Diagrama

Descripción generada automáticamente con confianza mediaUna caricatura de una persona

Descripción generada automáticamente con confianza baja

RCM Software SL

www.rcm.es www.rcmsoftware.us

**DATAGATE GDP  
SYNTAX   
HANDBOOK**

V 3.6

**¿QUÉ ES DATAGATE GDP For OpenEdge?**

**DataGate GDP** es un framework de “marca blanca” para ayudar a crear rápidamente aplicaciones web de Front-End o capa de representación y optimizar el desarrollo del Back-End o capa de negocio, en un entorno **PROGRESS OPENEDGE ™**.

Incorpora el sitio web con la seguridad de acceso, así como el sistema de menús, roles, usuarios, ….

El entorno de trabajo es el siguiente:

* Capa de representación (Front-End): **DATAGATE GDP**

* Capa de negocio (Back-End): **PROGRESS OPENEDGE ™** (OEPAS) V 12.2.2 o superior
* Capa de almacenamiento: **PROGRESS ™ Data Base** (Workgroup o Enterprise DB)

**DataGate GDP** está estructurado para crear los formularios necesarios para las aplicaciones en un entorno Web y optimizar el desarrollo de la capa de negocio y su posterior almacenamiento en la base de datos.

Los formularios realizados con **DataGate GDP Designer** permiten incorporar tantos subformularios o layers como sea necesario en el desarrollo y enlaza directamente el Front-End con el Back-End de forma nativa (sin necesidad de realizar “API’s”).

**DataGate GDP** organiza la capa de negocio en diferentes procedimientos internos de **PROGRESS ™ OpenEdge** para que el desarrollo sea más rápido y planificado. El grupo de desarrollo siempre trabajará en una misma línea estructurada y cualquier desarrollador podrá entender fácilmente las aplicaciones realizadas por otro desarrollador.

**DataGate GDP** incorpora métodos y funciones, desarrollados en ABL por nuestra compañía, para enlazar la capa de representación con la capa de negocio y viceversa, de una manera sencilla a través del lenguaje ABL de **Progress Software** **™**.

A continuación, se relacionan los métodos y funciones junto con sus sintaxis que incorpora **DataGata GDP**.

**DEFINICIONES ESPECIALES QUE SE AÑADEN POR USING A LOS PROCEDIMIENTOS**

USING Progress.Lang.\*.

USING Progress.Json.ObjectModel.\*.

USING DataGateGDP.Tools.\*.

USING DataGateGDP.Interfaces.\*.

USING DataGateGDP.Utils.\*.

**DEFINICIONES CON TIPOS DE DATOS SOPORTADOS POR OpenEdge SEGÚN LOS “USING” DEFINIDOS**

DEFINE VARIABLE wVisibility AS DataGateGDP.Models.Visibility.

DEFINE VARIABLE wListItemCombo AS ListItemTools.

DEFINE VARIABLE wListItemGrid AS GridTools.

DEFINE VARIABLE Data\_State AS DataState.

DEFINE VARIABLE DataAction AS ActionSearchBox.

**PROCEDIMIENTOS INTERNOS DEFINIDOS EN EL PROCEDIMIENTO GENERAL**

Para enlazar directamente el formulario / subformularios de DataGateGDP con el procedimiento de ABL solamente debe indicar el mismo nombre del formulario GDP al procedimiento ABL.

Como en la definición del formulario en GDP puede indicar el objetivo o descripción del formulario, podrá obtener una búsqueda rápida de los programas de la aplicación.

En los procedimientos (.p) que se enlazan automáticamente con el formulario definido en el Diseñador de DataGateGDP, puede utilizar cuatro (4) procedimientos internos específicos para el enlace entre el frontend (GDP) y el backend (Progress OpenEdge) y estas procedures no son obligatorias que existan todas.

PROCEDIMIENTOS INTERNOS Y SU UTILIZACIÓN

**LOAD** (Se ejecuta al cargar el formulario y permite asignar los valores iniciales de los campos definidos en el formulario – Procedure Load:)  
 **STATE** (Se ejecuta al salir del campo -leave- si ha habido un cambio de valor en la casilla del formulario – Procedure State:).

**ACTIONSEARCHBOX** (Exclusivo para recoger los dos eventos del Search-Box y permite recoger el dato introducido letra a letra y cargar la lista de valores a presentar al usuario – Procedure ActionSerachBox:).

**ACTIONDATAGRID** (Exclusivo para recoger los eventos del DataGrid que tengan la casilla de ENABLE activada – Procedure ActionDataGrid: ).

**VALIDATE** (Se ejecuta antes de llamar al procedimiento **STATE** y está creado para poder validar los datos introducidos por el usuario)

Por lógica deberá existir una visualización de información o grabación de datos y normalmente estarán en el **LOAD** o/y **STATE**.

Para que se pueda optimizar el código ABL se ha incorporado otros cinco (5) procedimientos externos (.p) que permiten definir, una única vez en todo el proyecto que se realiza con OpenEdge, y permite definir los eventos que pueden ser globales a la aplicación. Para lo cual hay que utilizar para el mismo concepto en la aplicación el mismo nombre de variable, y que se ejecutarán después de ejecutar los procedimientos **Load, State, ActionSearchBox y ActionDataGrid**.

Estos cinco procedimientos son:

R-LoadStdGDP.p 🡪 Carga Estándar de datos para toda la aplicación.

R-StateStdGDP.p 🡪 Eventos a realizar a la salida de una variable definida en el formulario GDP.

R-SeachBoxActionsStdGDP.p 🡪 Eventos a realizar en el control o componente Search-Box.

R-GridActionsStdGDP.p 🡪 Eventos a realizar en un DataGrid que tiene columnas de entrada de datos activas.

R-ValidateStdGDP.p 🡪 Validaciones estándar de los datos sin necesidad de incorporarlos en cada procedimiento.

Si el evento del campo del formulario ya está definido en **Load, State, ActionSearchBox, ActionDataGrid o Validate** no se ejecutan los procedimientos denominados estándar (STD) por medio del parámetro de salida **ExistField**, como veremos a continuación.

Estos procedimientos reducen considerablemente el código ABL necesario en las aplicaciones.

Los cuatro procedimientos necesitan tener los parámetros de entrada y salida siguientes:

**Para el LOAD, STATE y VALIDATE**:

DEFINE INPUT PARAMETER IdGuid AS CHARACTER.

DEFINE INPUT PARAMETER NameForm AS CHARACTER.

DEFINE INPUT PARAMETER LayoutId AS INTEGER.

DEFINE INPUT PARAMETER Name AS CHARACTER.

DEFINE INPUT PARAMETER Data AS LONGCHAR.

DEFINE INPUT PARAMETER DataState AS DataState.

DEFINE OUTPUT PARAMETER ExistField AS LOGICAL.

**Para el ActionSearchBox**:

DEFINE INPUT PARAMETER IdGuid AS CHARACTER.

DEFINE INPUT PARAMETER NameForm AS CHARACTER.

DEFINE INPUT PARAMETER LayoutId AS INTEGER.

DEFINE INPUT PARAMETER Name AS CHARACTER.

DEFINE INPUT PARAMETER Data AS LONGCHAR.

DEFINE INPUT PARAMETER DataAction AS ActionSearchBox.

DEFINE OUTPUT PARAMETER ExistField AS LOGICAL.

**Para el ActionDataGrid**:

DEFINE INPUT PARAMETER IdGuid AS CHARACTER.

DEFINE INPUT PARAMETER NameForm AS CHARACTER.

DEFINE INPUT PARAMETER LayoutId AS INTEGER.

DEFINE INPUT PARAMETER Name AS CHARACTER.

DEFINE INPUT PARAMETER Data AS LONGCHAR.

DEFINE INPUT PARAMETER DataAction AS ActionGrid.

DEFINE OUTPUT PARAMETER ExistField AS LOGICAL.

Estos parámetros indican el campo o variable del formulario de GDP que ha modificado su valor o estado.

La variable **IdGuid** entrega el Identificador único del proceso. Importante porque el sistema de estabilidad interna de la aplicación se basa en esta variable.

La variable **NameForm** indica el nombre del formulario que se está ejecutando.

La variable **LayoutId** indica el número de subformulario o layer que se está ejecutando.

La variable **Name** entrega el nombre de la variable o campo que se está tratando.

La variable **Data** entrega el valor en pantalla de la variable **Name**.

La variable **DataState** entrega una clase (Array) para poder interactuar con el FrontEnd.

La variable **ActionGrid** (solo para el Datagrid) entrega los valores de la línea del DataGrid que se está tratando.

La variable **ActionSearchBox** (solo para SearchBox) entrega los valores del Search-Box carácter a carácter.

La variable **ExistField** indica al proceso si se ha encontrado o no el nombre del campo del formulario en el procedimiento y buscar o no dicho campo en los procedimientos globales (R-LoadStdGDP.p, R-StateStdGDP.p, R-ValidateStdGDP.p, R-SeachBoxStdGDP.p y R-GridActionsStdGDP.p)

Permite manipular su valor si desea que además de ejecutar el procedimiento en cuestión, se ejecute el procedimiento global.

**RECOLECCIÓN DE DATOS INTRODUCIDOS EN UN FORMULARIO GDP**

En DataGateGDP no es necesario generar URI’s o End-Point de comunicación entre el Front-End y el Back-End. Se podrá realizar, pero no es obligatorio.

Los datos llegan a los procedimientos anteriormente indicados, por medio de los parámetros de entrada, como se ha definido anteriormente.

Para mayor comodidad en el desarrollo el DataState recomendamos pasarlo al objeto Form\_Data (JsonObject) y se podrá conocer el valor de cualquiera de las variables del formulario por medio de GetCharacter, GetInteger, … como indicamos a continuación.

Form\_Data = Data\_State:**GetCurrentLayoutDataJsonObject().**

Si el formulario tiene diferentes subformularios o layouts se pueden recoger todos los valores a la vez para que el desarrollo sea más fácil.

Form\_Data\_Layout\_1 = Data\_State:**GetLayoutDataJsonObject(1).**

Form\_Data\_Layout\_2 = Data\_State:**GetLayoutDataJsonObject(2).**

Form\_Data\_Layout\_3 = Data\_State:**GetLayoutDataJsonObject(3).**

El lenguaje ABL de **OpenEdge** incorpora los mandatos necesarios para recuperar desde un objeto “Json” el valor en pantalla dependiendo del tipo de dato.

MiCampoAlfa = Form\_Data:**GetCharacter(**"Nombre del Campo en el Formulario").

MiCampoIntegro = Form\_Data:**GetInteger**("Nombre del Campo en el Formulario").

MiCampoDecimal = Form\_Data:**GetDecimal**("Nombre del Campo en el Formulario").

MiCampoLogico = Form\_Data:**GetLogical**("Nombre del Campo en el Formulario").

MiCampoFecha = Form\_Data:**GetDate**("Nombre del Campo en el Formulario").

…..

Los nombres de campos deben coincidir en mayúsculas y minúsculas con los nombres del formulario (Case Sensitive).

Recuerde que desde el procedimiento **STATE** puede ejecutar el procedimiento **LOAD** para algunos campos específicos porque esta en el mismo procedimiento (.p)

**RUN Load** (IdGuid, NameForm, LayoutId, "Nombre del Campo en el Fomulario", Data, DataState, OUTPUT ExistField).

**CARGA O MODIFICACION DE LOS DATOS EN EL FORMULARIO**

Los datos en el formulario del cualquier subformulario pueden ser cargados o modificados en cualquier momento por medio de dos funciones o métodos:

**SetCurrentLayoutData** 🡪 para el formulario que está activo en el GDP.

**SetLayoutData** 🡪 para un formulario en específico. Recuerde cargar el JsonObject desde el DataState.

SINTAXIS:

Data\_State:**SetLayoutData**(“Número de Subformulario”, Nombre del Campo", “Visibilidad del campo (\*)”, “Valor del campo a cargar(\*\*)”, ”Lista de Valores (\*\*\*)”).

Ejemplo: Data\_State:**SetLayoutData**(5,CodigoCliente, DataGateGDP.Models.Visibility:nottouch, wObjectValue, ?).

(\*) La visibilidad tiene definidos los valores: Normal, Disabled, ReadOnly, Hidden, NotVisible y NotTouch.   
 (\*\*) Si el componente permite agregar lista de valores (Combo-Box, Search-Box, DataGrid,..), este valor se utilizará para asignar el valor inicial de la lista.   
 Si no lo permite, se utiliza para rellenar el campo con dicho valor. El componente DataGrid está exento de dicho valor.  
 (\*\*\*) Si el componente o control no admite una lista de valores (Ejemplo: Control Input) en la lista se le envía el valor NULL.

Si no desea modificar el valor del campo a cargar en cualquier debe indicar entre compillas el valor interrogante “?”.

Para los componentes que no pueden recibir una lista (Input, Label, ...) puede enviar el valor null de Progress (?) sin entrecomillado.

Data\_State:**SetCurrentLayoutData**(“Número de Subformulario”, Nombre del Campo", “Visibilidad del campo”, “Valor del campo a cargar”, ”Lista de Valores”).

Ejemplo: Data\_State:SetCurrentLayoutData(CodigoCliente, DataGateGDP. Models.Visibility:nottouch, wObjectValue, wValueList).

**CARGAR LOS DATOS EN UN DATAGRID**

Uno de los componentes o controles más importante en los formularios es el DataGrid, porque permite visualizar una lista de registros, seleccionar un registro, marcar varios registros a la vez, añadir, modificar, clasificar, ordenar, cambio de estilo de la celda que contine el dato, ...

Para este componente hemos incorporado dos maneras de carga de los registros:

Carga Simple 🡪 El sistema de implementación es más sencillo, aunque es más lento de ejecución. Carga sobre 1.060 registros x segundo en un grid de 30 columnas + Style para un WorkGroup database (utiliza un solo núcleo).

Carga Rápida 🡪 El sistema de implementación es un poco más compleja, aunque la carga ejecución de mucho más rápida. Carga sobre 1.600 registros  
 x Segundo en un Grid con 30 columnas + Style para un WorkGroup database (utiliza un solo núcleo).

**CARGA SIMPLE**

wListItemGrid = Data\_State:**Create\_Grid**(“Nombre del control o componente DataGrid”). /\* se instancia \*/

FOR EACH Table NO-LOCK:

wListItemGrid:**BEGIN\_ROW**(). /\* Inicio de fila \*/

wListItemGrid:**ADD\_COLUMN\_DATA**("Nombre Columna", Table.MyField).

/\* para incorporar un campo con Style se ha incorporado “color:Red” \*/

wListItemGrid**:ADD\_COLUMN\_DATA**("Nombre Columna", Table.MyField, "color:red").

/\* más columnas \*/

/\* No es necesario indicar el tipo de dato que se envía y si no se desea cambiar el estilo no se es necesario enviar el parámetro de estilo \*/

/\* si la columna no existe en el DataGrid se devolver error en tiempo de ejecución (CASE SENSITIVE) \*/

wListItemGrid:**END\_ROW()**. /\* notificar el final de fila \*/

END.

/\* publicar la información enviada al DataGrid por medio de SetLayoutData o SetCuyrrentLayoutData \*/

Data\_State:**SetLayoutData**(Numero de Layer, "Nombre del Grid", DataGateGDP.Models.Visibility:normal, "?", wListItemGrid).

/\* La lista de valores va en el objeto wListItemGrid y el valor único con “?” se le indica que no se va a cambiar el valor y no lo carga \*/

**CARGA RÁPIDA**

wListItemGrid = Data\_State:**Create\_Grid** (Numero de Layer,"Nombre DataGrid"). /\* igual que en la carga simple \*/

hBuffer = wListItemGrid:**GET\_HANDLE**(). /\* recuperar el handle del DataGrid \*/

FOR EACH Table NO-LOCK:

hBuffer:**BUFFER-CREATE** (). /\* Inicio de fila \*/

**/\* carga normal de la columna del datagrid\*/**

hField = hBuffer:**BUFFER-FIELD**("Nombre Columna").

hField:**BUFFER-VALUE** = Table.MyField.

**/\* incorporar \_STYLE al nombre del campo si se desea cambiar\*/**

hField = hBuffer:**BUFFER-FIELD**(("Nombre Columna" + “\_STYLE").

hField:**BUFFER-VALUE** = "color:red".

/\* más columnas \*/

/\* No es necesario indicar el tipo de dato que se envía y si no se desea cambiar el estilo no se es necesario enviar el parámetro de estilo\*/

/\* si la columna no existe en el DataGrid se devolver error en tiempo de ejecución (CASE SENSITIVE) \*/

hBuffer:**BUFFER-RELEASE**(). /\* notificar el final de la fila \*/

END.

/\* publicar la información enviada al DataGrid por medio de SetLayoutData o SetCuyrrentLayoutData \*/

Data\_State:**SetLayoutData**(Numero de Layer, "Nombre del Grid", DataGateGDP.Models.Visibility:nottouch, "?", wListItemGrid).

/\* La lista de valores va en el objeto wListItemGrid y el valor único con “?” se le indica que no se va a cambiar el valor y no lo carga \*/

**MARCAR LA CASILLA DE MULTISELECCION.**

Esta opción permite enviar al DataGrid la(s) seleccion(es) necesarias que deben cumplir las filas del DataGrid para que aparezcan activada la casilla “multiselección”.

Se pueden enviar en cualquier momento de la carga y se visualizarán al visluazar el DataGrid cargado.

Para marcar la casilla de “MultiSeleccion” al cargar la información en el DataGrid se debe realizar de la siguiente manera:

En la variable definida como GridTools (DEFINE VARIABLE **wListItemGrid** AS GridTools NO-UNDO.) hay que incorporar el siguiente código dentro de la carga del DataGrid:

**wListItemGrid**:ADD\_SELECTOR("*NombreColumnaDelDatagrid a Seleccionar*",

“*Valor de la Variable a Seleccionar*”,

DataGateGDP.Models.FilterOperator:”*Operador Soportado*”).

Los operadores soportados son:

None

Contains

StartsWith

EndsWith

Equal

NotEqual

GreaterThan

GreaterThanOrEqual

LessThan

LessThanOrEqual

IsNull

IsNotNull

IsEmpty

IsNotEmpty

DoesnotStartWith

DoesNotContain

DoesNotEndWith

Es decir, Por cada registro que se desee marcar la casilla de “Multi-Seleccion” se debe incorporar el siguiente código:

**wListItemGrid**:BEGIN\_SELECTOR().

**wListItemGrid**:ADD\_SELECTOR("Ejercicio", Albaranes.CodEjercicio, DataGateGDP.Models.FilterOperator:Equal).

**wListItemGrid**:ADD\_SELECTOR("Albarán", Albaranes.CodAlbaran, DataGateGDP.Models.FilterOperator:Equal).

**wListItemGrid**:END\_SELECTOR().

Y al acabar la carga, este GridTools (**wListItemGrid**) se debe colocar dos veces. Una en el valor del objeto y otra en la lista del filas que le llegarán al DataGrid.

Ejemplo para carga del DataGrid de CARGAR RAPIDA:

/\* Instanciar el objeto \*/

**wListItemGrid** = Data\_State:**Create\_Grid**(4,"GridAlbaranes").

hBuffer = **wListItemGrid**:GET\_HANDLE().

FOR EACH Albaranes NO-LOCK:

hBuffer:BUFFER-CREATE(). /\* indicar el inicio de la fila \*/

DEFINE VARIABLE wColor AS CHARACTER NO-UNDO.

wColor = ‘ ‘. /\* colorear a la vez la línea \*/

IF Albaran.Facturado = TRUE THEN wColor = "color:red".

hField = hBuffer:BUFFER-FIELD("Empresa").

hField:BUFFER-VALUE = Albaran.Tenand.

hField = hBuffer:BUFFER-FIELD("Empresa\_STYLE").

hField:BUFFER-VALUE = wColor.

hField = hBuffer:BUFFER-FIELD("Centro").

hField:BUFFER-VALUE = Albaran.Centro.

hField = hBuffer:BUFFER-FIELD("Importe").

hField:BUFFER-VALUE = Albaran.Importe.

hField = hBuffer:BUFFER-FIELD("Cliente").

hField:BUFFER-VALUE = Albaran.Nombre.

hField = hBuffer:BUFFER-FIELD("ProntoPago").

hField:BUFFER-VALUE = Albaran.ImpPpgo.

/\* … resto de columnas del DataGrid \*/

/\* seleccionar los albaranes que están facturados y se desea que aparezcan marcados con la calilla” Multi-selección” seleccionada. \*/

**IF Albaran.Facturado = TRUE THEN DO:**

**wListItemGrid:BEGIN\_SELECTOR().**

**wListItemGrid:ADD\_SELECTOR("Centro", Albaran.Centro,**DataGateGDP.Models.FilterOperator**:Equal).**

**wListItemGrid:ADD\_SELECTOR("Empresa", Albaran.Tenand,**DataGateGDP.Models.FilterOperator**:Equal).**

**wListItemGrid:END\_SELECTOR().** /\* final de la selección de esta fila \*/

**END.**

**END.** /\* final for each \*/

/\* cargar el DataGrid con las selecciones de marcado. Primera para la selección y segunda para las filas a visualizar \*/

**Data\_State:SetLayoutData(4,"GridAlbaranes",** DataGateGDP.Models.Visibility:NotTouch**, wListItemGrid, wListItemGrid).**

DataGateGDP seleccionará una y otras y asignará exclusivamente las líneas del SELECTOR para la selección y las filas se asignarán como “Data” a representar.

Ejemplo para la carga normal:

/\* Instanciar el objeto \*/

**wListItemGrid** = = Data\_State:Create\_Grid(4,"GridLineasAlb").

FOR EACH Albaranes NO-LOCK:

wListaDeGrid:BEGIN\_ROW(). /\* indicar el inicio de la fila \*/

DEFINE VARIABLE wColor AS CHARACTER NO-UNDO.

wColor = ‘ ‘. /\* colorear a la vez la línea \*/

IF Albaran.Facturado = TRUE THEN wColor = "color:red".

wListaDeGrid:ADD\_COLUMN\_DATA("Empresa", Albaran.Tenand, wColor).

wListaDeGrid:ADD\_COLUMN\_DATA("Centro", Albaran.Centro, wColor).

wListaDeGrid:ADD\_COLUMN\_DATA("Importe", Albaran.Importe, wColor).

/\* … resto de columnas del DataGrid \*/

/\* seleccionar los albaranes que están facturados y se desea que aparezcan marcados con la calilla” Multi-selección” seleccionada. \*/

**IF Albaran.Facturado = TRUE THEN DO:**

**wListItemGrid:BEGIN\_SELECTOR().**

**wListItemGrid:ADD\_SELECTOR("Centro", Albaran.Centro,**DataGateGDP.Models.FilterOperator**:Equal).**

**wListItemGrid:ADD\_SELECTOR("Empresa", Albaran.Tenand,**DataGateGDP.Models.FilterOperator**:Equal).**

**wListItemGrid:END\_SELECTOR().** /\* final de la selección de esta fila \*/

**END.**

**END.** /\* final for each \*/

/\* cargar el DataGrid con las selecciones de marcado. Primera para la selección y segunda para las filas a visualizar \*/

**Data\_State:SetLayoutData(4,"GridAlbaranes",** DataGateGDP.Models.Visibility:NotTouch**, wListItemGrid, wListItemGrid).**

**RECUPERAR DATOS DE UN DATAGRID** (Desde State)

En un control de tipo DataGrid se pueden recuperar los valores de varias maneras:

**Multi-Selección**: Si tiene la multi-selección activada.

**Selección Simple**: Seleccionando una línea que aparece con el icono seleccionado.

**Líneas Editables**: Si alguna de las columnas del DataGrid tiene la marca de “Actualizable”

**MULTI-SELECCIÓN**

Permite recuperar una lista con las filas del DataGrid seleccionados por la selección múltiple.

IF Form\_Data:**GetCharacter**("Nombre del DataGrid” + “\_Action") = "**MultiSelected**" THEN DO:

Selectors = Form\_Data:**GetJsonArray**("Nombre del DataGrid"). /\* pasar el Array a un objeto \*/

DO iCount = 1 TO **Selectors:LENGTH**: /\* número de lineas seleccionadas \*/

Element = Selectors:**GetJsonObject(iCount).** /\* preparer en un JsonObject los valores de las columnas \*/

MyVariable = Element:**GetCharacter**("Nombre del Campo en el formulario"). /\* recupera valores del DataGrid \*/

/\* incorporar aquí el código deseado para esta opción \*/

END.

END.

**SELECCIÓN SIMPLE**

Permite recuperar los valores de la fila que ha sido seleccionada. La información llega en un Objeto:

IF Form\_Data:**GetCharacter**("Nombre del grid” + “\_Action") = "**Selected**" THEN DO:

Element = Form\_Data:**GetJsonObjec**t(“Nombre del Grid”).

/\* recuperar los datos de la línea del grid seleccionada \*/

MyCampo = Element:GetCharacter("Nombre del Campo en el Grid").

END.

Las líneas editables devuelven “**Update**” al modificar una línea del DataGrid o “**New**” al crear una nueva línea, como lo veremos en la sección de

“DATAGRID CON COLUMNAS ACTIVAS.

Líneas Editables: Estas acciones sirven para añadir o actualizar un campo del DataGrid cuando las columnas del Grid están marcadas como  
“Editables”. Para más información revise la sección “DataGrid con Columnas Activas”.

**NEW** 🡪 Se puede recuperar el evento al añadir una nueva línea en el DataGrid.

IF Form\_Data:**GetCharacter**("Nombre del grid” + “\_Action") = "**New**" THEN DO:  
 /\* incorporar el código deseado \*/

END.

**UPDATED 🡪** Se puede recuperar el evento al seleccionar una línea previamente insertada en el DataGrid.

IF Form\_Data:**GetCharacter**("Nombre del grid” + “\_Action") = "**Update**" THEN DO:  
 /\* incorporar el código deseado \*/

END.

**CARGA DE DATOS EN UN COMBO-BOX Y SEARCH-BOX** (Desde Load y State)

Cargar un combo-box o search-box implica la generación de un objeto Json, al igual que en el DataGrid, que será enviado al componente o control del formulario por el SetLayoutData o SetCurrentLayoutData.

El proceso es mucho más simple que el DataGrid.

wListaCarga = **NEW ListItemTools**(). /\* instanciar \*/

FOR EACH Table NO-LOCK:

wObjectValue = "NA". /\* (\*\*\*) indicar el valor que debe visualizarse seleccionado en el combo-box o search-box \*/

/\* el combo-box y search box solo necesita un ADD-ROW por cada registro introducido \*/

wListaCarga:**ADD\_ROW**(“Valor del elemento”, “Descripción a visualizar”, “Icono(\*)”, “Visualización SI/NO(\*\*)”).

/\* (\*) Icono de la libreria de iconos que incorpora DataGateGDP 🡪 DataGateGDP.Models.Icons:NO\_ICON para no indicar icono \*/

(\*\*) Visualización 🡪 TRUE / FALSE normalmente a TRUE \*/

END.

/\* publicar la información en Combo-Box o Search-Box por medio de SetLayoutData o SetCuyrrentLayoutData \*/

Data\_State:**SetCurrentLayoutData**(Numero de Layer, "Nombre del Grid", DataGateGDP.Models.Visibility:nottouch, wObjectValue (\*\*\*), wListItemGrid).

/\* (\*\*\*) Valor a visualizar seleccionado \*/

**MODALIDADES DEL SEARCH-BOX**

El componente SEARCH-BOX pude utilizarse en dos modalidades:

* **INMEMORY** 🡪 Se debe cargar la información tal y como se ha indicado anteriormente y busca la información sobre los datos cargados.
* **REMOTE** 🡪 Permite recoger carácter a carácter el dato introducido y pasa ese valor al procedimiento (.p) del formulario al procedimiento interno  
   SeachBoxState. Se puede cargar el combo-box, como hemos presentado anteriormente, filtrado previamente la información a  
   presentar sin tener tiempos de espera de carga.

Para la modalidad REMOTE, los datos se reciben en el procedimiento SearchBoxState, un ejemplo sería:

WHEN "TestField" THEN DO:

DEF VAR MyCode AS CHARACTER.

DEF VAR RecordNumber AS INT.

RecordNumber = 0.

MyCode = dataAction:**GetSearchValue**(). /\* recuperar el valor en pantalla \*/

IF MyCode <> " " THEN DO:

MyListItemCombo = NEW ListItemTools().

FOR EACH Items WHERE Items.Descripction MATCHES "\*" + wCode + "\*"

NO-LOCK:

MyListItemCombo:**ADD\_ROW**(Items.Code, Items.Description, DataGateGDP.Models.Icons:NO\_ICON, TRUE).

RecordNumber = RecordNumber + 1.

IF RecordNumber > 50 THEN LEAVE.

END.

RELEASE Items NO-ERROR.

END.

dataAction:**SetSearchBoxValue**(MyListItemCombo).

END.

**OTROS METODOS O FUNCIONES DISPONIBLES EN DATAGATEGDP**

**NEXTFORM**

Permite colocar en pantalla el formulario deseado. A partir de dicho momento el formulario se visualiza en pantalla y se ejecuta el LOAD del formulario específico. Ha dos posibilidades:

A.- Ir a un subformulario indicado en el diseñador de un componente o control:

Data\_State:**NextForm**(“Nombre de campo del formulario”).

B.- Ir a un subformulario indicando directamente el número de subformulario.

Data\_State:**NextForm**(Número de subformulario).

**CLEARFORM**

Esta función permite limpiar el subformulario indicado o, si no se indica ningún número de subformulario, todos los subformularios del formulario GDP.  
Se limpian los valores para aquellos componentes o controles que se ha indicado en el diseñador GDP que se tienen que limpiar el valor.

Data\_State:**ClearForm()**. /\* para todos los subformularios o layers \*/

Data\_State:**ClearForm**(Número de subformulario). /\* para un subformulario o layer en concreto \*/

**INFOFIELD**

Permite enviar al usuario un mensaje informativo que aparecerá en el texto de helper asignado al control o componente. Es un mensaje particular de un control o componente.

Data\_State:**InfoField**(“Nombre del control”, “Texto del mensaje”).

**ERRORFIELD**

Similar a la anterior definición, pero es un mensaje de error. Es un mensaje de error particular de un control o componente. Aparecerá en color rojo y no permitirá validar la información en pantalla si se utiliza R-ValidateStdGDP.p

Data\_State:**ErrorField**(“Nombre del control”, “Texto del mensaje”).

**ERRORGLOBAL  
E**ste mensaje de error se presenta al usuario en un cuadro de alerta similar al VIEW-AS ALERT-BOX de ABL en escritorio. Se permite acumular mensajes y a la salida del evento de control del componente se visualizarán todos los mensajes indicados.

Data\_State:**ErrorGlobal**("Mensaje de error").

**TOAST**

Estos mensajes desaparecen de pantalla después del tiempo asignado en el diseñador GDP y sirven para enviar mensajes al usuario, por ejemplo, que la actualización ha sido correcta.

Data\_State:**SetCurrentLayoutData**("Nombre de la variable Toast", DataGateGDP.Models.Visibility:Normal, "Texto del mensaje", ?).

Atención, Si en la definición del formulario GDP ha indicado un texto específico para el mensaje, este texto prevalecerá sobre los indicados en el procedimiento ABL.

**SETCURRENTFOCUS**

Permite llevar el foco del cursor al control o componente indicado si el control está en visibilidad Normal o ReadOnly.

Data\_State:**SetCurrentFocus**("Nombre del control”).

**DATAGRID CON COLUMNAS ACTIVAS**

El DataGrid permite definir columnas activas (Editables) para permitir la entrada de datos por cada línea. Estas columnas también pueden contener una definición de Combo-Box o Search-Box y para lo cual, se ha implementado una serie de funciones o métodos que permiten cargar y recuperar los datos de cada fila, manteniendo el valor anterior al indicado por el usuario en la entrada.

Los botones de “Grabar” y “Cancelar” son colocados automáticamente al final de la fila, al situar el cursor en cualquier columna de la línea marcada, para que se confirme la transacción o retroceda la transacción. En el caso de pulsar “Cancelar” se volverán a visualizar los datos originales. Esta retrocesión es totalmente automática y el desarrollador no debe realizar ninguna acción.

ATENCIÓN- Si está editando una línea del DataGrid no se permite ir a otra línea del DataGrid si no se ha cancelado o grabado la línea que se está editando.

Para seguir con la distribución de tareas y estandarización del código ABL que DataGateGDP lleva a cabo, esta programación se debe realizar en una sub-procedure que se debe denominar ACTIONDATAGRID, como ya hemos indicado anteriormente.

A continuación, explicamos los métodos disponibles en lo que llamamos: ACTIONGRID.

**SINTAXIS PARA ACTIONGRID**

En un procedimiento interno del programa ABL de un formulario cualquiera, podemos crear el procedimiento interno “ACTIONDATAGRID” (Procedure ActionDataGrid:) para recuperar y cargar los valores de cada columna del DataGrid, además de la carga inicial (LOAD) que ya se ha explicado anteriormente, y controlar los valores incorporados por el usuario en un DataGrid. Para que se pueda activar estos eventos las columnas deseadas deben estar marcadas ENABLE en la definición del GDP.

Existen tres (3) operaciones que se notifican al desarrollador para que defina los controles de la información a realizar:

**ADDROW** Permite indicar los valores iniciales de la línea que se va a crear

**VALIDATE** Permite Validar el dato introducido.

**COMBOBOXDATAREQUEST** Permite rellenar con valores el combo-box o search-box en momento de ejecución según los datos registrados en la fila,

aunque no se haya confirmado la finalización de la entrada por medio del botón “Grabar” que aparecerá automáticamente al final de la fila.

***Recuerde que una vez presionado sobre el botón “Grabar” el flujo de datos de DataGateGDP se irá al procedimiento STATE con el tipo de “\_Action”  
en “NEW” o “UPDATE”, según la acción realizada por el usuario, para que pueda definir la actualización requerida.***

Ejemplo:

CASE WHEN “MyFieldColumn” THEN DO: /\* control de la columna MyFieldColumn \*/

DEFINE VARIABLE Myoperation AS CHARACTER.

Myoperation = Data\_Action:**Operation()**.

IF CAPS(Myoperation) = "**VALIDATE**" THEN DO:

/\* Incorporar la validación de la columna \*/

END.

ELSE IF UPPER(Myoperation) = "**ADDROW**" THEN DO:

/\* incorporar el código ABL deseado para la actualización \*/

END.

ELSE IF UPPER(Myoperation) = "**COMBOBOXDATAREQUEST**" THEN DO:

/\* incorporar la validación o carga porque el tipo de campo recibido es Combo-Box o Search-Box \*/

.

END.

END.

En cualquier momento están disponibles los estados introducidos y anteriores de cada columna.

Los métodos o funciones disponibles son:

Para recuperación de los datos de la fila y columna:

MyField = Data\_Action:**GetFieldCurrentValueCharacter**(“My Character Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueCharacter**(“My Character Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueLongchar**(“My Longchar Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueLongchar**(“My Longchar Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueInteger** (“My Integer Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueInteger**(“My Integer Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueDecimal** (“My Decimal Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueDecimal**(“My Decimal Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueLogical** (“My Logical Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueLogical**(“My Logical Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueDate** (“My Date Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueDate**(“My Date Column Name”).

MyField = Data\_Action:**GetFieldCurrentValueDateTime** (“My DateTime Column Name”).

MyPreviousField = Data\_Action:**GetFieldPreviousValueDateTime**(“My DateTime Column Name”).

Para recuperar todos los datos de una fila 🡪 MyObject = Data\_Action:**GetCurrentLayoutDataJsonObject**().

Después se puede extraer los datos de cada campo con MyField = MyObject:**GetCharacter**(“My Column Name”) como se ha presentado anteriormente.

Para presentar al usuario un mensaje 🡪 Data\_Action:**AddMessageError**(My Message).

Para permitir o no la salida de la casilla de la entrada de datos 🡪 Data\_Action:**SetAllowExitToCel**l(MyLogicalField – True/False).

Para cargar los datos necesarios 🡪 Data\_Action: **SetFieldRowValue**(“My Column Name”, “MyField Value Column”).

Para cargar datos en un Combo-Box o Search-Box (devuelve el control a la entrada del campo) 🡪 Data\_Action:**SetComboBoxValue**(“MyListData”).

Para conocer la columna que está enviando la petición se debe utilizar la función 🡪 My Field Column = Data\_Action:**GetFieldLaunchedEvent()**.

Si desde un procedimiento ActionDataGrid se necesita recuperar algún valor del formulario utilice el método:

Form\_State = Data\_Action: **GetLayoutDataJsonObject(**Número de layout o subformulario**).**

/\* y a continuación con GetCharacter(“My Control Name”) y los ya indicados anteriormente podrá recuperar los valores \*/

**TAB CONTROL**

El control TAB permite al usuario obtener diferentes informaciones en un espacio reducido.

El desarrollador puede conocer en todo momento la TAB en donde está el usuario por medio del **STATE** del nombre del control.

WHEN "PanelTab" THEN DO: **/\* nombre del control o componente asignado al TAB Control \*/**

**/\* mensaje de ejemplo. wData siempre devuelve el valor del campo solicitado. Recordemos que es un parámetro de entrada \*/**Data\_State:ErrorGlobal ("TAB Seleccionada" + STRING(wData)).

END.

**EJEMPLOS PRÁCTICOS**

**COMO VISUALIZAR UNA IMAGEN**

En el ejemplo se presenta como cargar la imagen de un artículo y si no existe se presenta una imagen para informar al usuario que no existe imagen asignada a ese código. Este evento se ejecutará en el “STATE” del procedimiento del formulario.

Se puede enviar al control “Imagen” el path y nombre del fichero imagen.

DEFINE VARIABLE wCodigo AS CHARACTER.

DEFINE VARIABLE wImagen AS CHARACTER.

DEFINE VARIABLE Data\_State AS DataState.

DEFINE VARIABLE Form\_Data AS JsonObject.

DEFINE VARIABLE wImg AS DataGateGDP.Tools.DownloadTools.

Data\_State = DataState.

Form\_Data = Data\_State:GetCurrentLayoutDataJsonObject(). /\* estas dos lineas van al inicio de la procedure Load / State / … \*/

WHEN "Codigo" THEN DO:

/\* se recupera el valor de la variable “Codigo” del formulario en un campo de trabajo \*/

wCodigo = Form\_Data:GetCharacter("Codigo").

/\* se lee el articulo para comprobar si existe \*/

FIND Articulo WHERE Articulo.Codigo = wCodigo NO-LOCK NO-ERROR.

IF AVAILABLE Articulo THEN DO:

wImagen = "F:\DatosUsuarios\Imagenes\REVISADAS\" + TRIM(wCodigo) + ".jpg".

IF SEARCH(wImagen) = ? THEN wImagen = "C:\inetpub\wwwroot\CodigoSinImagen.jpg".

wImg = **New DataGateGDP.Tools.DownloadTools(DataGateGDP.Models.DownloadTypes**:**File**,wImagen). **/\* instanciar campo para enviar a la imagen \*/**

**/\* FILE porque se le envia un fichero . Valores que se pueden enviar: Base64 y Url \*/**

**Data\_State:SetCurrentLayoutData("Imagen", /\* nombre del control de imagen en el formulario \*/**

**DataGateGDP.Models.Visibility:NotTouch, /\* no modificar el enable del control imagen \*/**

**“”, /\* path y nombre de la imagen \*/**

**wImg).** **/\* no se envia detalle con múltiples campos \*/**

END.

ELSE DO:

/\* enviar mensaje al usuario conforme el articulo no está en el fichero \*/

Data\_State:ErrorGlobal("No Existe el Registro del Articulo").

END.

END.

Para convertir la imagen a base64 se puede utilizar la siguiente funcion:

**FUNCTION Base64Image RETURN LONGCHAR (INPUT MyFile AS CHARACTER) FORWARD**. **/\* Definición de la Función \*/**

MyFile = "C:\inetpub\wwwroot\CodigoSinImagen.jpg". **/\* path y nombre de la imagen \*/**

**FUNCTION Base64Image RETURN LONGCHAR (INPUT MyFile AS CHARACTER): /\* funcion \*/**

DEFINE VARIABLE encdmptr AS MEMPTR NO-UNDO.

**COPY-LOB FROM FILE** MyFile **TO** encdmptr **NO-ERROR.**

**RETURN BASE64-ENCODE(**encdmptr**).**

**END FUNCTION.**

**COMO CARGAR UN SEARCH-BOX / COMBO-BOX**

Como se ha comentado anteriormente, hay dos modalidades de trabajo del Search-Box que debe ser seleccionado en la definición del formulario.

**MODALIDAD INMEMORY** 🡪 Se cargan todos los registros al Search-Box o Combo-Box para que el usuario pueda seleccionar el registro entre la lista cargada. Si el volumen de registros que se cargan al Search-Box es elevado, la búsqueda puede ser más lenta que la opción “REMOTE”. Para el Combo-Box la modalidad “REMOTE” no es aplicable.

Esta modalidad se utiliza normalmente en la sección de “Load” del procedimiento de formulario.

DEFINE VARIABLE wListaCarga AS ListItemTools.

DEFINE VARIABLE Data\_State AS DataState.

DEFINE VARIABLE wObjectValue AS CHARACTER.

DEFINE VARIABLE wNumReg AS INT. /\* Contador de registros cargados para saber si hay que desactivar o no el Combo-Box al solo existir un registro \*/

WHEN "ComboSearchControl" THEN DO:

wObjectValue = “”.

wNumReg = 0.

wListaCarga = NEW ListItemTools().

FOR EACH Items NO-LOCK:

/\* contar los registros \*/

wNumReg = wNumReg + 1.

/\* cargar una variable con el valor que se desea quede preestablecido al visualizar el control o componente Search-Box / Combo-Box \*/

IF wObjectvalue = “” THEN wObjectValue = Items.Code.

/\* cargar la lista de elementos a pasar al control Combo-Box o Search-Box \*/

**wListaCarga:ADD\_ROW(Items.Code, /\* código del elemento a recuperar – no se visualiza al usuario \*/  
 Items.Description, /\* descripción a visualizar al usuario \*/  
 DataGateGDP.Models.Icons:NO\_ICON, /\* o cualquier icono de la lista de iconos (\*\*) \*/   
 TRUE). /\* visible = TRUE or FALSE \*/**

END.

RELEASE Items NO-ERROR.

**IF wNumReg > 1 THEN**

**Data\_State:SetCurrentLayoutData(“ComboSearchControl”, /\* nombre del control definido en el formulario GDP \*/**

**DataGateGDP.Models.Visibility:Normal, /\* tipo de visibilidad deseada \*/**

**wObjectValue, /\* valor inicial que se desea que se visualice \*/**

**wListaCarga). /\* lista de valores a cargar en el Combo-Box o Search-Box \*/**

**ELSE Data\_State:SetCurrentLayoutData(“ComboSearchControl”, /\* nombre del control definido en el formulario GDP \*/  
 DataGateGDP.Models.Visibility:Disabled, /\*porque solo hay un registro cargado se desactiva \*/  
 wObjectValue, /\* valor inicial que se desea que se visualice seleccionado \*/  
 wListaCarga). /\* lista de valores a cargar en el Combo-Box o Search-Box \*/**

END.

**MODALIDAD REMOTE** 🡪 Permite recoger carácter a carácter el dato introducido y pasa ese valor al procedimiento (.p) del formulario al procedimiento  
 interno SeachBoxState. En esta opción NO SE PUEDE SELECCIONAR UN ELEMENTO en la carga de la información. Si no se controlan el numero de registros  
 que se carga, el programa presenta los 100 primeros registros cargados

ATENCION!!, Para la modalidad REMOTE, los datos se reciben en el procedimiento **SearchBoxState**, un ejemplo sería:

DEFINE VARIABLE wListaCarga AS ListItemTools.

DEFINE VARIABLE DataAction AS ActionSearchBox.

DEFINE VARIABLE wMyCode AS CHARACTER.

DEFINE VARIABLE wNumReg AS INT. /\* Contador de registros cargados \*/

DEFINE VARIABLE Form\_Data AS JsonObject.

WHEN "SearchBoxControl" THEN DO:

wNumReg = 0.

wMyCode = DataAction:**GetSearchValue**(). /\* recuperar el valor en pantalla \*/

IF wMyCode <> " " THEN DO:

wListaCarga = NEW ListItemTools().

FOR EACH Items WHERE Items.Descripction MATCHES "\*" + wMyCode + "\*"

NO-LOCK:

**wListaCarga:ADD\_ROW(Items.Code, /\* código del elemento a recuperar – no se visualiza al usuario \*/  
 Items.Description, /\* descripción a visualizar al usuario \*/  
 DataGateGDP.Models.Icons:NO\_ICON, /\* o cualquier icono de la lista de iconos (\*\*) \*/   
 TRUE). /\* visible = TRUE or FALSE \*/**

wNumReg = wNumReg + 1.

IF wNumReg > 50 THEN LEAVE. **/\* cargar un cierto número de elementos a presenter en el SearchBox (max. 100) \*/**

END.

RELEASE Items NO-ERROR.

END.

**DataAction:SetSearchBoxValue(wListaCarga).**

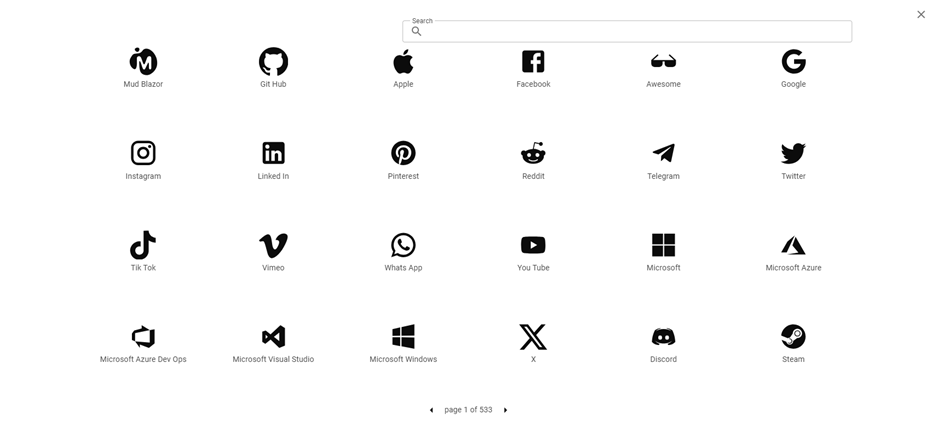
END.

Para recuperar valores de otros layouts en el procedimiento **SearchBoxState** debe utilizar, como en los otros procedimientos, las siguientes funciones:

Form\_Data = **GetCurrentLayoutDataJsonObject**().

Form\_Data = **GetLayoutDataJsonObject** ( LayoutId /\* Numero de Layer – Integer \*/ ).

(\*\*) Los nombres de los iconos que se pueden incorporar aparecen al solicitar la lista de iconos en cualquier control o componente.



**PERMITIR INCORPORAR UN NUEVO ELEMENTO NO CARGADO EN UN SearchBox (Solo disponible para el SearchBox)**

En la definición del formulario para un **SearchBox** se permite indicar “**Coerce Values**” a valor **TRUE** (por defecto) o **FALSE**.

****

Si se indica **TRUE** (por defecto) no se permite al usuario escribir un nuevo elemento dentro del la lista de valores que tiene el SearchBox cargada.

Si se cambia a **FALSE**, el usuario podrá escribir un nuevo elemento y al salir el foco del SearchBox, se devuelve el valor indicado por el usuario al procedimiento STATE del campo y se puede validar el valor conforme que no está creado en la base de datos para que el desarrollador decida la acción a reaiizar.

Por ejemplo: preguntar al usuario, por medio de un AlertBox, si desea dar de alta el nuevo elemento y grabarlo. A continuación, si se quiere que quede registrado en el SearchBox habrá que recargar el SearchBox.

**COMO CARGAR LOS DATOS EN UN DATAGRID**

Carga Simple 🡪 El sistema de implementación es más sencillo, aunque es más lento de ejecución. Carga sobre 1.060 registros x segundo en un grid de 30 columnas + Style para un WorkGroup database (utiliza un solo núcleo).

Carga Rápida 🡪 El sistema de implementación es un poco más compleja, aunque la carga ejecución de mucho más rápida. Carga sobre 1.600 registros x Segundo en un Grid con 30 columnas y sus estilos CSS correspondientes. Este tiempo está calculado para un WorkGroup DataBase (utiliza un solo núcleo).

**CARGA SIMPLE**

DEFINE VARIABLE wListItemGrid AS GridTools NO-UNDO.

wListItemGrid = Data\_State:**Create\_Grid**(1, /\* número de layout en donde está el DataGrid\*/   
 “ListaClientes”). /\* Nombre asignado al DataGrid” - se instancia el DataGrid \*/

FOR EACH Table NO-LOCK:

wListItemGrid:**BEGIN\_ROW**(). /\* Inicio de fila \*/

wListItemGrid:**ADD\_COLUMN\_DATA**("Nombre Columna", Table.MyField).

/\* para incorporar un campo con Style se ha incorporado “color:Red” \*/

wListItemGrid**:ADD\_COLUMN\_DATA**("Nombre Columna", Table.MyField, "color:red").

/\* más columnas ….. \*/

/\* No es necesario indicar el tipo de dato que se envía y si no se desea cambiar el estilo, no se es necesario enviar el parámetro de estilo \*/

/\* si la columna no existe en el DataGrid se devolver error en tiempo de ejecución (Nombres con CASE SENSITIVE) \*/

wListItemGrid:**END\_ROW()**. /\* notificar el final de fila \*/

END.

/\* publicar la información enviada al DataGrid por medio de SetLayoutData o SetCuyrrentLayoutData \*/

**Data\_State:SetLayoutData(Numero de Layer, /\* numero del layout en donde está ubicado el DataGrid \*/   
 "ListaClientes", /\* nombre del DataGrid que se está cargando \*/  
 DataGateGDP.Models.Visibility:normal, /\* asignación de la visibilidad -Normal, Disabled,.. \*/  
 "?", /\* Con este valor se le asigna el valor NULL a la variable y así no se realice ningún cambio de valor\*/  
 wListItemGrid). /\* Lista de Valores que se cargan en el dataGrid \*/**

**CARGA RÁPIDA**

DEFINE VARIABLE wListItemGrid AS GridTools NO-UNDO.

DEFINE VARIABLE hBuffer AS HANDLE NO-UNDO.

wListItemGrid = Data\_State:**Create\_Grid**(1, /\* número de layout en donde está el DataGrid\*/   
 “ListaClientes”). /\* Nombre asignado al DataGrid” - se instancia el DataGrid \*/

hBuffer = wListItemGrid:**GET\_HANDLE**(). /\* recuperar el handle del DataGrid \*/

FOR EACH Table NO-LOCK:

hBuffer:**BUFFER-CREATE** (). /\* Inicio de fila \*/

hField = hBuffer:**BUFFER-FIELD**("Nombre Columna"). /\* se recupera el Handle de la columna en hField \*/

hField:**BUFFER-VALUE** = Table.MyField. /\* carga del valor \*/

hField = hBuffer:**BUFFER-FIELD**(("Nombre Columna" + “\_STYLE"). /\* incorporar \_STYLE al nombre del campo si se desea cambiar\*/

hField:**BUFFER-VALUE** = "color:red". /\* cargar el valor asignado a la CSS de la columna. No es obligatorio \*/

/\* más columnas ….. \*/

/\* No es necesario indicar el tipo de dato que se envía y si no se desea cambiar el estilo, no se es necesario enviar el parámetro de estilo \*/

/\* si la columna no existe en el DataGrid se devolver error en tiempo de ejecución (Nombres con CASE SENSITIVE) \*/

hBuffer:**BUFFER-RELEASE**(). /\* notificar el final de la fila \*/

END.

/\* publicar la información enviada al DataGrid por medio de SetLayoutData o SetCuyrrentLayoutData \*/

**Data\_State:SetLayoutData(Numero de Layer, /\* numero del layout en donde está ubicado el DataGrid \*/   
 "ListaClientes", /\* nombre del DataGrid que se está cargando \*/  
 DataGateGDP.Models.Visibility:normal, /\* asignación de la visibilidad -Normal, Disabled,.. \*/  
 "?", /\* Con este valor se le asigna el valor NULL a la variable y así no se realice ningún cambio de valor\*/  
 wListItemGrid). /\* Lista de Valores que se cargan en el dataGrid \*/**

**COMO RECUPERAR DATOS DE UN DATAGRID**

Como se ha indicado anteriormente, en un control de tipo DataGrid se pueden recuperar los valores según se hayan activado las opciones en las propiedades del DataGrid:

|  |  |
| --- | --- |
|  | **Líneas Editables**: Si alguna de las columnas del DataGrid tiene la marca de “Editable”. Se tiene el control de la línea por el procedimiento **ActionDataGrid** y la grabación se debe realizar en elprocedimiento **STATE**  **Selección Simple**: Seleccionando una línea que aparece con el icono seleccionado. Se tiene el control por el procedimiento **STATE**  **Multi-Selección**: Se recupera la lista de líneas marcadas.  Se tiene el control por el procedimiento **STATE** |

**LINEAS EDITABLES**

Estas acciones sirven para añadir o actualizar columnas del DataGrid cuando dichas columnas están marcadas como “Editables”.



Al activar el indicador “EnableAdding” aparece en la parte inferior del DataGrid el signo + y al pulsarlo aparece al final una nueva línea coloreada en verde para que se pueda introducir los datos. Así mismo al final de la línea se presenta el botón de grabar (los estados se verán a continuación) y el botón de cancelar para que se pueda retroceder el registro. Este botón es totalmente automático y eliminará la línea que se estaba creando.

Mientras se crea una nueva línea, si se sale de la línea se eliminan las modificaciones realizadas automáticamente.

Desde el procedimiento **STATE** al finalizar la entrada de información (al pulsar el usuario el botón de Salvar) se va a poder recuperar los valores de todas las columnas o celdas para poder grabar la información.

Desde el procedimiento **ActionDataGrid** se pueden recuperar y modificar los valores de cada celda de la línea mientras se está creando o modificando.

***CREACION DE UNA NUEVA LINEA:*** Se recibe **NEW** en el procedimiento **STATE** cuando el usuario pulsa el botón GRABAR. Se recibe un JsonObject con los datos introducidos en la línea.

DEFINE VARIABLE Form\_Data AS JsonObject.  
 DEFINE VARIABLE Element AS JsonObject.  
 DEFINE VARIABLE wCodEjeEm AS INTEGER.

IF Form\_Data:**GetCharacter**("Nombre del DataGrid” + **“\_Action**") = "**New**" THEN DO:  
 /\* primero recuperar los valores de las columnas \*/  
 Element = Form\_Data:GetJsonObject("ListaClientes”). /\* Nombre del DataGrid \*/

/\* recuperar los datos uno a uno \*/  
wCodEjeEm = Element:GetInteger("CodEjeEm"). /\* nombre de la columna del DataGrid \*/

END.

***MODIFICACION DE UNA CASILLA:*** Se recibe **UPDATED** en el procedimiento **STATE**, para indicar que el usuario ha modificado un valor de la línea.   
Si se ha creado una línea se indica: NEW y nunca UPDATED.

Se recibe un JsonObject con los datos introducidos en la línea. Es similar al NEW pero solicitando “Updated”.

DEFINE VARIABLE Form\_Data AS JsonObject.  
 DEFINE VARIABLE Element AS JsonObject.  
 DEFINE VARIABLE wCodEjercicio AS INTEGER.

IF Form\_Data:**GetCharacter**("Nombre del DataGrid” + **“\_Action**") = "**Updated**" THEN DO:  
 /\* primero recuperar los valores de las columnas \*/  
 Element = Form\_Data:GetJsonObject("ListaClientes”). /\* Nombre del DataGrid \*/

/\* recuperar los datos uno a uno \*/  
wCodEjercicio = Element:GetInteger("CodEjeEm"). /\* nombre de la columna del DataGrid \*/

END.

**OPERACIONES EN LA EDICION DE LA LÍNEA.**

Al crear o editar líneas se pueden inicializar, recuperar y alterar los valores de cada una de las columnas. Todo lo explicado a continuación se debe realizar en el procedimiento **ActionDataGrid**.

*OPERACIONES:* Las operaciones informan al programa de donde está el cursor. Existe las siguientes operaciones:

* + **ADDROW** Indicando que se está en una columna para poder cargar valores iniciales de cualquier columna.
  + **VALIDATE** Se está saliendo de una columna para validar la información introducida / seleccionada por el usuario.
  + **COMBOBOXDATAREQUEST**: Se indica que se esta entrando a un ComboBox y se pueden rellenar los datos. Para activar esta característica en la definición de la columna se debe marcar el atributo “**Is ComboBox**” y el tipo de campo que se visualizará en la línea será un combo-box que podrá ser cargado su lista de valores. No se permiten SearchBox.



**EVENTOS O FUNCIONES PERMITIDAS**

Conocer el nombre de la columna desde donde se lanza el evento:

wMyCellName = **ActionGrid:GetFieldLaunchedEvent().**

Conocer el valor de la columna de donde se está saliendo:

wMyCharacterCell = **ActionGrid:GetFieldCurrentValueCharacter().**

wMyLongCharCell = **ActionGrid: GetFieldCurrentValueLongChar().**

wMyIntegerCell = **ActionGrid:GetFieldCurrentValueInteger().**

wMyDecimaCell = **ActionGrid:GetFieldCurrentValueDecimal().**

wMyDateCell = **ActionGrid:GetFieldCurrentValueDate().**

wMyDateTimeCell = **ActionGrid:GetFieldCurrentValueDateTime().**

wMyLogicalCell = **ActionGrid:GetFieldCurrentValueLogical().**

Conocer el valor anterior de la columna de donde se está saliendo antes del cambio:

wMyCharacterCell = **ActionGrid:GetFieldPreviousValueCharacter().**

wMyLongCharCell = **ActionGrid:GetFieldPreviousValueLongChar().**

wMyIntegerCell = **ActionGrid:GetFieldPreviousValueInteger().**

wMyDecimalCell = **ActionGrid:GetFieldPreviousValueDecimal().**

wMyDateCell = **ActionGrid:GetFieldPreviousValueDate().**

wMyDateTimeCell = **ActionGrid:GetFieldPreviousValueDateTime().**

wMyLogicalCell = **ActionGrid:GetFieldPreviousValueLogical().**

Permitir salir o no de la celda:

**ActionGrid:SetAllowExitToCell(TRUE)** 🡪 Permitir salir

**ActionGrid:SetAllowExitToCell(FALSE)** 🡪 NO Permitir salir

Visualizar mensaje de error o advertencia:

**ActionGrid:AddMessageError(“Message”).**

Para error (se visualiza en Rojo) **SetAllowExitToCell** debe estar a TRUE

Para Advertencia (se visualiza el Ambar) **SwtAllowExitToCell** debe estar a FALSE

Recuperar los valores de las columnas de la línea que se está editando en el DataGrid:

wMyCharacterCell = **ActionGrid:GetFieldRowValueCharacter(“ColumnName“).**

wMyLongCharCell = **ActionGrid:GetFieldRowValueLongChar(“ColumnName“).**

wMyIntegerCell = **ActionGrid:GetFieldRowValueInteger(“ColumnName“).**

wMyDecimalCell = **ActionGrid:GetFieldRowValueDecimal(“ColumnName“).**

wMyDateCell = **ActionGrid:GetFieldRowValueDate(“ColumnName“).**

wMyDateTimeCell= **ActionGrid:GetFieldRowValueDateTime(“ColumnName“).**

wMyLogicalCell = **ActionGrid:GetFieldRowValueLogical(“ColumnName“).**

Cargar valor en la celda correspondiente.

**ActionGrid:SetFieldRowValue (“ColumnName“, NewValue).**

Cargar un Combo-Box en un DataGrid:

DEFINE VARIABLE wListaCarga AS ListItemTools.

DEFINE VARIABLE Data\_State AS DataState.

DEFINE VARIABLE wObjectValue AS CHARACTER.

WHEN "ComboBoxControl" THEN DO:

wObjectValue = “”.

wListaCarga = NEW ListItemTools().

FOR EACH Items NO-LOCK:

/\* cargar una variable con el valor que se desea quede como valor inicial en el control o componente Combo-Box \*/

IF wObjectvalue = “” THEN wObjectValue = Items.Code.

/\* cargar la lista de elementos a pasar al control Combo-Box o Search-Box \*/

**wListaCarga:ADD\_ROW(Items.Code, /\* código del elemento a recuperar – no se visualiza al usuario \*/  
 Items.Description, /\* descripción a visualizar al usuario \*/  
 DataGateGDP.Models.Icons:NO\_ICON, /\* o cualquier icono de la lista de iconos (\*\*) \*/   
 TRUE). /\* visible = TRUE or FALSE \*/**

END.  
 RELEASE Items NO-ERROR.  
 **ActionGrid**:**SetComboBoxValue (wListaCarga). /\* carga la lista de valores en el Combo-Box \*/   
 ActionGrid:SetFieldRowValue (wOjectValue). /\* presentar la usuario el valor inicial \*/**

END.

**¿COMO RECUPERAR LOS VALORES DE LOS OTROS CAMPOS DE LOS FORMULARIOS EXISTENTES?**

Es normal que al validar la información dentro del procedimiento **ACTIONDATAGRID** se necesiten valores de otros campos de los formularios del programa. Estos valores se pueden recuperar de la siguiente manera:

Para el Layout activo: **Action\_DataGrid:GetCurrentLayoutDataJsonObject()** 🡪 recupera todos los valores del Layout activo (current). A continuación, se puede extraer cualquiera de los campos por su nombre del campo.

Ejemplo: DEFINE VARIABLE Action\_DataGrid AS ActionDataGrid.

DEFINE VARIABLE Form\_Data AS JsonObject.

DEFINE VARIABLE wCodEjercicio AS Integer.

Form\_Data = **Action\_DataGrid:GetCurrentLayoutDataJsonObject().** /\* recuperamos todos los datos del layout Corriente \*/

wCodEjercicio = Form\_Data:**GetInteger**("Nombre del Campo en el Formulario"). /\* sintaxis de OpenEdge estándar \*/

Para cualquier Layout del procedimiento (.p): **Action\_DataGrid:GetLayoutDataJsonObject(Número de Layout)** 🡪 recupera todos los valores del Layout indicado entre parentesis. A continuación, se puede extraer cualquiera de los campos por su nombre del campo.

Ejemplo: DEFINE VARIABLE Action\_DataGrid AS ActionDataGrid.

DEFINE VARIABLE Form\_Data AS JsonObject.

DEFINE VARIABLE wCodEjercicio AS Integer.

Form\_Data = **Action\_DataGrid:GetLayoutDataJsonObject(1 ).** /\* recuperamos todos los datos del layout 1 \*/

wCodEjercicio = Form\_Data:**GetInteger**("Nombre del Campo en el Formulario"). /\* sintaxis de OpenEdge estándar \*/

**¿COMO RECUPERAR DE UN DATAGRID EL VALOR DE LA FILA SELECCIONADA?**

**FILAS DE SELECCIÓN SIMPLE**

Imagen que contiene Gráfico

Descripción generada automáticamente

Permite recuperar los valores de la fila que ha sido seleccionada.

La información llega en un Objeto:

DEFINE VARIABLE Form\_Data AS JsonObject.  
 DEFINE VARIABLE Element AS JsonObject.  
 DEFINE VARIABLE wCodEjeEm AS INTEGER.

IF Form\_Data:**GetCharacter**("Nombre del grid” + **“\_Action**") = "**Selected**" THEN DO:

Element = Form\_Data:**GetJsonObject**(“Nombre del Grid”).

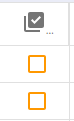
/\* recuperar los datos de la línea del grid seleccionada \*/

wCodEjeEm = Element:**GetInteger**("CodEjeEm").

… /\* recuperar el resto de columnas e incorporar la lógica deseada \*/

END.

**MULTI-SELECCIÓN DE FILAS**

Permite recuperar una lista con las filas del DataGrid seleccionados por la selección múltiple. En este caso no se recibe un JsonObject, sino un JsonArray porque entrega todos los registros marcados por el usuario.

DEFINE VARIABLE Form\_Data AS JsonObject.  
 DEFINE VARIABLE Element AS JsonObject.  
 DEFINE VARIABLE wCodEjeEm AS INTEGER.

IF Form\_Data:**GetCharacter**("Nombre del DataGrid” + “**\_Action**") = "**MultiSelected**" THEN DO:

Selectors = Form\_Data:**GetJsonArray**("Nombre del DataGrid"). /\* pasar el Array a un objeto \*/

DO iCount = 1 TO **Selectors:LENGTH**: /\* leer el número de lineas seleccionadas \*/

Element = Selectors:**GetJsonObject(iCount).** /\* preparer en un JsonObject los valores de las columnas \*/

MyVariable = Element:**GetInteger**(CodEjeEm). /\* recupera valores del DataGrid \*/

/\* incorporar aquí el código deseado para esta opción \*/

END.

END.

Para la selección múltiple se le pueden enviar líneas preseleccionadas como veremos a continuación.

**MARCAR FILAS COMO SELECCIONADOS PARA MULTI-SELECCION**

Se pueden marcar las filas que se deseen y aparecerá el indicador siguiente pre-seleccionado

Se le deben enviar los siguientes parámetros:

* + *Nombre de la Columna*
  + *Valor de la Columna*
  + *Operador de búsqueda para la columna*

para encontrar los registros del DataGrid que cumplan los valores y que el DataGrid los marque en Multi-Selección.

Al momento de realizar la carga del DataGrid con la información de todas las filas y columnas se puede enviar los filtros para la marca de las filas en el registro deseado. También se puede enviar después de haber cargados los valores en el DataGrid, como veremos a continuación.

Los parámetros de selección pueden ser diferente en cada solicitud.

La carga de los parámetros para marcar la Multi-Seleccion del DataGrid se realiza de la siguiente manera.

DEFINE VARIABLE Grid1 AS **GridTools**. **/\* esta variable prepara un objeto Json \*/**

Grid1 = Data\_State: **Create\_Grid** ("Nombre del Grid").

**/\* cargar de los parámetros de selección \*/**

Grid1:**BEGIN\_SELECTOR()**. **/\* iniciar el registro de selección de las filas \*/**

Grid1:**ADD\_SELECTOR** ("CodCliente", **/\* nombre de la columna por la cual se desea seleccionar \*/**

"0235", **/\* valor de la columna a seleccionar \*/**

FilterOperator:Equal). **/\* definición del tipo de filtro a utilizar en la selección \*/**

Ggrid1:**ADD\_SELECTOR**("Fecha", "", FilterOperator:IsNotNull).

….. **/\* más filtros** \*/

Grid1:**END\_SELECTOR()**.

Los valores de operadores admitidos son:

|  |  |
| --- | --- |
| **None**  **Contains**  **StartsWith**  **EndsWith**  **Equal**  **NotEqual**  **GreaterThan**  **GreaterThanOrEqual** | **LessThan**  **LessThanOrEqual**  **IsNull**  **IsNotNull**  **IsEmpty**  **IsNotEmpty**  **DoesnotStartWith**  **DoesNotContain**  **DoesNotEndWith** |

**REPOSICIONAR EN UNA FILA (COLOCARSE EN CUALQUIER FILA DEL DATAGRID)**

La reposición consiste en situarse a la fila solicitada en un DataGrid.

Se le deben enviar los valores de las columnas que se deseen para encontrar el registro que cumplan los valores y que el DataGrid se reposicione en esa fila.

La reposición permite poder enviar uno o varias columnas a la vez como parámetros. Siempre se posiciona en la primera fila que cumple los parámetros recibidos.

Al momento de realizar la carga del DataGrid con la información de todas las filas y columnas se puede enviar los filtros para el reposicionamiento en el registro deseado.

Las columnas de selección y los valores de selección pueden ser diferente en cada solicitud.

La carga de los parámetros para reposición del DataGrid se realiza de la siguiente manera.

DEFINE VARIABLE Grid1 AS **GridTools**. **/\* esta variable prepara un objeto Json \*/**

Grid1 = Data\_State: **Create\_Grid** ("Nombre del Grid").

**/\* carga de los parámetros de reposicionamiento \*/**

Grid1:**BEGIN\_REPOSITION().**

Grid1:**ADD\_REPOSITION**("CodCliente", **/\* Nombre de la columna \*/**

"0235"). **/\* valor de la columna en donde se debe reposicionar \*/**

Grid1:**ADD\_REPOSITION**("Fecha", **/\* Nombre de la columna \*/**

"20241210"). **/\* valor de la columna en donde se debe reposicionar \*/**

**…. /\* más valores del registro a exclusivo a seleccionar. Tantos como sean necesarios \*/**

Grid1:**END\_REPOSITION()**. **/\* final seccion definición del reposicionamiento \*/**

Data\_State:SetCurrentLayoutData ("Nombre del Grid", **/\* nombre del control DataGrid \*/**

DataGateGDP.Models.Visibility:Normal, **/\* visibilidad \*/**

Grid1, **/\* nombre del objecto con los parámetros de reposicionamiento \*/**

?). **/\* nombre del objeto que incorpora los valores del DataGrid.   
 si se le envía “interrogante” es que ya tiene los datos cargados y solo se recibirá el reposicionamiento \*/**

**VENTANAS MODALES EN DATAGATE GDP**

En DataGateGDP ee ha incorporado la opción de abrir ventanas modales que pueden incorporar en su interior:

* **Un Layout definido en el mismo formulario.**
* **Un Layout de otro procedimiento externo (.p).**

Cada control de ventana (Window) solamente puede incorporar un formulario o procedimiento (.p) externo.

Se genera una venta “modal”. Es decir, hasta que no se cierre la ventana no se permite pasar a otro formulario o a otra opción del menú.

Interfaz de usuario gráfica

Descripción generada automáticamente



Recordemos que las propiedades del control “Window”, ya presentadas en el manual Designer V 3.6, permiten:

* + Definir Altura y Anchura. Puede indicar parámetros CSS para su definición.
  + Incorporar la opción de cerrar la ventana.
  + Permitir Barras de desplazamiento (scroll) automáticas o no.
  + Indicar el programa y el layout a incorporar en su interior.

Con la opción “This” se le indica que es un Layout del mismo formulario.

* + Se puede indicar el Padding que se desea también en el panel de propiedades.

En caso de que se indique otro programa diferente al que se está ejecutando se le pueden pasar

y recoger valores (parámetros) entre los dos procedimientos, como a continuación explicamos.

Indicar que como medida de seguridad se genera un GUID diferente para los casos que “Target Window”  
no sea “This”, por lo tanto, solamente hay comunicación entre los dos programas (.p) por medio  
de parámetros.

**PASO DE PARAMETROS ENTRE LAYOUT EN UN MISMO PROCEDIMIENTO (.p)**

Si es el mismo procedimiento y diferente Layout no existen paso de parámetros porque funciona como si no estuviera en una ventana y todo lo explicado anteriormente en este manual está disponible para el desarrollador.

**PASO DE PARAMETROS ENTRE PROCEDIMIENTOS (.p)**

Los valores a “recoger” o “cargar” entre los dos formularios se realiza directamente actualizando los valores de las variables definidas en los formularios.

Ejemplo: Desde el formulario “A” vamos a cargar los valores de los campos “CodEjeEm” (Ejercicio) y “Description” (Descripción)

que está en el formulario “B”.

DEFINE VARIABLE Data\_State AS DataState.

DEFINE VARIABLE wVisibility AS DataGateGDP.Models.Visibility.

DEFINE VARIABLE parameters AS **BundleParameters**. **/\* definición especial para el traspaso entre formularios \*/**

**parameters = NEW BundleParameters().** **/\* se instancia \*/**

**parameters:SetParameter("CodEjeEm",2020).** **/\* se carga el nombre del campo del formulario “B” y el valor que deseamos pasar (2020) \*/**

**parameters:SetParameter("Description","Test"). /\* tantos parámetros como sea necesario \*/**

wVisibility = DataGateGDP.Models.Visibility:NotTouch. **/\* se puede cambiar la visibilidad del campo receptor del dato de Normal a Disabled,… \*/**

**/\* enviar los datos a procedimiento (.p) y layout definidos en los parámetros del control “Window1” en el formulario “A”**

**Se utiliza la instrucción SetCurrentLayoutData o se puede utilizar SetLayoutData indicando el layout como si estuviera en el mismo  
 formulario \*/**

**Data\_State:SetCurrentLayoutData("Window1", /\* Nombre del control Window \*/**

**wVisibility , /\* visibilidad deseada para todos los campos que se envían en los parámetros \*/**

**“?”, /\* este valor no es necesario enviarlo porque Window1 no admite valores \*/**

**parameters). /\* Lista de campos y valores a cargar \*/**

**KEYCAPTURE**

El control KeyCapture nace de la necesidad de controlar teclas de función o automatizar algunos juegos de teclas para ayudar a las tareas del usuario, ya que los navegadores no lo incorporan.

Este control funciona exclusivamente para dispositivos PC / Computadoras.

Al definir el control “KeyCapture” deberá indicar un nombre al control. En el procedimiento **STATE** tendrá el control cuando pulsen la techa indicada en la definición del formulario.

La definición de teclas que se desean activar en cada uno de los “layouts” del formulario debe indicarse en el panel de propiedades del control “KeyCapture” dentro de “Source” con el siguiente formato:

**{F1}{F3}{F4}{MediaPlayPause}{ScrollLock}{Ctrl|Alt|KeyA}{Ctrl|F1}**

**/\* teclas activadas F1, F3, F4, Play/Pause, ScrollLock, Cltr+Alt+A, Cltr+F1 \*/**

Es decir, se van incorporando en un string separados por “**}**” “**{**“, aunque lo normal es incorporar por cada “KeyCapture” una sola tecla para efectuar la acción requerida.

Ejemplo: supongamos que definimos en el control “KeyCapture” denominado “**FuncionF1**” la techa asociada en el Source **{F1}**. Pues en el procedimiento **STATE** deberemos incorporar el siguiente código:

**WHEN "FuncionF1" THEN DO:**

**Data\_State:ErrorGlobal("Tecla Pulsada F1"). /\* mensaje de ejemplo. \*/**

**END.**