(For the latest version of this document, please visit the Lianja Wiki .)

In the beginning there is data. Without data we have nothing to build LOB (Line Of Business) Apps against.

The primary goal of Lianja is to provide an Enterprise RAD platform for Desktop, Web and Mobile Apps.

When building large enterprise class applications with 100's of tables that are used in multiple Apps and their associated Lianja UI elements, it can be tedious and error prone to apply the same section, field and grid column attributes consistently throughout the UI using the attributes panel in the App Builder.

To alleviate this problem and to speed up app development we have extended the Lianja App Builder and the Lianja engine to handle metadata with a **MetaData API**. This mechanism provides a way of dynamically setting up UI element attributes when a new section is created or when an app is loaded and its pages and their corresponding sections are loaded.

Also, and probably more importantly, any changes to the metadata for the tables and/or their columns contained in the database schema that are used in many different Lianja apps and their UI elements (Sections, Fields, Grid Columns) are propagated throughout apps dynamically as they are loaded at design time.

In Lianja 3.4 we have added a MetaData Editor for databases, tables and columns. This can be activated by clicking the edit icon in the database, table or columns panel in the "Data" workspace.

This standard metadata format should be used if you want to edit using the MetaData Editor.

```
// create an object from a metadata string
```

obj = metadata_decode("name=barry;company=Lianja;amount=10") // encode a string from an object representing metadata
str = metadata_encode(obj)

Tip: When specifying MetaData names they can be in any format you want:

this.is.a.name=value

The MetaData names can also be applied conditionally (note how OData expression syntax is used):

section.backcolor=[amount lt 1000]red

section.backcolor=[amount ge 1000]lightgreen

The "true" and "false" selections can be combined:

section.backcolor=[amount lt 1000]red,lightgreen

When data is refreshed in a section, the table metadata is read and if any conditional metadata exists it is evaluated and applied to the section.

The same procedure is then performed on formitems (column MetaData is read) which should be in this format.

formitem.backcolor=[amount lt 1000]red

Note that Lianja 3.4 introduced a new Lianja system object method Lianja.applyMetaData(), and also added this method to page, section and formitem objects returned from Lianja.getElementByID(). e.g.

Lianja.getElementByID("page1.section1").applyMetaData()

THis is useful when called in a changed delegate to adjust the appearance of a section after editing a UI control.

Let's now take a look at how this all works.

Lianja Databases contain tables and their associated indexes. You can associate metadata with a database.

alter database mydb metadata "metadata_string" open database mydb omd = databaseMetaData()

Each table in a Lianja database can have an associated active data dictionary which contains business rules and triggers for the table itself and its columns. You manipulate these using standard SQL DDL Column constraints:

alter table customers modify constraint name set default “Barry” alter table customers modify constraint name drop default alter table customers modify constraint name set check not empty(name) alter table customers modify constraint name drop check alter table customers modify constraint name set error “Name cannot be blank” alter table customers modify constraint name drop error alter table customers modify constraint name set picture “@!” alter table customers modify constraint name drop picture alter table customers modify constraint name set choices “Apples,Oranges,Bananas” alter table customers modify constraint name drop choices alter table customers modify constraint name set not null alter table customers modify constraint name set null alter table customers modify constraint pkid set autoinc alter table customers modify constraint pkid drop autoinc alter table customers modify constraint name set help “Customer name” alter table customers modify constraint name drop help

Triggers (written in Lianja/VFP):

alter table customers modify onopen “customers_onopen” alter table customers modify onclose “customers_onclose” alter table customers modify oninsert “customers_oninsert” alter table customers modify onbeforeinsert “customers_onbeforeinsert” alter table

customers modify onafterinsert “customers_onafterinsert” alter table customers modify onupdate “customers_onupdate” alter table customers modify onbeforeupdate “customers_onbeforeupdate” alter table customers modify onafterupdate “customers_onafterupdate” alter table customers modify ondelete “customers_ondelete” alter table customers modify onbeforedelete “customers_onbeforedelete” alter table customers modify onafterdelete “customers_onafterdelete”

We call this an “Active Data Dictionary” as these business rules are associated with the table whenever it is used in a desktop App, Lianja SQL Server or Lianja Cloud Server. Altering any of the attributes has immediate effect on all applications that use the corresponding table without needing to make any code changes.

The Active Data Dictionary can also be associated and used with Virtual Tables (third party databases e.g MSSQL, MySQL, PostgreSQL etc) when the tables are opened. To provide a flexible solution so that any type of user defined metadata can be associated with a table you use the METADATA clause on the CREATE/ALTER table SQL DDL commands.

The metadata you specify should be a standard metadata encoded character string. After a table is opened you can obtain a metadata object using the new TABLEMETADATA() function. Example:

```
Use customers   omd = metadata_decode( tableMetaData() )
```

The TABLEMETADATA() function returns the METADATA string from the Active Data Dictionary associated with the table. You can change the elements of the metadata object then update the Active Data Dictionary with the new metadata. Example:

```
use customers  omd = metadata_decode( tableMetaData() )  omd.searchpanelvisible = .t.  use
alter table customers metadata metadata_encode(omd)
```

It is important to realize that the members of the metadata object are completely up to the data architect who creates the database schema. This provides them with the ability to create a complex relationship of database tables and their associated UI attributes in a database design tool such as XCase or other similar type of ER CASE tool.

Metadata can also be associated with columns of a table. You use the columnMetaData(cExp) function to obtain the column metadata. Example:

```
omd = object()  omd.caption = "Customer Name"  omd.searchfield = .t.  alter table customers
modify constraint name metadata metadata_encode(omd)  use customers  omd =
```

metadata_decode(columnMetaData("name")) omd.caption = "Customer" omd.picture = "@!" omd.searchfield = .t. alter table customers modify constraint name metadata metadata_encode(omd)

How does this all work with the Lianja App Builder?

When you create a new section by dragging a table onto a page (or opening an existing App) Lianja will layout the section, then if the script setupUI.prg exists in the App or setupUI_tablename exists in the database it will be called with the section id as the first parameter and the cursor for the table bound to that section will be current. Example:

```
  // File: database:setupUI_tablename.prg then app:setupUI.prg  parameter id  local osection = Lianja.getElementByID( id )    // check to see if UI section attributes are same version as last time applied  // and if so do nothing (improves performance)  if osection.metaDataVersion = tableMetaDataVersion()      return  endif  osection.metaDataVersion = tableMetaDataVersion()  local omd = metadata_decode( tableMetaData() )    // Call setupUI to override deferred page loading when applying the MetaData  osection.setupUI()    // Interrogate the metadata and apply the attributes to the section  // …  // foreach formitem in the form section (can also traverse grid columns)  local i  local oitem  local count = osection.count  for i=1 to count    oitem = osection.item( i )    omd = metadata_decode( columnMetaData( oitem.controlsource ) )    if not is_object(omd)      loop    endif    // Interrogate the column metadata and apply the attributes to the field or grid column    // ...  endfor
```

If your App is a Web or Mobile App you will now need to “Preview” it and then “Deploy” it (or rebuild a Mobile App) to reflect the changes made in the metadata.

To set the set the attributes for a page, section or form item use the **setAttr(name, value)** method which has been added to each of these objects.

For all attribute names that can be set on pages, sections and formitems, follow the links here .