usamos sin tener que estar continuamente referenciando a la imagen.

Porque cuando definimos una Acción de tipo imagen, siempre tenemos la posibilidad de referenciarle la imagen mediante la propiedad DefinedImage:



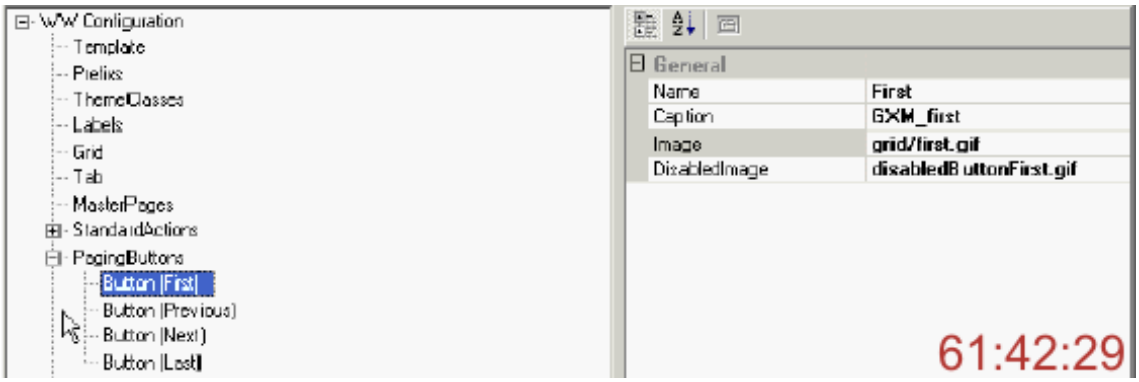
O usar la propiedad Image:



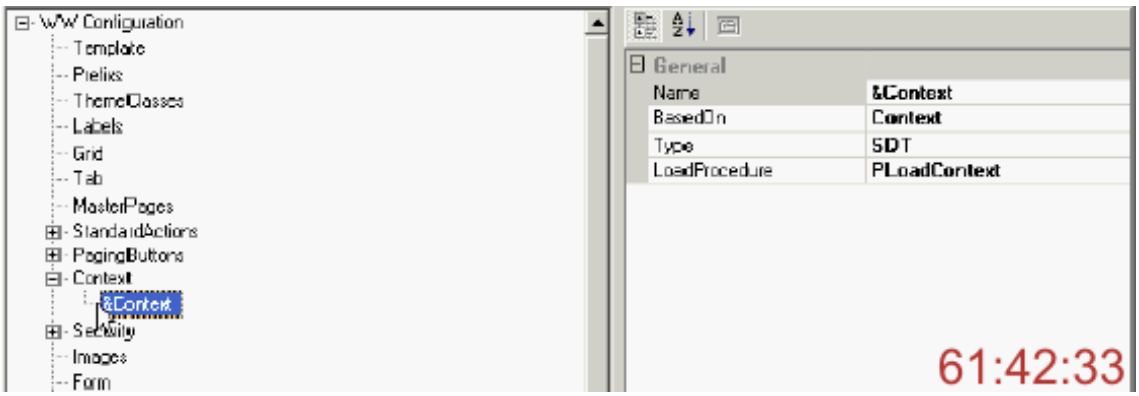
y aquí escribir directamente algo similar a lo que ponemos en el Config: grid/pepe.gif.

De lo que se trata es que si se justificara incluir esta imagen entre las DefinedImages ustedes nos avisen y nosotros lo implementemos para que no quede hardcoded esta referencia aquí. En principio creemos que las más comunes ya están contempladas.

Después tenemos las referencias a los cuatro Botones de paginación que vemos en la parte superior izquierda de la Grilla para acceder a la primera página, a la anterior, a la siguiente y a la última página de la misma:

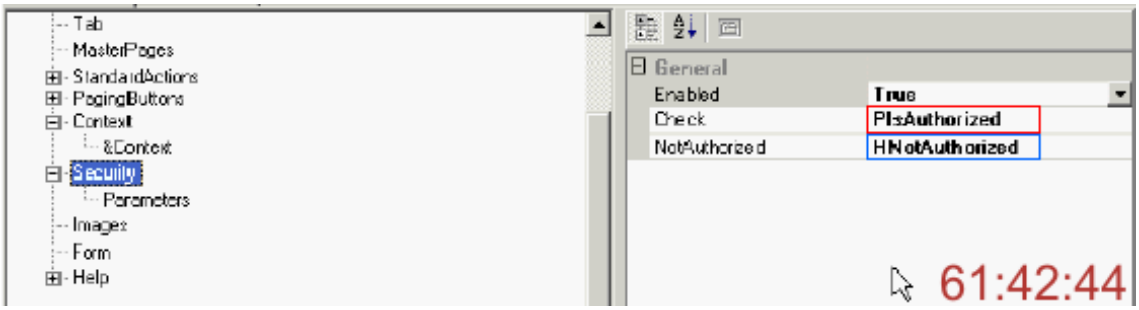


Después tenemos el nodo “Context”:



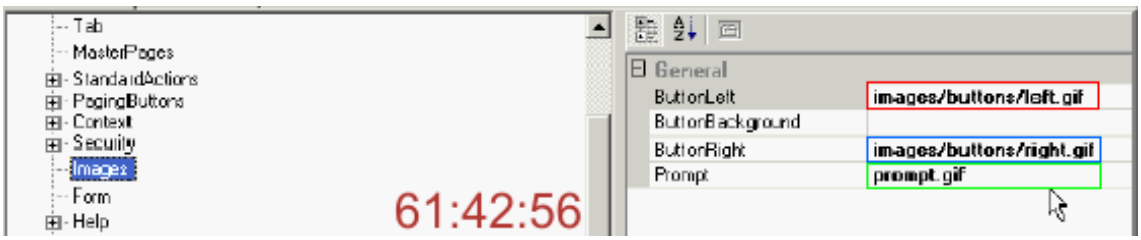
donde simplemente se indica la variable que usa el contexto y la función, esto no deberían cambiarlo ustedes.

Después en el nodo “Security”:



declaramos la rutina que se ocupa del control de seguridad (recuadro rojo) y el Web Panel del “NotAutorized” (recuadro azul).

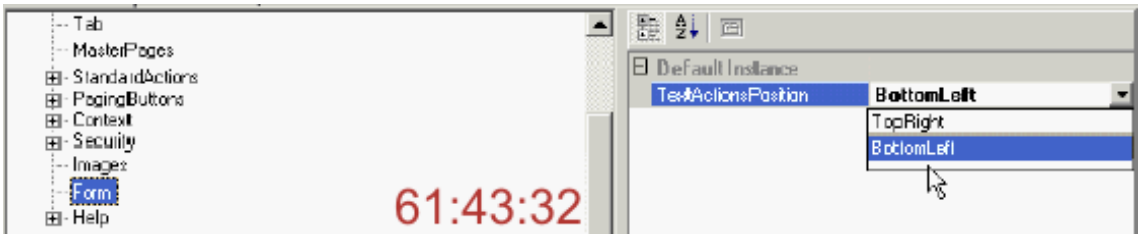
Después tenemos el nodo “Images”:



con las imágenes de los Botones, tanto la izquierda (recuadro rojo) como la derecha (recuadro azul).

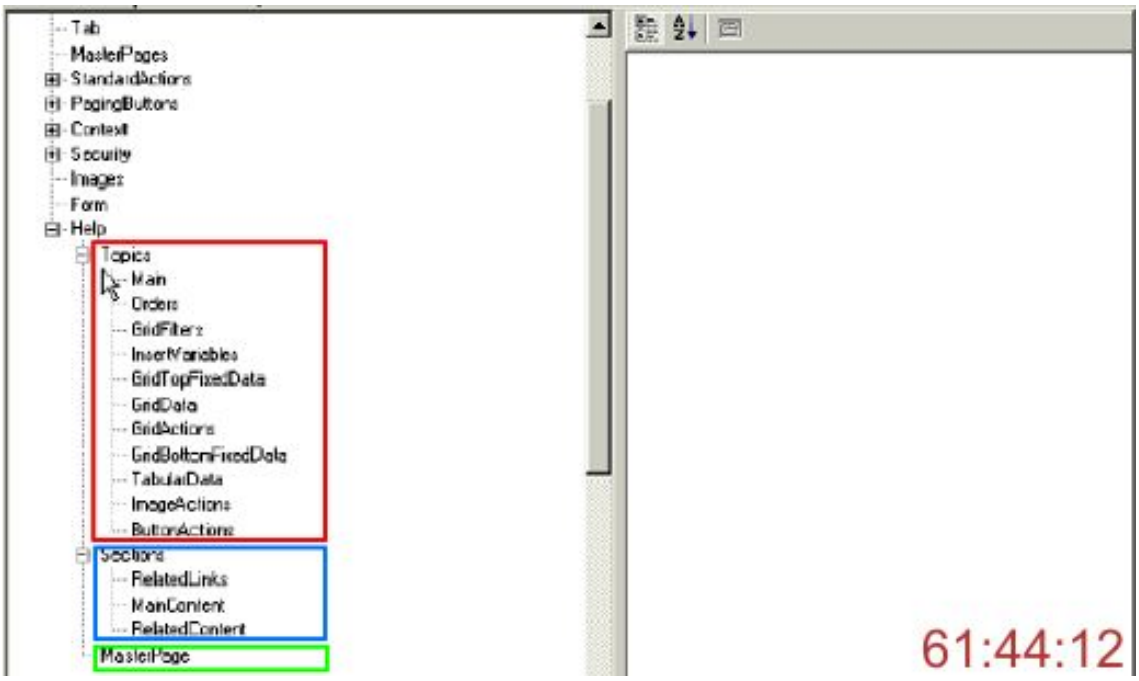
El “ButtonBackground” ya no aplica más y vamos a tener que sacarlo de acá porque aplica directamente la Clase del Botón, pero la izquierda y la derecha del Botón son imágenes que se ponen explícitamente y aquí se referencian, así como a la imagen del Prompt (recuadro verde).

Después tenemos el nodo “Form” que tiene un comportamiento que se fue mejorando para aumentar la flexibilidad del sistema respecto a la ubicación de los Botones de las Acciones de tipo Texto:



que en el “WorkWith” original iban arriba a la derecha, después se dio la opción de ponerlos abajo a la izquierda y si por requerimientos de la aplicación a desarrollar fuese necesario agregar alguna otra opción, nos informan y nosotros agregaremos la opción de ubicar estos botones donde sea necesario. Por ahora estas son las dos ubicaciones habilitadas.

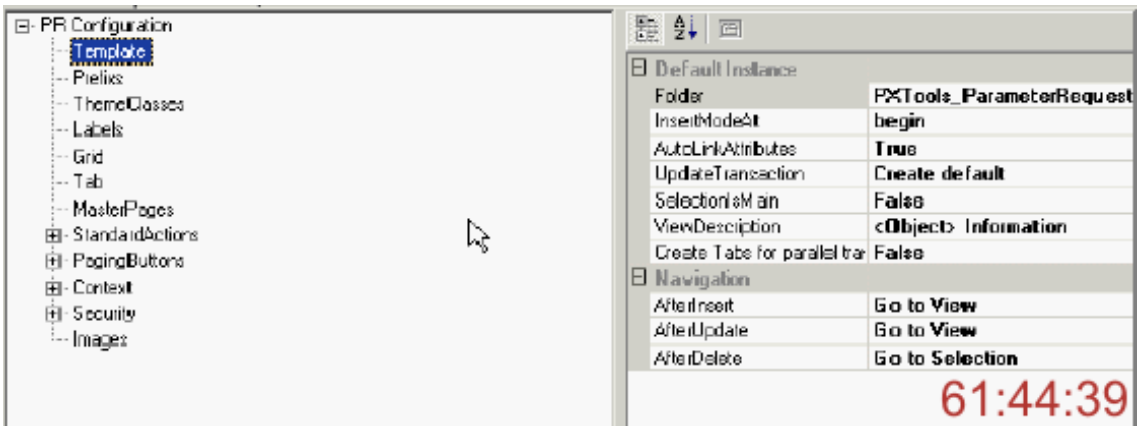
Por último tenemos el nodo “Help” que ya vimos al estudiar la Help Master Page en V 5 | 01:54:16:



Como dijimos entonces, aquí nosotros vamos a tener ya predefinidos los Tópicos que el sistema tiene soportados (recuadro rojo).

También tenemos las secciones principales de la ventana, que vimos en la figura 51:49:31 (recuadro azul) y la información relativa a la Help Master Page (recuadro verde).

Todo esto es a nivel del “WorkWith”, a nivel del “ParameterRequest” es muy parecido:

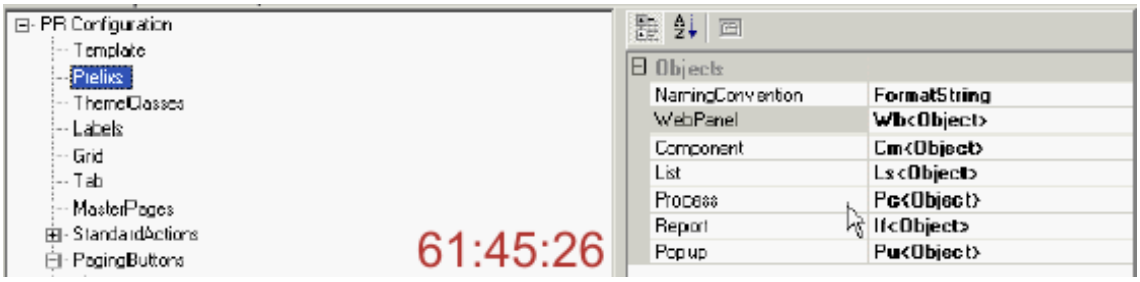


tenemos una lógica similar en la estructura del Archivo de Configuración.

En la sección “Default Instance” del nodo Template, por ejemplo, en la propiedad “Folder”, se declara el Folder por defecto donde va a estar todo lo generado por el “ParameterRequest”, en este caso. Y también para este patrón, en el nodo raíz de cualquiera de sus Instancias podríamos indicar este Folder con la propiedad “Folder” pero sólo para lo generado por la misma. Habitualmente a esta propiedad la dejamos en <default> para que aplique lo declarado aquí, en el Config del “ParameterRequest”.

También es similar la lógica en los Prefixs, las ThemeClasses, los Labels, el Grid porque también maneja Grillas, las StandardActions que se usan en el “ParameterRequest”, el soporte del paginado, en fin, algunas cosas las van a ver repetidas porque en realidad la tecnología Pattern no permite la posibilidad de manejar un Archivo de Configuración global para todos los Patterns. Cada Patrón tiene su Archivo de Configuración y no hay algo más arriba que eso.

El más importante quizás es el nodo de prefijos:



Acá hemos manejado prefijos más que por nodos reales de la Instancia, por conceptos de orden funcional, entonces WebPanel, Component y Popup sí, son tres elementos muy concretos pero después según se trate de “ParameterRequest” que van a llamar a un Listado o que van a llamar a un Proceso o que van a un Reporte, incorporan los prefijos “Ls”, “Pc” e “If” como estándar de nomenclatura que no es obligatorio seguir y si quieren ponerle a

todo “Wb” o el prefijo que prefieran para que represente un “ParameterRequest”, el momento de hacerlo es al principio para que cuando tengan que invocar a Objetos GeneXus, ya sepan que prefijos van a tener.

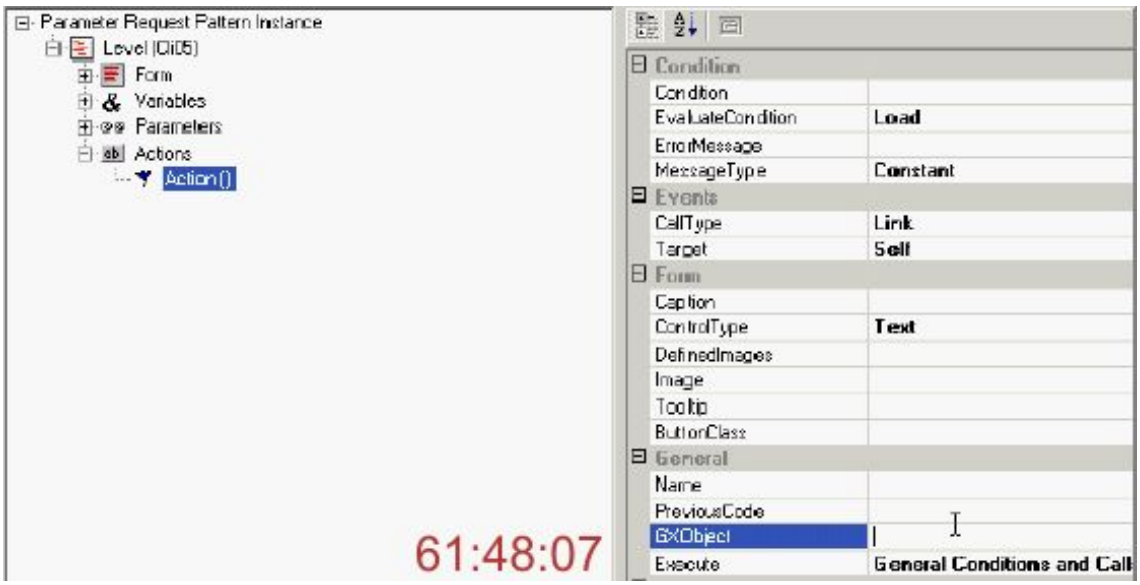
Por último, el Archivo de Configuración del Composer no tiene ninguna diferencia con el “ParameterRequest”, es exactamente igual.

Quizás el asunto más importante respecto a los Config de los tres patrones es el tema de los prefijos, que debería ser lo primero que resuelvan a nivel de los aplicativos del sistema porque una vez que empiecen a programar las invocaciones a los Objetos ya van con todo incluido.

Por este motivo en algún momento pensamos en la posibilidad de, en lugar de hacer referencia a Objetos GeneXus, hacer referencia a Instancias directamente y así cubrir el caso de cambios de prefijos pero por ahora la referencia es a Objetos GeneXus. Si un “ParameterRequest” tiene que llamar a un “WorkWith” tiene que llamar al Objeto GeneXus generado por el “WorkWith”.

Otra posibilidad en la que hemos pensado sería poder llamar al Level “Tal” del nodo Selection de la Instancia “Cual”. Entonces si mañana cambia el prefijo se cambiaría automáticamente en los dos, en el Objeto y en su referencia. Pero por ahora eso no está implementado por lo que hay que tener mucho cuidado al momento de resolver los prefijos porque después un cambio en los mismos les obliga modificar en todos los lugares desde donde se llama a los Objetos que cambiaron su prefijo.

Un detalle que les puede servir si les pasa eso, cuando definimos internamente la llamada de una Acción, por ejemplo a un Objeto GeneXus:



todos los campos que son referidos a entidades de GeneXus como es el caso del “GXObject”, como es el caso de un Atributo, en realidad lo que se guarda a nivel de la Instancia es la referencia interna.

Vieron que GeneXus cuando crea un Objeto, cualquier cosa, una Transacción, un Work Panel, internamente en realidad tiene una “Id”, un número de Id interno y lo que se graba en la Instancia es este número de Id interno.

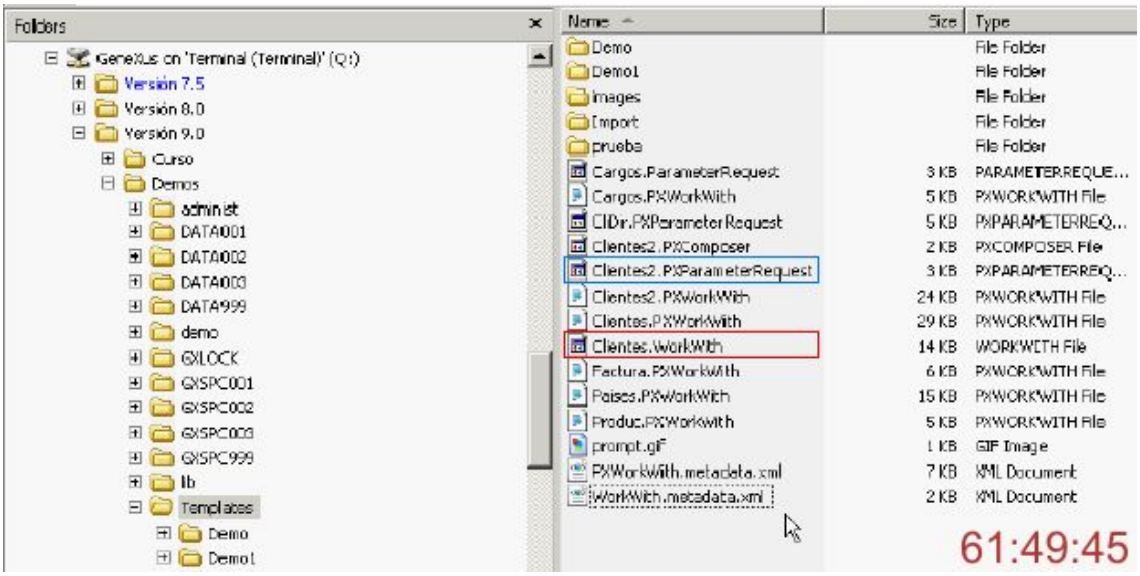
Esto lo que permite es que si mañana ustedes, desde GeneXus renombran el Objeto se les va a trasladar eso a la Instancia.

Si tuvieran algún problema del tipo de los que vimos con relación al cambio de los prefijos, es recomendable que en vez de renombrar el Objeto a nivel de la Instancia lo hagan a nivel de GeneXus porque ya les va a quedar automáticamente renombrado a nivel de la Instancia.

Y el otro punto que les quería mencionar es el tema del manejo de Instancias a nivel de la KB por este mismo concepto.

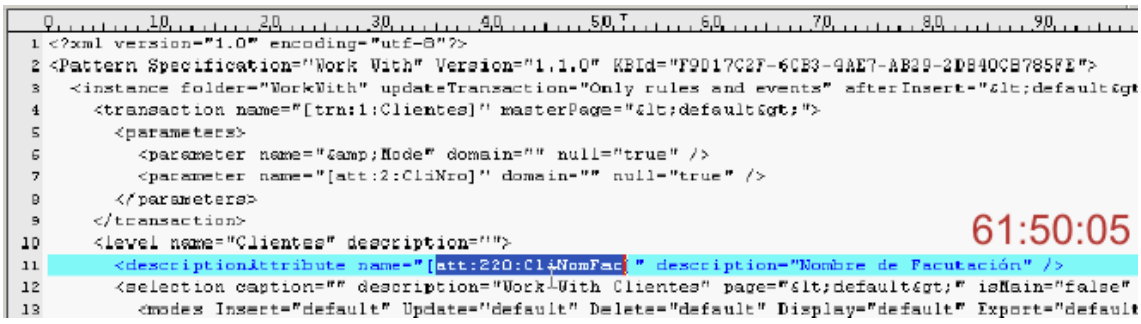
Como cada KB tiene su "Id" de su Objeto y como en la mayoría de los casos van a trabajar en modo distribuido, esto es, con KB distribuidas donde cada uno tiene su KB, puede ser que para cada Objeto que se les consolide (al distribuir el Objeto para otro lado), el "Id" que se consolida en una KB no sea el mismo que el de la otra.

Entonces como justamente les acabo de mencionar que lo que queda grabado dentro de la Instancia es el "Id" del Objeto, si ustedes una Instancia la copian desde el lado del Template, que es donde está grabada la Instancia:



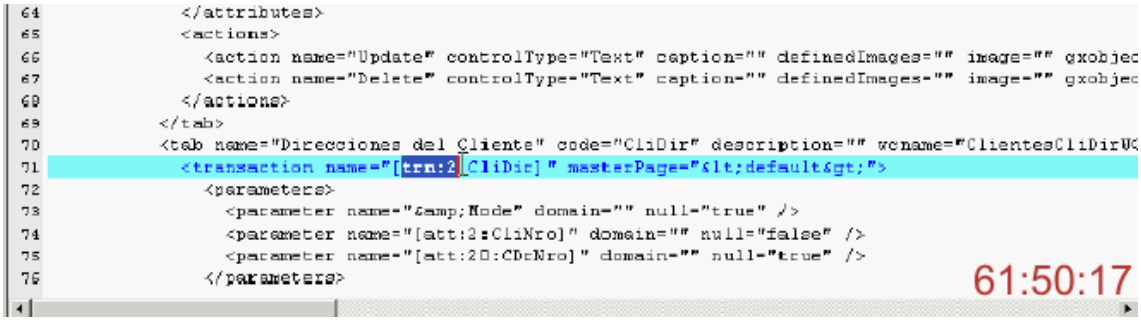
(en el mismo directorio que vimos en la figura 61:25:09, donde también se copian las Instancias con la extensión del nombre del patrón al que pertenecen), entonces pueden tener problemas porque el "Id" de los Objetos referenciados que estén grabados en esa Instancia no van a corresponder con los "Id" de esos Objetos en la otra KB.

Si en el ejemplo anterior abrimos con el UltraEdit la Instancia "Clientes.WorkWith" (recuadro rojo), dado que se trata de un archivo XML:



ven que aquí refiere al “att:220”, este es el “Id” del Atributo, seguido de “CliNomFac” que es el nombre externo del Atributo.

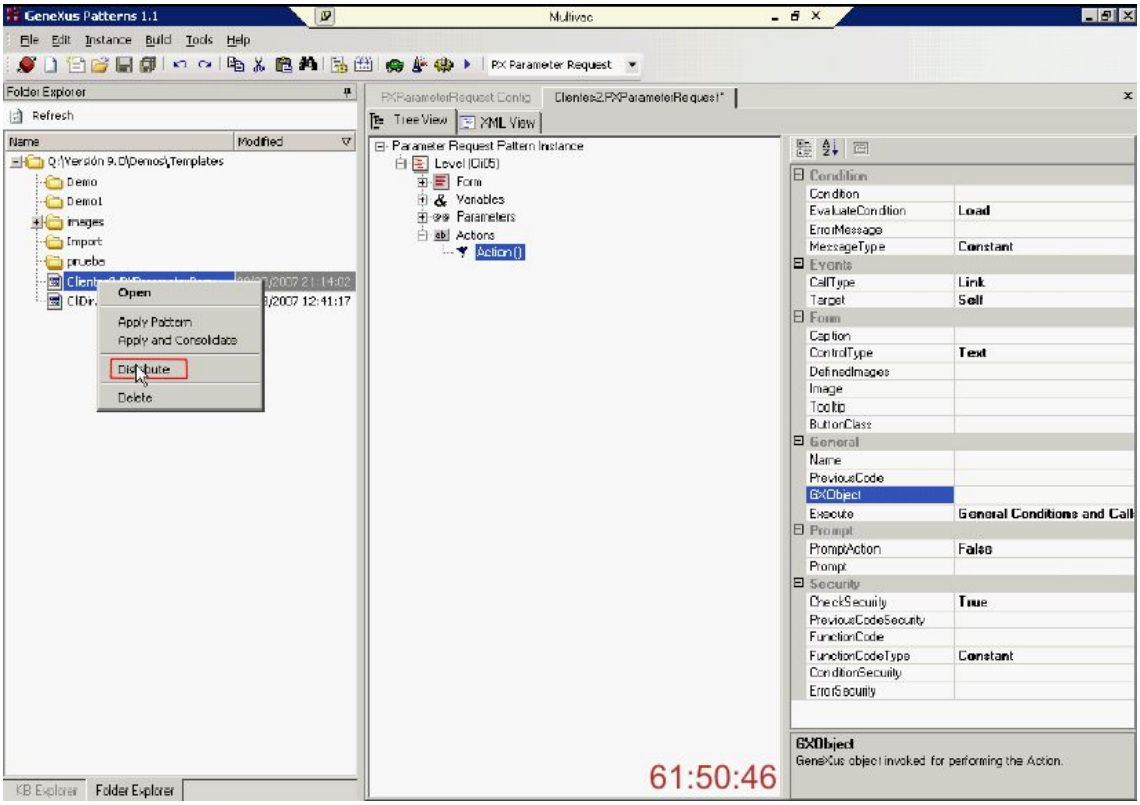
Esto también pasa con los nombres de Objetos GeneXus:



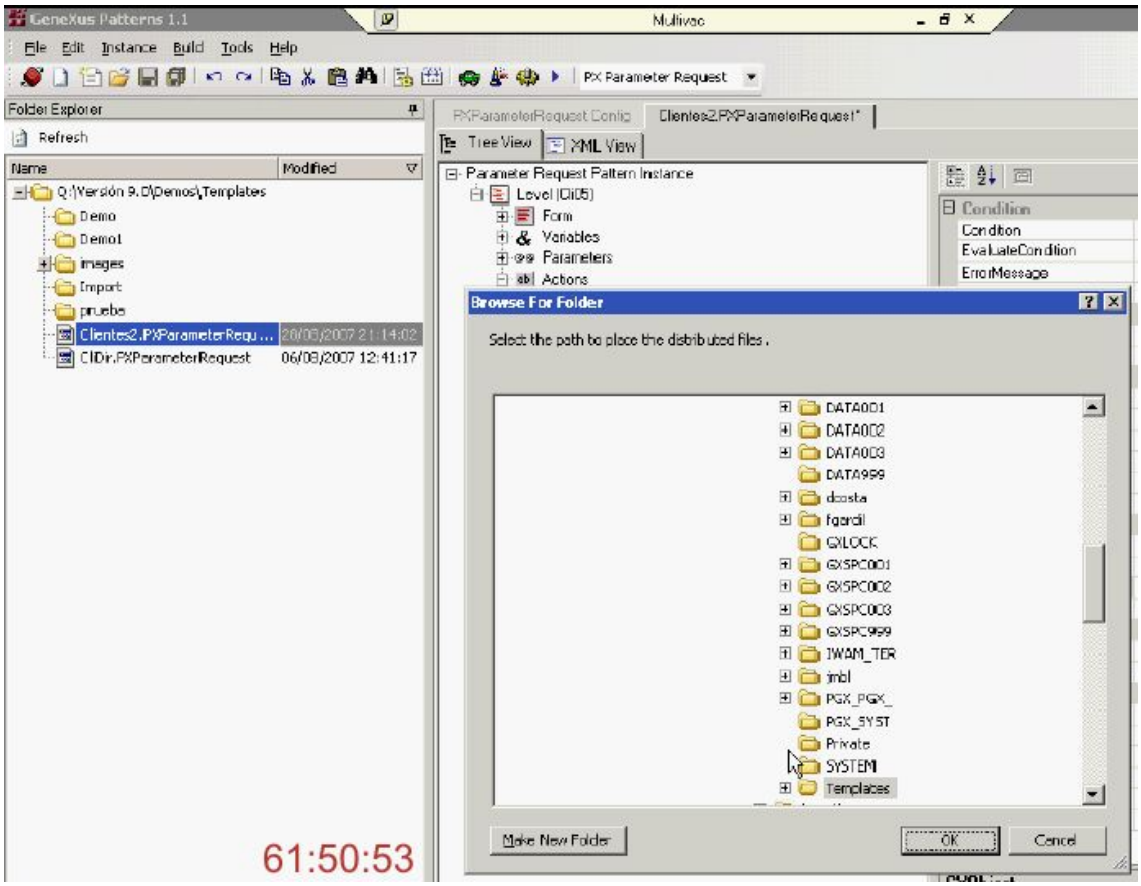
aquí refiere a “trn:2” que es el “Id” de la Transacción.

Y el problema es que estos “Id” dependen de la KB en la que estemos trabajando. Por lo tanto, para evitar este inconveniente, este tipo de distribuciones de Instancias se deben hacer siempre desde GeneXus Patterns.

Si en el ejemplo que veíamos en la figura 61:48:07 hacemos botón derecho sobre la Instancia “Clientes2.PXParameterRequest” (que es la misma recuadrada en azul en la figura 61:49:45):



podemos acceder a la opción “Distribute” (recuadro) de GeneXus Patterns que lo que hace es abrir una ventana de exploración de directorios para indicar el Path al directorio donde quieren moverla:



y al momento de distribuir esta Instancia a ese otro lugar se le quitan todos los identificadores internos para que cuando el Pattern de la otra KB vaya a abrir esa Instancia, en ese momento se regenere la carga de los identificadores.

Esto lo hace Artech por un tema de optimización para no tener que estar accediendo continuamente a GXPublic, es mucho más rápido identificar las referencias por el “Id” interno, sin tener que primero ir a buscar por el nombre para después obtener el “Id” que se requiere para trabajar con algún elemento subordinado a la Tabla o al Atributo que se está referenciando.

Esto es a nivel de GXPublic, pero esta ventaja nos genera ciertas desventajas al momento de tener que manipular las Instancias y por eso hicieron ellos esta opción de “Distribute”.

Hay que acordase entonces de siempre hacerlo así, porque si no, en caso de mover la instancia directamente a otra KB, al momento de abrir la Instancia en la nueva KB les va a aparecer un mensaje con algo así como: “Error: Identificador no concuerda con el nombre del Objeto”, a pesar de lo cual el Pattern va a tratar de recargarlos. Esto por lo general funciona bien y los recarga pero en algunos casos esto no anda bien y queda mal la Instancia.