deLarrobla & Asociados

Manual de Estándares de Programación

Número de Documento: BT-XXX-XXX-00
Fecha de Última Revisión: mayo de 2013

Fecha de Última Revisión (**mayo de 2013**)

Esta edición aplica a Bantotal Versión 2.1

Un Formulario de Evaluación es provisto al final del Manual para comentarios del lector.
Los comentarios pueden ser dirigidos a:

documentacion@dlya.com.uy

de Larrobla & Asociados Internacional
Zonamérica Business & Technology Park
Edificio Synergia, Of. 002
Ruta 8 Km. 17.500
Montevideo – Uruguay
América del Sur

Resumen

En este manual se describen estándares aplicables al desarrollo y mantenimiento de programación (fuentes). Está dirigido a técnicos. Alcanza normas en la escritura de un fuente Genexus y su entorno. La aplicación de estos estándares es obligatoria desde la aplicación del presente documento.

(29 páginas)

Tabla de contenido

Información Relevante

Los siguientes términos son marcas registradas de de Larrobla & Asociados en Uruguay y/o otros países:

- Bantotal
- Bancaria
- Comex
- Fondos

Los siguientes términos son marcas registradas de otras compañías:

AS/400, OS/400 son marcas registradas de International Business Machines Corporation.

Windows, Windows NT, Word and Excel son marcas registradas de Microsoft Corporation.

GeneXus, GXplorer son marcas registradas de Artech – Advanced Research & Technology.

Otras marcas registradas son marcas registradas de sus respectivas compañías.

Prefacio

El objetivo principal es establecer estándares aplicables al desarrollo de aplicaciones, a utilizar por la totalidad de los programadores de todos los sectores.

En una primera etapa se cubrirán aspectos de programación, y se pretende que un programa pueda ser fácilmente interpretado por cualquier persona con conocimientos básicos de la aplicación.

Organización del Manual

Este manual está organizado en 9 capítulos:

Capítulo 1, "Consideraciones Generales"
En este capítulo, se hace referencia a la utilización de Nomenclaturas en la programación, ver *Manual de Nomenclaturas de Programación*.

Capítulo 2, "Código de los programas fuentes"
En este capítulo, se establecen normas sobre cómo debe hacerse la escritura del código principal de un programa, sus sentencias, utilización de variables, subrutinas, documentación del fuente.

Capítulo 3, "Parámetros estándares recibidos"
En este capítulo se describe el uso de parámetros que aplican a cierto tipo general de programas.

Capítulo 4, "Rutinas Generales"
En este capítulo se hace referencia a la conveniencia de la utilización de rutinas generales y se citan casos especiales aplicados especialmente en *Bantotal*.

Capítulo 5, "Métodos de Control"
Se describe un método de control, que proporciona un elemento más para saber si un cliente está utilizando la versión correcta del programa. Se sugiere la utilización de "debugs" para controlar posibles errores reportados por el cliente.

Capítulo 6, "Definición de transacciones GX y su compilación"
Se hace referencia a la conveniencia de tener una utilización controlada de transacciones *Genexus* ejecutables.

Capítulo 7, "Auditorias de Modificaciones en tablas"
Aplicación para desarrollos en el entorno de modelos Java.

Capítulo 8, "Multi Idioma"
Describe el procedimiento para escribir código con posterior lectura en otros idiomas (Java, C#).

Capítulo 9, "Controles Obligatorios"

Publicaciones Relacionadas

Las publicaciones listadas a continuación son referenciadas a efectos de obtener una mayor información sobre los tópicos incluidos en este documento:

Número de Documento	Descripción
	Bantotal – Manual de Paralelización de Procesos.
	Bantotal – Manual de Nomenclaturas de Programación

1.0 Consideraciones Generales

1.1 Nomenclatura para objetos GeneXus

Ver documento relacionado: *Manual de Nomenclaturas de Programación.*

1.2 Uso de subtipos y atributos existentes

No se permite el uso de subtipos con atributos referidos a alguno que forma parte de la nómina de atributos de *otro Rango* al que se está trabajando.

Es decir, se permite definir un subtipo entre dos atributos declarados dentro del mismo conjunto de tablas y/o programas que se estén desarrollando referidos a un tema:

Por Ej. :

Un nuevo Atributo BRD002Cta subtipo de CtNro (atributo ya existente en Bantotal) clave de la tabla FSD008 ➔ **NO PERMITIDO**

Atributo BRE002Cta subtipo de BRD001Cta (ambos recién definidos) está permitido.

Se hace extensivo el uso de las *nomenclaturas* para denominar a los *subtipos*. En el caso del ejemplo anterior BRD001SubCta para el subtipo, y en el caso de definirlos como **SUBTIPO GRUPO**, aunque estén involucrados sólo un par de atributos. Nunca definir subtipos que queden en el grupo *NONE.

➔ **Se prohíben las definiciones de subtipos**:

- que signifiquen reorganizaciones (creación de nuevos índices) de tablas ya existentes, o
- que obliguen a que una transacción comience a navegar por una nueva tabla para mantener su integridad referencial, en el caso de la modalidad ‘delete’.

Por ejemplo:

Si se define el atributo BRE002Mon subtipo del atributo Moneda (Clave de la tabla FST005), obliga a recompilar la transacción TTRT005, por cambiar su navegación. ➔ NO PERMITIDO.

En este caso la transacción TTRT005, al intentar hacer un ‘delete’ de un registro en la tabla que define: FST005, hará un control de integridad de existencia de registros de la tabla en la que está el atributo BRE002Mon.

➔ **Asimismo, se prohíben las definiciones de tablas utilizando atributos de tablas Bantotal ya existentes en una aplicación distinta a Bantotal Coree.**

1.3 Medir el impacto del cambio

Al momento de modificar / crear un programa, siempre se debe evaluar la posibilidad de que lo que se esté resolviendo sea útil para otros clientes o para otros casos del mismo cliente. Es decir, antes de programar la solución debe tenerse claro cuan abierta (parametrizable) debe ser, siempre buscando que sea lo más adaptable mediante parámetros posible.

Ejemplos:

Un cliente solicita incorporar una funcionalidad que implica un subsidio estatal a determinado tipo de préstamos.

Si es un subsidio estatal es una funcionalidad que aplica para la plaza, por lo tanto si obtenemos un nuevo cliente en la plaza en cuestión es muy probable que requiera dicha funcionalidad, en este caso es conveniente incorporarla de forma genérica, es decir no restringirla al cliente solicitante.

Un cliente solicita un nuevo tipo de comisión que se calcula en base al capital en las cuotas de cierto tipo de préstamo.

En el desarrollo debe contemplarse la posibilidad de calcular ésta comisión sobre los restantes componentes de la cuota o sobre la combinación de estos. También debe contemplarse que sea posible calcularla para cualquier tipo préstamo.

De esta forma el conocimiento se va acumulando y el abanico de posibilidades se agranda, de modo que al llegar al próximo cliente las necesidades de incorporar funcionalidades mediante programación se reducen y al mismo tiempo se amplían las posibilidades de diferenciación.

1.4 Presentación de los datos

Al desarrollar programas que desplieguen datos, ya sea por pantalla o spool (PDF, TXT, etc.) debemos asegurar que la información desplegada sea legible, para que le resulte útil a la persona que la está utilizando.

Para ello mínimamente se deben considerar los siguientes ítems:

- TÍtulo – Todos los programas deben tener un título centrado y con letras mayúsculas.
- Cabezales de listados – Si encabezan campos numéricos deben estar alineados a la derecha de lo contrario a la izquierda.
- Alineaciones – Las diferentes filas del listado deben estar alineadas de forma horizontal y las columnas de forma vertical.
- Corrección ortográfica – Las palabras deben estar escritas de forma correcta.
- Etiquetas – Los textos de las etiquetas deben ser descriptivos, se deben evitar nombres de atributos (scmda, pgcod, etc.).
- Debugs – Los mismos deben cumplir los criterios antes mencionados.

A continuación presentamos dos ejemplos de buenas prácticas desplegando datos en reportes, el primero en formato PDF y el siguiente en formato TXT, los cuales representan los patrones que deben seguirse al momento desarrollar / corregir programas:

CMBC004 Banco Modelo (BTV2R3)

Fecha : 06/05/13
Hora : 09:45:32
Página : 1

CONSULTA DE MONEDAS Y BILLETES DE CAJA

Caja : 55 **Sucursal:** 25 Sucursal ZonaAmerica
Cajero: Usuario Generico (BANTOTAL)

Cantidad útil	Cantidad inútil	Valor	Tipo	Total
Moneda 0 - $ PESOS				
10	0	1.00	M	10.00
0	0	2.00	M	0.00
0	0	5.00	M	0.00
0	0	10.00	M	0.00
0	0	20.00	B	0.00
0	0	50.00	B	0.00
0	0	100.00	B	0.00

ANUR001a

Fecha: 26/04/13
Hora : 17:39:26
Hoja : 1

CONTROL DE SALDOS POR SUCURSAL/RUBRO
(Saldo Inicial + Movimientos <> Saldo Final)

Período: 19/04/11 a 20/04/11

Rubro	Descripción	Fecha	Suc	Saldo Anterior	Movimientos	Saldo Actual	Diferencia
335136057	ACREEDORES VARIOS - MODULOS 20 Y 21	20/04/11	100	714.60	0.00	0.00	714.60
335136722	AS400-RENTAS Y AMORT. TITULOS	20/04/11	17	150.13	0.00	0.00	150.13
335136722	AS400-RENTAS Y AMORT. TITULOS	20/04/11	100	0.03	0.00	0.00	0.03
351003001	SUC Y AGS FDOS TERC TRIB AUT	20/04/11	17	8,910,889.50-	0.00	0.00	8,910,889.50-

2.0 Código de programas fuentes

2.1 Uso de Mayúsculas y Minúsculas

Se debe prestar mucha atención al uso de minúsculas y mayúsculas. Para la sintaxis de comandos se recomienda la sugerida por el editor GeneXus.

A modo de ejemplo: no es lo mismo leer en un código `if bcsdmn >= &aimpmin` que leer `If BCSdMn >= &aImpMin`

2.2 Indentación

Se considera imprescindible la indentación correcta de los fuentes de los programas, ya que facilitan en mucho su lectura y comprensión.

El programador elegirá la modalidad que le resulte más cómoda dentro de las dos opciones que se describen. Sin embargo, no está permitido variar la indentación de un programa ya indentado con otra modalidad, es decir, no se admite mezclar distintos criterios en un mismo fuente.

2.3 Sintaxis de Sentencias particulares

2.3.1 "For Each"

En el uso del `For Each` se han detectado dos modalidades de uso más frecuentes. Ambas se consideran válidas en tanto mantengan la indentación general del fuente, aunque se sugiere el uso de la primera opción.

```
For Each // -- <Tabla>-<Índice>: <Descripción de la Tabla>
      Where
      Where
      Defined By

      Código
EndFor      // <Tabla>
```

ó

```
For Each // -- <Tabla>-<Índice>: <Descripción de la Tabla>
Where
Where
Defined By

      Código
EndFor      // <Tabla>
```

➔ Se considera de uso obligatorio la sentencia del DEFINED BY, estipulando un atributo secundario de la tabla recorrida, para evitar navegaciones no deseadas al trasladar un programa de un modelo a otro.

2.3.2 "If"

Se considera imprescindible utilizar por cada '`If`' su '`Else`' correspondiente, aún en el caso en que no se deba incluir código para el mismo, como asimismo el uso de comentario de lo que está sucediendo si se cumple la condición.

```
If   // Si sucede tal cosa

Else // Si NO sucede tal cosa

EndIf
```

Ha demostrado ser de utilidad en la prevención de errores, sobre todo en el anidamiento de decisiones.

2.3.3 "Case"

Es de uso obligatorio el '`Otherwise`' en las sentencias `Case` para evitar tener efectos laterales por la no especificación de casos posibles.

```
Do Case
   Case      // Aquí es cuando se cumple tal condición
   Case
   Otherwise // Cuando no se da ningún caso anterior
EndCase
```

De la misma forma que para la sentencia If, se debe comentar la condición que se cumple.

2.4 Variables

Como regla general no se deben reutilizar variables en distintas partes del código, cuando éstas deben reflejar conceptos distintos.

Esto puede conducir, en caso de uso descuidado, a efectos laterales difíciles de encontrar en el código.

En particular la regla que se recomienda es la siguiente:

> "A mismo concepto misma variable, a conceptos distintos variables distintas"

El caso más notorio de reutilización son las variables bandera o flag.

Estas se utilizan para indicar el cumplimiento de una condición booleana.

En este caso es recomendable utilizar variables distintas para indicar condiciones distintas, se recomienda que el nombre refleje la condición que controla.

Ejemplo:

```
&Hay
&Ok
```

Si hubiera que aplicar estas variables a conceptos más concretos como cuentas y usuarios se recomienda lo siguiente:

```
&HayCta
&CtaOk

&HayUsu
&UsuOk
```

➔ Se recomienda la utilización de **Dominios** Genexus, de manera de definir las variables de forma similar a dichos dominios.

A modo de ejemplo, se citan dominios:

Fechas,
Importes…, de donde se toma la definición y el picture correspondiente, uniformizando toda la aplicación.

2.4.1 Definición

Se considera de buena práctica definir, en lo posible, todas las variables basadas en un atributo.

Tipos de variables:

- Atributo: **Para aquellas variables que deban representar un atributo se recomienda utilizar el nombre del atributo como nombre de la variable**.

- Parámetro: las variables que se utilizan como parámetros se le agrega una "p" inicial. Ejemplo: `&pPgCod`

- Vector: para este tipo de variables se aceptan dos nomenclaturas posibles, agregan una "v" o "vec" al inicio. Ejemplo: `&vCtas() &vecCtas()`

- Otros Casos: También han surgido generalizaciones de los casos anteriores a otros usos.
  - Variables Auxiliares: se utiliza para éstas el prefijo "a", aunque es también común la utilización del prefijo "aux".
    Ejemplo: `&aCodEnt`, `&AuxCta`)
  - Constante: aunque poco utilizadas ya que generalmente se utilizan como auxiliares agregarían el prefijo "f", aunque puede utilizarse también el prefijo "cte"
    Ejemplo: `&fCodEnt`, `&cteCta`)
  - Iteración: Contrariamente al uso general de i, j y k como variables de iteración, se propone su sustitución por variables que incluyan en el nombre el concepto que quieren recorrer. A modo de ejemplo: `&iCta, &iVec, etc`
  - Contador: para las variables contador se recomienda un esquema parecido al anterior.
  - Condición: Para el caso de la variable condición se recomienda evitar la reutilización de las mismas de acuerdo a lo expresado anteriormente.

➔ **NO *se deben definir variables ni vectores que no se utilicen dentro del programa***. Esto provoca que Genexus genere programas más grandes sin sentido alguno. Además de existir una

restricción en la generación para la plataforma AS/400 en RPG, al generar con ILE. O sea TODAS las variables definidas deben estar utilizadas en algún lugar del objeto GX.

2.4.2 Parámetros

- No se debe utilizar una variable definida como parámetro, como de entrada y salida a la vez. Es decir, un parámetro DEBE **pasar datos al programa o devolverlos**.

- Definir cada parámetro si es de entrada o de salida, en la 'rule' :
  parm(out:&par1, in:&par2, &par3)

  La rule "parm" así detallada es soportada por Gx para todo lenguaje generado, más allá de tener sus beneficios de performance en Java.

2.5 Condiciones Booleanas

Como regla general, se ha encontrado que a los efectos de la legibilidad del código no es bueno tomar demasiadas decisiones en una línea de código.

No es recomendable definir condiciones booleanas con más de dos o tres operandos, ya que no es natural la lectura de ANDs y ORs anidados.

Es importante también destacar el hecho de tratar de dar significado semántico a las condiciones booleanas.

Si se tiene por ejemplo una condición del tipo:

```
Cond1 and Cond2 .. and CondN
```

Es preferible definir una subrutina que reduzca la condición a una variable cuyo nombre represente mejor el concepto que se quiere controlar.

```
Do 'Controlar Condición'
If &Cond = 'S'
```

Lo mismo es aplicable a condiciones que no son entendibles directamente de la lectura de sus componentes, recomendándose el uso de paréntesis como separadores, aunque no sean necesarias para la evaluación de la condición.

2.6 Comentarios

Es de suma importancia el tratamiento de comentarios en los fuentes, haciéndose imprescindible el MANTENIMIENTO de los mismos.

2.6.1 Comentario en el "Cuerpo" del Fuente

Se deben utilizar comentarios en todas las líneas donde se considere necesario para dar legibilidad al código.

Se recomiendan (exige) algunos casos particulares de uso:

- For Each

```
For Each     // -- <Tabla><[índice]>: <Descripción >
 Where…
 Defined By <Atributo>
 EndFor      // <Tabla>
```

A modo de ejemplo:

```
For Each     // -- FSD014-01: Plan de Cuentas
   Where Rubro = &aRubro
   Defined By PcNomR

EndFor // FSD014
```

- Antes de SubRutinas o antes de Call's:

```
// -- Obtengo Datos de Cuenta:
&aBCCta = BCCta
Do 'Datos de Cuentas'

// -- Obtengo Tipo de Cambio:
call(..)
```

- Antes decisiones:

```
//-- Validar Cuenta:
If &aCtaVal = 'S'
Else
EndIf
```

- Aclarando el cumplimiento de condiciones en las sentencias If ó Case.

2.7 Subrutinas

Es de muy buena práctica el uso de subrutinas para 'modularizar' la obtención de datos auxiliares, toma de decisiones, etc.

Como se mencionó con anterioridad, no es bueno tomar demasiadas decisiones en un bloque de código. Facilita mucho la comprensión de un fuente el agrupar bloques de lógico que tienen una funcionalidad identificable como subrutinas.

Para el uso de subrutinas se recomienda el siguiente estándar:

```
        // ----------------------
        Sub 'Nombre Subrutina'  // -- Descripción de la subrutina
        // ----------------------
              // Recibe  : Variable(s)
              // Devuelve: Variable(s)

              ...

        EndSub // -- 'Nombre Subrutina'
```

La especificación de las variables de entrada y salida nos permite identificar fácilmente el alcance de la subrutina y evita la producción de efectos laterales.

A manera de ejemplo:

```
// ------------------------------------------------------------------------
// --  S U B - R U T I N A S
// ------------------------------------------------------------------------

// ---------------------------
Sub 'Inicio'               // -- Inicialización de Ambiente
// ---------------------------

     // -- Grabo versión del Programa (FBC013);
     call(PVersion, &pPgCod, &Pgmname, &aFchPrg)

     // -- Nivel de Detalle[1] (0 a 9):
     call(PBCDebug, &pPgCod, &PgmName, &aDbgNiv)

     // -- Si la Fecha pasado como parámetro NO es hábil, busco la anterior
     call(PBCFchHa, &pPgCod, &pFchFin, 'D')  // <A>scendente <D>escendente

     ...
 EndSub      // -- 'Inicio'

 // ---------------------------
 Sub 'Obtengo Algo'          // -- Determina datos de...
 // ---------------------------
      // Recibe  [2]: &Rubro, ...
      // Devuelve[3]: &aAlgo

      &aAlgo = nullvalue(&aAlgo)
      ...

EndSub       // -- 'Obtengo Algo'
```

[1] Todos los programas deben tener la posibilidad (en caso de ser necesario) de imprimir datos de debug, o sea, datos que ayuden al usuario ante cualquier duda a resolver el problema. Se maneja también diferentes niveles de debug (0 a 9); cuanto mayor es el nivel de debug, mayor detalle se presentará.
[2] Parámetro(s) que son utilizados en la rutina
[3] Parámetro(s) que devuelve la rutina

En el ejemplo antes citado se puede apreciar con claridad:

- Control de versión (sólo en programas batch no llamados por ningún otro)
- Uso de mayúsculas y minúsculas
- Tabulación (Indentación)
- Uso de comentarios
- Uso de subrutinas

2.8 Modificaciones/Mantenimiento del programa

Se deberán comentar los diferentes cambios que "sufre" el programa ("Historia del Objeto"). Se trata de aportar una frase que describa el cambio. Algo concreto que de una buena idea del cambio, sin entrar en detalles. Dichos comentarios se deben registrar en la sección "Documentation" que posee un fuente GeneXus.

Los comentarios se escriben en formato XML y tiene la siguiente forma:

```
<Historia>

   <Modificación>
      <Fecha>dd/mm/aaaa</Fecha>
      <Autor></Autor>
      <Solicitud>Nro.Web</Solicitud>
      <País></País>
      <Cliente></Cliente>
      <Descripción>Descripción del Cambio</Descripción>
   </Modificación>

</Historia>
```

NOTA: Los ítems País y Cliente sólo deben llenarse si el cambio está condicionado a un país y/o Cliente en particular.

2.9 Restricciones

2.9.1 Ejecución en Servidor vs. Cliente

Queda terminantemente PROHIBIDO que si un programa está definido para ejecutar en el servidor, se modifique para que corra en el cliente.

3.0 Parámetros Recibidos

3.1 Parámetros Generales

3.1.1 Parámetros Estándar de programas batch

Versión sin extensión de parámetros

Estos parámetros son los utilizados en la cadena de cierre. Dichos parámetros son:

- Empresa
- Fecha de Cierre
- Fecha Apertura
- ¿Contabiliza?

Versión con extensión de parámetros

Estos parámetros son los utilizados en la cadena de cierre. Dichos parámetros son:

- Empresa
- Fecha de Cierre
- Fecha Apertura
- ¿Contabiliza?
- Impreso
- Secuencia
- Vía

4.0 Rutinas Generales

Lo que intentan estas rutinas es resolver casos de mediana y alta complejidad y, sobre todo, que se utilicen con frecuencia.

La gran ventaja (además de ahorrar tiempo de programación al evitar tener que hacer o rehacer funcionalidades en cada programa) es la del alcance de un cambio, o sea, si por algún motivo hay que cambiar alguna funcionalidad ésta se realizará en un único programa.

Como ejemplo tenemos varios, pero se citarán sólo algunos:

- Obtención de Integrantes de cuenta
- Obtener empresa de usuario
- Desglose de cuotas de préstamo

4.1 Rutinas de Tratamiento Especial (RTE's)

Es un programa (procedure o work panel) hecho a medida, que se asocia a un ordinal (más específicamente a un subordinal) de una transacción Bantotal. Debe tener un código específico asignado, por lo que se debe reservar a través de la WEB.

Parámetros que recibe obligatoriamente*:*

- Empresa
- Sucursal (de ingreso)
- Módulo (de ingreso)
- Transacción
- Relación
- Ordinal
- Subordinal (para el caso de un ordinal repetitivo)

Estos primeros seis parámetros constituyen la clave de la tabla FSD016

Parámetros que devuelve obligatoriamente*:*

- Retorno: devuelve C(1) indicando si hubo error o no. Este parámetro devuelto en 'S' provoca que el transaccional Bantotal finalice la ejecución. Devuelto en 'N', indica que se puede continuar la ejecución normalmente.

Ejecución de Rutinas de Tratamiento Especializado.

Aquellas rutinas que deban ser ejecutadas dentro del ciclo de confirmación de una transacción contable (En Confirmación), en cuyos trabajos se alteren registros de tablas, deberán poseer la característica COMMIT ON EXIT = NO.

Las propiedades de cada RTE cambian respecto al entorno de donde ejecuten. Ver el documento Momento ejecución RTEs.xlsx para mas detalle.

4.2 Preformatos

Es un programa work panel hecho a medida, que se asocia a un asiento de una transacción Bantotal. Normalmente tiene un programa 'Procedure' asociado. Deben tener un código específico asignado, por lo que se debe reservar a través de la WEB.

Parámetros que recibe obligatoriamente*:*

- Empresa
- Sucursal (de Ingreso)
- Módulo (de Ingreso)
- Transacción
- Relación

Parámetros que devuelve obligatoriamente:

- Retorno: devuelve C(1) indicando si hubo error o no. Este parámetro devuelto en 'S' provoca que el transaccional Bantotal finalice la ejecución. Devuelto en 'N', indica que se puede continuar la ejecución normalmente.

Cumplen con una nomenclatura asignada: P o W, seguido de PF.
Por ej. :

WPF60009 – Preformato
PPF60009 – Llamado por el WP del ejemplo

➔ La definición de los parámetros de las RTEs y Preformatos, se corresponden con los atributos Bantotal:

Empresa	–	Pgcod
Sucursal	–	Sucurs
Módulo	–	TrMod
Transacción	–	TrNro
Relación	–	ItNRel
Ordinal	–	TrOrd
Subiordinal	–	TrSBor

5.0 Métodos de Control

5.1. Control de Versión

El objetivo es disponer de un procedimiento que permita realizar un seguimiento en DLYA y en los clientes de las versiones de los programas que ejecutan.

La idea es que cada programa que se desee controlar se le agregue al comienzo un call a la rutina pVersion de la siguiente manera:

```
&aFchProg = YMDToD(2013,04,29)
call(pVersion, &pPgCod, &PgmName, &aFchProg)
```

donde:

- &pPgCod : es la empresa (N3),
- &PgmName : es el nombre del programa donde se coloca el call (C8) y
- &aFchProg : es la fecha en que se está modificando el programa (Date).

Este programa genera un registro en la tabla FBC013 donde se graba también la fecha de ejecución (&Today) y la Hora (&Time).

El Método a aplicar consiste en que cuando se va a modificar el programa PXXX, se efectúa el cambio solicitado y si no tiene el call a la rutina pVersion, se agrega y se carga la fecha &FchVer con la fecha del día que se hace la modificación. Si el programa a modificar ya posee el call sólo se debe MODIFICAR la fecha &FchVer con la nueva fecha de cambio (y se deja comentada la versión anterior).

El cliente tendrá un panel de trabajo (wVersion) donde puede ver la versión de los programas que ejecutó, para esto basta con llamar al wVersion (sin parámetros), donde debe ingresar el nombre del programa y se desplegará las fechas de ejecución del mismo y la(s) versión(es).

➔ Hay que tener en cuenta que NO es conveniente utilizarlo dentro de programas que son llamados por otros, ya que se grabarían varios registros en la tabla FBC013 de una misma ejecución de un programa.

5.2 Manejo de Debug

El manejo de 'debug' consiste en ofrecerle al usuario más información sobre lo que se está realizando; aplica a todo tipo de objetos pero por lo general es utilizado en los procedimientos.

Para hacer un 'debug' se debe crear un campo de información -tipo fijo- con el nombre del programa. Para setear niveles de detalle se debe cargar en valor específico (FSI002) un número entre 1 y 9, cuando mayor es el valor más detalle se mostrará.

6.0 Definición de transacciones GX y su compilación

Las transacciones GX deben utilizarse *generalmente* para los efectos de la definición de tablas y/o lógicas simples de ABM.

Si la lógica de la transacción va a ser compleja contando con reglas y controles, no se deben utilizar.

7.0 Auditorias de modificaciones en tablas

Lo que se describe a continuación tiene específica aplicación a desarrollos o actualizaciones de programas que modifiquen tablas, para todo modelo Java.

Quedan exceptuados aquellos objetos que intervienen en la ejecución de una transacción contable (asiento contable), ya que se conoce a través de la estructura Fsd015, qué usuario y cuándo provoca la alteración de registros en otras tablas.

En el momento de modificar o crear Transacciones o Procedimientos Gx en modelos Java, que eliminen y/o agreguen registros de/a la tabla correspondiente, se debe agregar el llamado a un procedimiento que genera un registro en una estructura (FsdLog), cuya finalidad es establecer el nombre del usuario, estación de trabajo, programa y fecha de la modificación.

La forma de hacerlo es incluir el llamado al programa PLOGGEN, con los siguientes parámetros:

Call(PLogGen,&PusuLog,&Pgmname,&Wrkst) donde &PusuLog es 'C' de 60 , &Wrkst,'C' de 60.

8.0 Multi Idioma

Lo que se describe a continuación tiene específica aplicación a desarrollos para todo modelo Java y Web.

Para facilitar la tarea de traducción de los fuentes a otros idiomas (portugués e inglés) y reconocer dentro del código variables que contienen un texto que debe ser traducido; se restringe el uso de algunas variables reservadas.

Por ejemplo tengo un listado que debe dar un mensaje como el que sigue:

&Msg = 'Por favor traducir'

Msg(&Msg)

Existe una herramienta que detecta la variable &Msg en el fuente y sustituirá el texto 'Por favor traducir' por el correspondiente en el lenguaje destino. Dado que esa herramienta es un reconocedor sintáctico deben acotarse las variables a utilizar con este fin. Así mismo no deben usarse dichas variables reservadas para almacenar datos que no se deben traducir por ejemplo &Msg = Ctnom (El resultado si el nombre de la cuenta fuese 'Carlos Bueno' sería 'Charlie Good').

También debe respetarse el espacio entre la variable y el signo de igual (&Msg= 'Por favor traducir' no sería reconocido).

Las variables reservadas para este cometido son:

&Mensa ,&Msg ,&Pmsg ,&WMsg ,&MsgError ,&Pmensa ,&Pmensa1 ,&Texto ,&Texto1 ,&Texto2 ,&Texto3

9.0 Controles Obligatorios

En este Manual se han definido como obligatorios algunos puntos del mismo. Estos puntos se encuentran subrayados y en fuente de color rojo.

La obligatoriedad de dichos puntos implica que cualquier integrante de DLYA debe cumplirlos para que puedan ingresar a los modelos consolidados. Estos puntos podrán variar con el correr del tiempo así como adicionarse otros.

Al momento los puntos obligatorios son: 1.1, 1.2, 2.2, 2.3.1, 2.6.1, 2.7, 2.8, 2.9.1, 3.1.1, 4.1, 4.2, 7.0 y 8.0.

Glosario

Por palabras de uso frecuente remitirse al **Manual de Terminología Bantotal**.

A continuación se explicitan las palabras específicas o particulares del tema tratado en el presente documento:

Ordinal: Es cada una de las “líneas” que conforman el asiento contable tipo que se define a través del Monitor Transaccional. La definición de los mismos y sus parámetros queda almacenada en la tabla FST035.

Subordinal: Es cada una de las opciones o sinónimos que puede tomar un ordinal. La definición de los mismos queda almacenada en la tabla FST036.

Formulario de Evaluación

Bantotal – Manual de Estándares de Programación
Número de Documento: BT-XXX-XXX-00

Sus comentarios son muy importantes y nos ayudan a mantener la calidad de los Manuales de Bantotal. Por favor sírvase completar el Formulario de Evaluación adjunto y háganoslo llegar a través de los siguientes medios:

Correo a de Larrobla & Asociados Internacional
Zonamérica Business & Technology Park
Edificio Synergia, Of. 002
Ruta 8 Km. 17.500, Montevideo – Uruguay

Fax a de Larrobla & Asociados Internacional
005982- 518.27.78

Por correo electrónico a:
documentacion@dlya.com.uy

POR FAVOR CLASIFIQUE EN UNA ESCALA DE 1 A 5 DE ACUERDO A:
(1 = muy bueno, 2 = bueno, 3 = en el promedio, 4 = pobre, 5 = muy pobre)

Satisfacción General _____

Organización del documento _____
Relevancia de la Información _____
Gramática / puntuación / deletreo _____
Facilidad de lectura y comprensión _____
Nivel de detalle técnico _____

Sugerencias & Comentarios:

_____

_____

_____

_____
Nombre

_____
Firma