Plot Layout Programming - "LAYOUT macros"

Preliminary documentation.

Revision date: May 16th 2006.
For comments and suggestions on this package please contact: peter-rene.steiner@bruker-biospin.de

Bruker software support is available via phone, fax, e-mail, Internet, or ISDN.
Please contact your local office, or directly:
Address: Bruker BioSpin GmbH Software Department Silberstreifen D-76287 Rheinstetten Germany
Phone: +49 (721) 5161 455
Fax: +49 (721) 5161 943
E-mail: nmr-software-support@bruker-biospin.de
WWW: www.bruker-biospin.de / www.bruker-biospin.com .

Why using LAYOUT macros?

A number of customers have shown their interest in programmatic access to the behavior of TOPSPIN layouts. There are working scenarios where the WYSIWYG approach of the Plot Editor application is not the best or easiest solution. Although such applications are typically part of "advanced" solutions for NMR automation, there has been a lack of support for those problems.

However, several approaches in the past to write TOPSPIN layout files (*.XWP files, former "XWIN-PLOT" layout files) on-the-fly, or to manipulate existing XWP files, have experienced severe difficulties. The syntax used is very intransparent and the formal structure had never been designed for external manipulation.

LAYOUT macros try to fill this gap. They provide a tool that allows customers to create plot layouts (XWP files) "on-the-fly" while providing the necessary information in a logical context and hiding the complex technical structure. Additionally, they provide an abstraction layer that enables Bruker to include further improvements and changes in Plot Editor without urging customers to rewrite existing AU programs. As long as users rely on the documented LAYOUT macros Bruker has the freedom to modify the implementation without breaking the compatiblity.

Installation

If you already had installed the preliminary version of August 15th, 2005, you need to undo the changes applied <ts_inst>/exp/stan/nmr/au/vorspann. Edit <ts_inst>/exp/stan/nmr/au/vorspann with any standard text editor and find and **remove** the line

```
#include "lcs_decl.h"
```

All AU programs using the LAYOUT macros need to include the line

```
#include <inc/layout_package>
```

as their first line. This conforms with similar AU extensions already delivered by Bruker.

Compatibility

This package can be used together with TOPSPIN 1.3 and above. It is included with TOSPIN 2.0. LAYOUT macros create layouts for the most current TOPSPIN version by default (i.e. TOPSPIN 2.0). The layout version created can be

controlled by using a #define statement as the first line in your AU code:

```
#define LAYOUT_VERSION_CREATE 43  // for TOPSPIN 1.3
```

If you do not want to modify all your AU programs, as an alternative, edit the file

```
<ts_inst>/prog/include/inc/layout_package_h>
```

and change the global definition accordingly. In that case you should keep in mind that new installations will overwrite that setting again. The macros include a test against the TOPSPIN version used, and "older" Plot Editors will deny opening "new" layouts automatically.

Note: The files included in "TOPSPIN 1.3 patchlevel 6" are already preconfigured to create layouts compatible with 1.3 .

How is it done?

The LAYOUT macros provide new "variable" types for typical objects in a plot layout, i.e. title, parameters, 1D spectrum, text, etc. Such an object has a number of values describing its design. E.g. a text object knows its position, its size and its textual content. For each object type there is corresponding macro that allows to initialise the variable with default settings. The user could then modify some properties of the created object as required. A third type of macro is used to "add" this object to a current layout.

Macros for beginning and ending a layout file complete this command set. Thus the workflow is 'to begin', then 'to add' several objects, finally to 'to end' the layout creation. Once a layout file has been created it is available on disk and there is technically no difference to other layouts delivered by Bruker or created with TopSpin Plot Editor.

All macros begin with "LAYOUT_". Technically, the "variables" are C style structs with a number of members. For each type of object xyz there are macros defined:

- **LAYOUT_OBJ_xyz** giving the variable type when declaring on the stack
- **LAYOUT_xyz_DEFAULTS** providing default values for all struct members, used to initialise the variable
- **LAYOUT_ADD_xyz** used to add the an xyz object to the currently created layout file with settings given by the current values of its struct members.

Examples

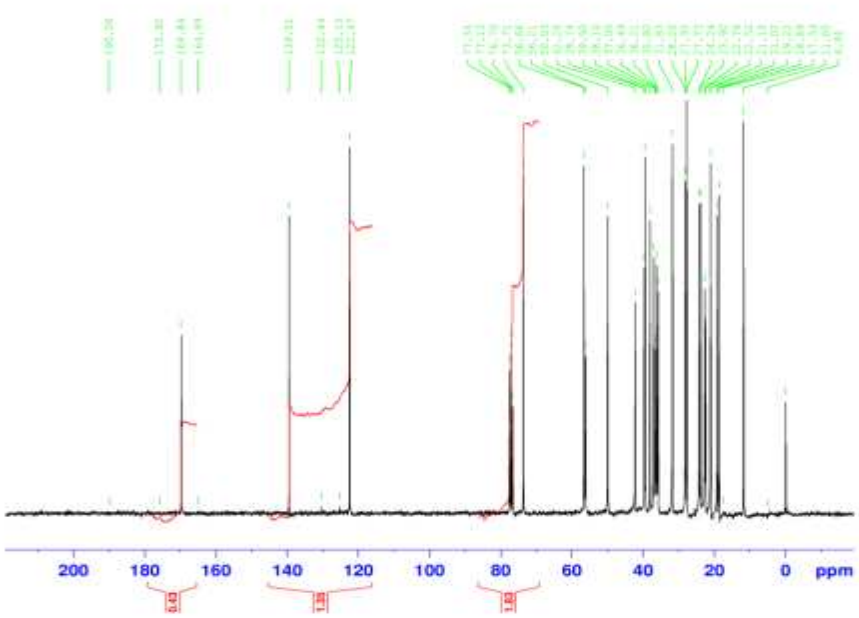
Example 1: Minimal 1D layout

The minimal number of instructions to construct a plot layout file could look like:

```
#include <inc/layout_package>

LAYOUT_FORMAT my_format = LAYOUT_FORMAT_DEFAULTS;
LAYOUT_OBJ_1D my_obj = LAYOUT_1D_DEFAULTS;
LAYOUT_BEGIN_FILE("c:\\temp\\plot.xwp", my_format);
LAYOUT_ADD_1D_OBJECT(my_obj);
LAYOUT_END_FILE;
QUIT
```

It only contains a 1D spectrum and nothing else. Due to the default settings contained in LAYOUT_FORMAT_DEFAULTS and LAYOUT_1D_DEFAULTS the spectrum will have a standard position on A4 paper and standard colors. Reset actions are preset accordingly so when using this layout on an arbitrary dataset (change the parameter LAYOUT to "c:\temp\plot.xwp", then enter 'plot -r') it should show the entire dataset with a reasonable scaling of intensities.

Note that backslashes in Windows file names must be doubled due to the C style compilation of AU programs. Using one forward slash instead is also accepted and may be easier to use ("c:/temp/plot.xwp").

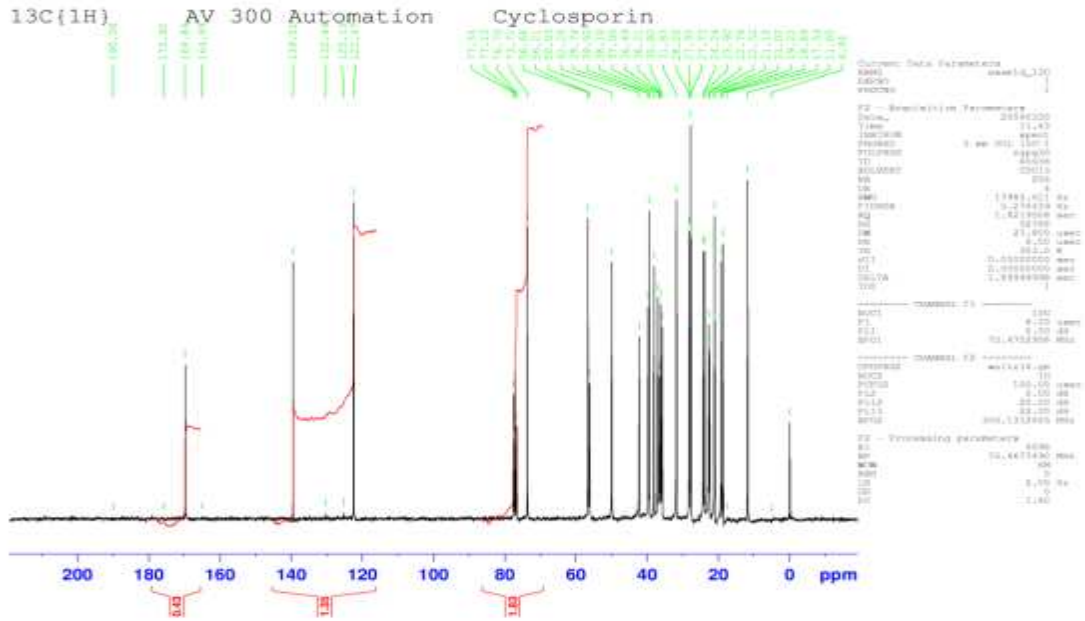


Source code provided as AU program "layout_ex_simple_1d".

Example 2: AU program creating a standard 1D layout

To provide something more typical for NMR users the following example creates a layout similar to the "1D_H.xwp" standard layout delivered with TOPSPIN.

```
#include <inc/layout_package>

LAYOUT_FORMAT my_format = LAYOUT_FORMAT_DEFAULTS;
LAYOUT_OBJ_1D my_obj = LAYOUT_1D_DEFAULTS;
LAYOUT_OBJ_TITLE my_title = LAYOUT_TITLE_DEFAULTS;
LAYOUT_OBJ_PARAMETERS my_param = LAYOUT_PARAMETERS_DEFAULTS;
char plotname[256];
sprintf(plotname, "%s/plot-output.xwp", PathSystemHome());
STOREPAR("LAYOUT", plotname);
LAYOUT_BEGIN_FILE(plotname, my_format);
LAYOUT_ADD(my_title);
LAYOUT_ADD_1D_OBJECT(my_obj)
LAYOUT_ADD_PARAMETERS(my_param);
LAYOUT_END_FILE;
CPR_exec("plot -r", WAIT_TERM);
QUIT
```



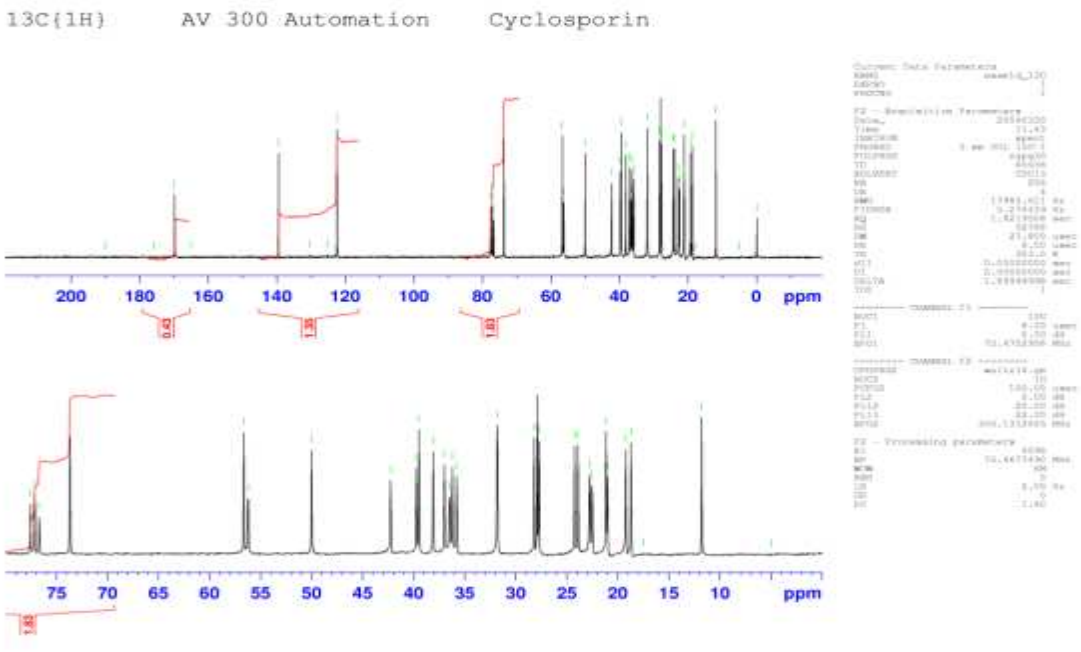
Source code provided as AU program "layout_ex_featured_1d".

Example 3:

Once "added" to the layout an existing variable can be modified and reused to add a second object of this type with sligthly different settings. This is an economic way to create sophisticated layouts. Only elements in bold letters are new compared to example 2:

```
#include <inc/layout_package>

LAYOUT_FORMAT my_format = LAYOUT_FORMAT_DEFAULTS;
LAYOUT_OBJ_1D my_obj = LAYOUT_1D_DEFAULTS;
LAYOUT_OBJ_TITLE my_title = LAYOUT_TITLE_DEFAULTS;
LAYOUT_OBJ_PARAMETERS my_param = LAYOUT_PARAMETERS_DEFAULTS;
char plotname[256];
sprintf(plotname, "%s/plot-output.xwp", PathSystemHome());
STOREPAR("LAYOUT", plotname);
LAYOUT_BEGIN_FILE(plotname, my_format);
LAYOUT_ADD(my_title);

my_obj.xMin = 80.0;                                    /* first 1D, fixed plot li
my_obj.xMax = 0.0;
my_obj.resetActionX = LayoutResetActionAxisDontChange;
my_obj.yPos = 2.5;
my_obj.xDim = 19.0;
my_obj.yDim = 4.8;
my_obj.showPeaks = 0;
LAYOUT_ADD_1D_OBJECT(my_obj)

my_obj.yPos = 10.5;                                  /* second 1D, limits from F1
my_obj.resetActionX = LayoutResetActionAxisUseF1PF2P;
LAYOUT_ADD_1D_OBJECT(my_obj)

LAYOUT_ADD_PARAMETERS(my_param);
LAYOUT_END_FILE;
```

```
CPR_exec("plot -r", WAIT_TERM);
QUIT
```



Source code provided as AU program "layout_ex_1d_with_zoom".

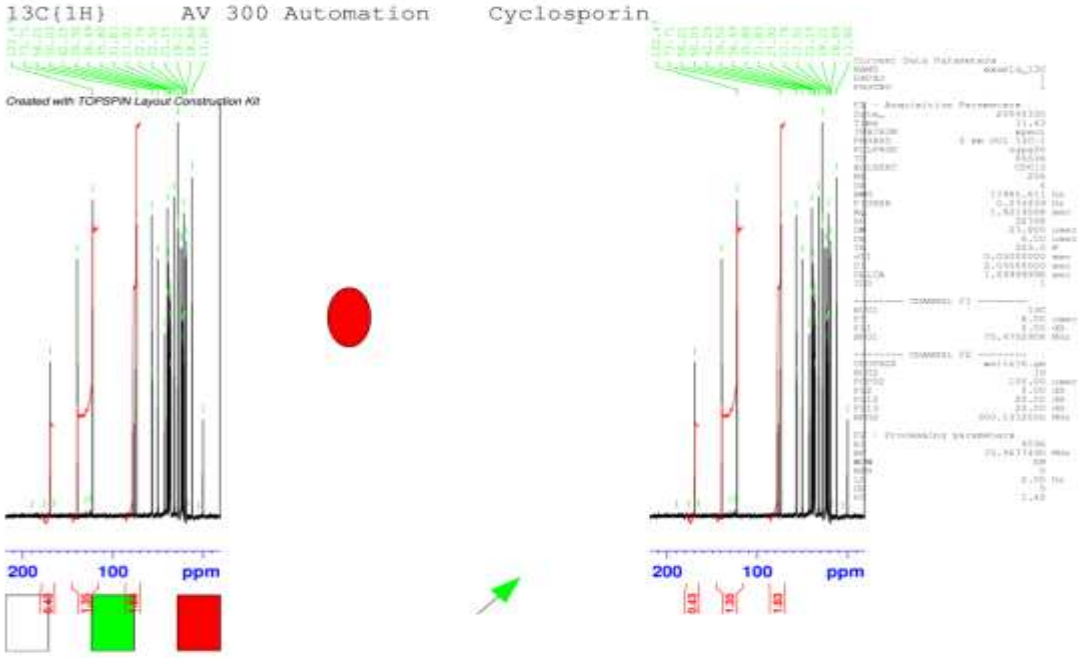
Example 4:

The existing "plotx" AU program has been rewritten to use the LAYOUT macro set. "plotx" reads in the "intrng" file and peak picking results and creates single plots of all integral regions while economically saving paper (several integrals on one paper as possible). Using the new macro set dramatically improves the readability of the "plotx" code (compared to the previous version) and makes it much easier to apply personal modifications.

Source code provided as AU program "plotx".

Example 5:

Some more gaming around with different objects including changed colors.

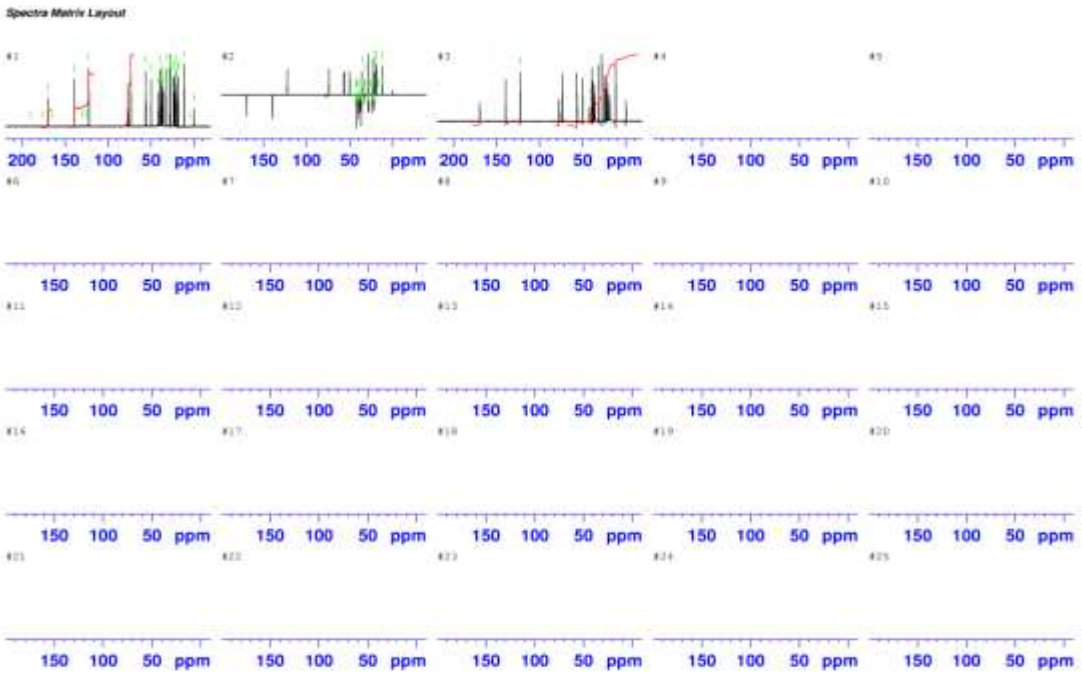


Source code provided as AU program "layout_ex_colors".

Example 6:

This example shows the construction of a 5x5 "matrix" of 1D spectra. Once the data set portfolio is preset with 25 datasets the constructed layout will show all 25 spectra on one paper. Due to the structure of this AU program it would be very easy to decide at run-time whether the layout should contain e.g., a 3x3, 4x4, 5x5 or else matrix of data sets.

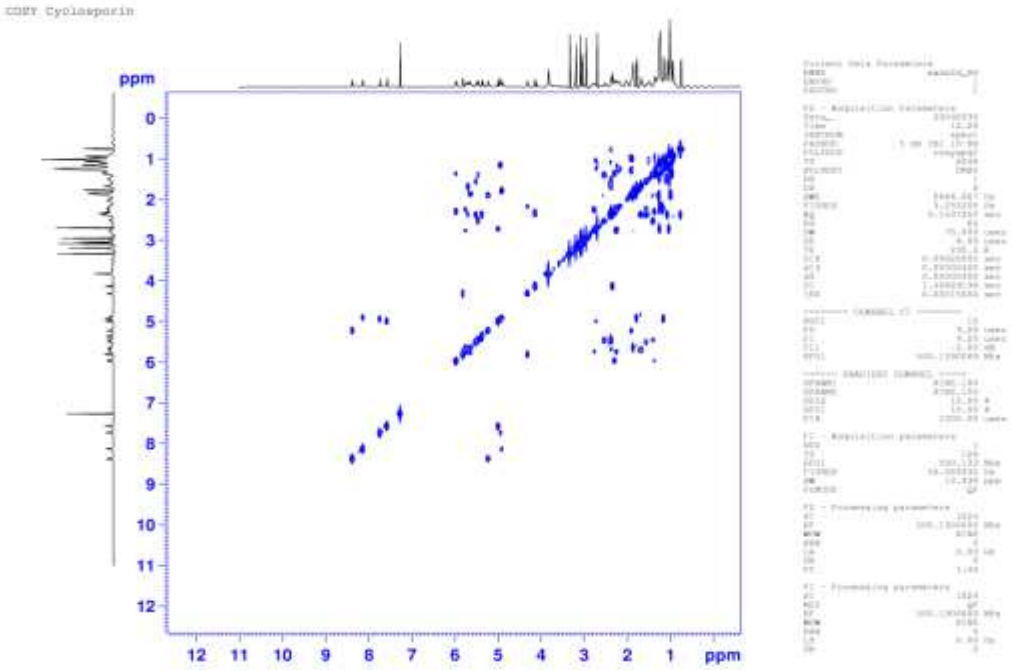
The source code for this "advanced" layout is rather short.



Source code provided as AU program "layout_ex_matrix_1d".

Example 7

A minimal 2D layout.

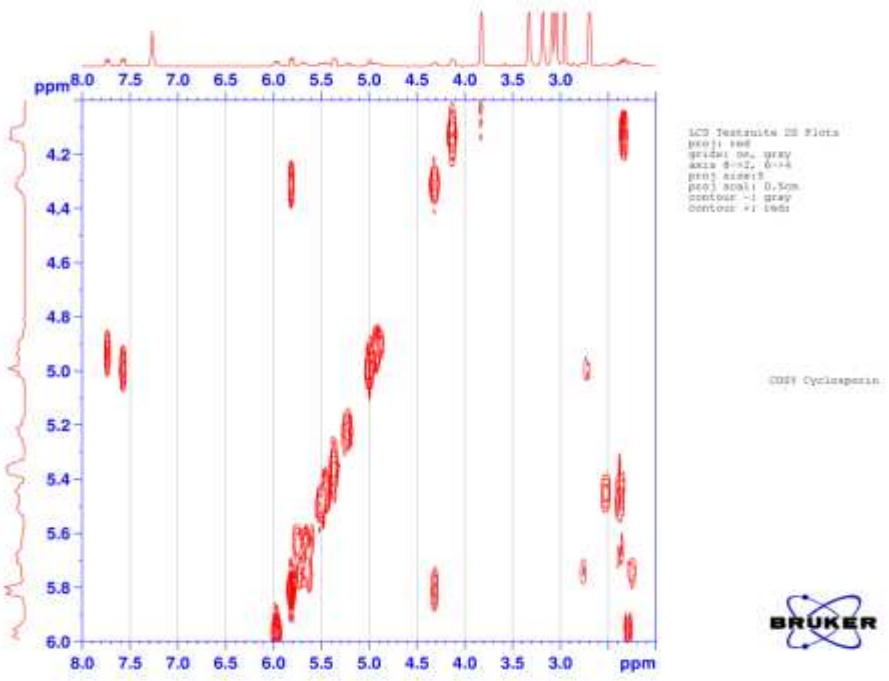


Source code provided as AU program "layout_ex_simple_2d"

Example 8

A more complex code illustrating how to modify settings for projections and contour levels. Note that Plot Editor normally gets the contour levels from data set so there is no need to predefine contour levels in the AU program. This applies to scenarios "plot" (*using TOPSPIN plot limis*) and "plot -r" (applying *Reset Actions*). Only in the case of "plot -n" (*doing nothing after loading data set and layout*) the contour levels are set as specified by the layout / AU program.

This example also shows the usage of Metafile and NMRText objects.



Source code provided as AU program "layout_ex_featured_2d".

Reference information about LAYOUT Macros

The LAYOUT macro package is work in progress and a full reference manual is still missing. However, documentation on available constants, color values etc. can be found by studying the header file

`<ts_inst>/prog/include/inc/layout_package_h`

It contains sections for "FORMAT", "TITLE", "PARAMETERS", "1D", "RECTANGLE" etc.

Each sections consists of the corresponding struct declaration followed by the macro providing the default settings. By inspecting the names of the struct members and their default values it should be possible to guess its meaning. For example the section for TITLE looks like:

```
/* ------------------------------------------------------ */
/* TITLE */
/* ------------------------------------------------------ */
typedef struct lpg_layout_obj_title {
int dataSetNumber;
double xPos;
double yPos;
...
```

```
} LAYOUT_OBJ_TITLE;

#define lpg_LAYOUT_TITLE_DEFAULTS { \
0, \
0.0, \
15.5, \
...
}
```

One can see that the title object contains values "xPos" and "yPos" which in fact describe the position of the title text box. The according default values from the DEFAULTS definition are 0.0 and 15.5, so the default position of a title object is 0.0 cm from left and 15.5 cm from bottom.

Positions and dimensions generally are given in centimeters from the bottom left edge of the printable area. The printable area is the defined by the paper size and the paper margins as seen in the the Plot Editor display.

Now look at the LINE section: For the "arrowType" there is default value defined as "LayoutArrowTypeNoTip". This indicates that possible other values are defined in that style, too. Near the beginning of the file you will find the corresponding enumeration section. We see that possible values are:

- LayoutArrowTypeNoTip
- LayoutArrowTypeBeginTip
- LayoutArrowTypeEndTip
- LayoutArrowTypeBothTip

The same principle applies to other values like e.g. LayoutAxisUnitPPM in a 1D object.

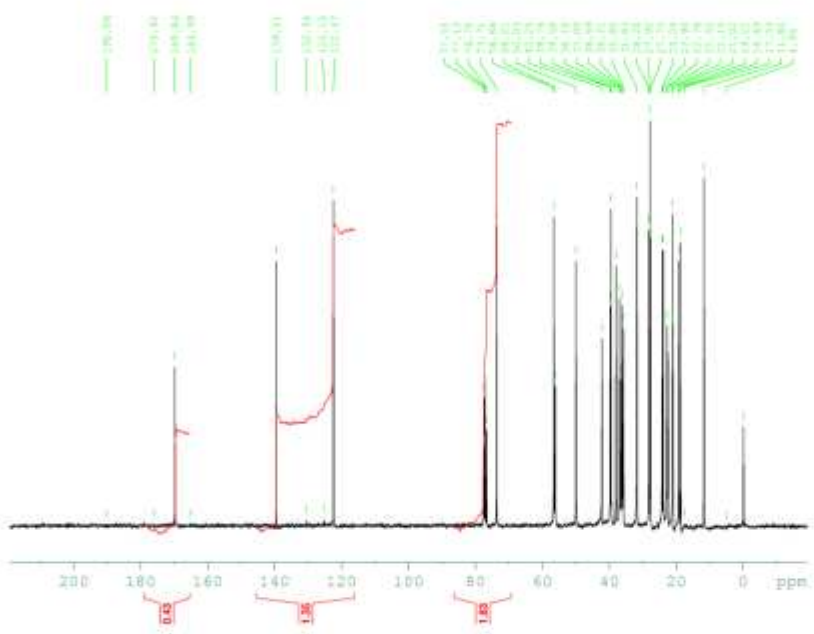
Working with attributes

Object properties like e.g. color of integral lines, font of axis labels, or line widths are not stored directly to an object. Instead, a complete collection of settings is stored as an attribute set (LAYOUT_ATTRIBUTES). Each attribute set has a unique ID. The possible number of attribute sets is unlimited. To specify how e.g. the integral lines of an 1D object should look like just the desired **`attributeId`** of the right attribute set is assigned to the 1D object's **`integralAttributeId`** value then

For the default settings the LAYOUT macro package uses preconfigured variables that should not be changed. Therefore one should create a new attribute set, apply the desired changes, add it to the layout by LAYOUT_ADD_ATTRIBUTES(), and then use its **`attributeId`** with another graphical object.

The significant elements to get the curve of an 1D graph in dark green color are *(complete code provided as AU program "layout_ex_attributes")*:

```
...
LAYOUT_OBJ_1D     my_1d  = LAYOUT_1D_DEFAULTS;
LAYOUT_ATTRIBUTES my_att = LAYOUT_ATTRIBUTES_DEFAULTS;
...
my_att.lineColor = LayoutColorSeaGreen;
LAYOUT_ADD_ATTRIBUTES(my_att);
...
my_1d.axisAttributeId = my_att.attributeId;
LAYOUT_ADD_1D_OBJECT(my_1d)
...
```



With that example now the axis has green color but, unfortunately, also shows the labels with Courier font instead of Helvetica. This is because the general default for attributes is set to using Courier while the 1D object already had been preconfigured for Helvetica. As a help the internal defaults of the LAYOUT macro package are accessible by variables. That allows to make "copies" as a starting point thus saving additional code:

```
...
LAYOUT_OBJ_1D     my_1d  = LAYOUT_1D_DEFAULTS;
LAYOUT_ATTRIBUTES my_att = LAYOUT_ATTRIBUTES_DEFAULTS;
...
my_att = lpgLayoutAttributesStandardAxis;   // copy settings
my_att.lineColor = LayoutColorYellow;
LAYOUT_ADD_ATTRIBUTES(my_att);
...
my_1d.curveAttributeId = my_att.attributeId;
LAYOUT_ADD_1D_OBJECT(my_1d)
...
```

Now the axis uses green color but still uses Helvetica for the axis labels and also the typical font size.

The names of the default attribute sets are:

```
lpgLayoutAttributesStandardAxis;               // axis of 1D and 2D objects
lpgLayoutAttributesStandardGrid;               // grid of 1D and 2D objects
lpgLayoutAttributesStandardTitle;              // Title object
lpgLayoutAttributesStandardText;               // Text and NMRText object
lpgLayoutAttributesStandardData;               // curve of 1D object
lpgLayoutAttributesStandardParameters;         // Parameters object
lpgLayoutAttributesStandardPeaks;              // peaks of 1D object
lpgLayoutAttributesStandardIntegrals;          // integrals of 1D object
lpgLayoutAttributesStandardContourLevel;       // contour levels of 2D object
```

RELEASE INFORMATION

TODOs

- Implement structures and macros for FID, Stacked, EPSI objects
- Include programming reference in documentation
- Implement structures and macros for easy handling of integral list and peak list
- Provide tables with size information for standard page sizes (A3, A4, Letter, Legal)
- Extend test suite

NEW & CHANGED

Version March 27th/May 16th, 2006

- Support for features of TOPSPIN 1.3 and 2.0. Macros create layouts for TOPSPIN 2.0 by default (support for manual settings for axis ticks overriding the automatic axis tick calculation, digital signature in parameter block).
- Control over the layout version. See section "Compatibility" about how to create layouts for TOPSPIN 1.3.(support for "Show scaling information for X axis" and "Integrals above X axis" of 1D objects).
- The use of the "lcs" acronym has been removed completely to avoid a mix-up with LC spectroscopy code.
- This documentation is now located at <ts_inst>/prog/docu/english/xwinproc/plotprogramming.pdf . It will show up in TOPSPIN 2.0's doc overview as "Plot Layout Programming"
- Bruker AU program 'plotx' is now implemented using LAYOUT macros (there is no 'plotx2' anymore).

Version February 25th, 2006

- Package introduced to TOPSPIN installation, now under the name "Plot Layout Programming". However, the acronym 'LCS' used somewhere in the package for variable names and constants has not been changed for reasons of compatibility with existing code.

Version January 11th, 2006

- Reworked 'plotx' code (available as 'plotx2' in the release so customer changes to original 'plotx' are not overwritten unintentionally).

Version October 19th, 2005

- changed 'axesAttributeId' to 'axisAttributeId' in 1D object
- LayoutResetActionX renamed to LayoutResetActionAxis
- LayoutResetActionY renamed to LayoutResetActionIntensity
- support for LAYOUT_OBJECT_2D, LAYOUT_OBJECT_METAFILE, LAYOUT_OBJECT_NMRTEXT
- improved implementation of LAYOUT_xyz_DEFAULTS to allow assignments at any place within the code. The following construct is now possible:

  ```
  LAYOUT_OBJECT_2D my_obj = LAYOUT_2D_DEFAULTS;
  my_obj.xPos = 25.0;
  my_obj = LAYOUT_2D_DEFAULTS;
  // now e.g. the x position of my_obj has the the default again
  ```
- basic documentation about handling attribute sets