Introduction

The Image Sequence Analysis Tool (ISA) is a subpackage of the ParaVision Image Display & Processing Tool. Its concept is to provide a very general and flexible framework for the visualization and statistical analysis of sequences of images. It is a package intended for any kind of analysis of the type where some values, identically evaluated for each image of a series of images and depending on some parameter as a function, are compared with each other in some predefined manner. For other details about the ISA Tool see chapter O-10.

The central functionality of ISA allows the user to define an arbitrary Region of Interest (ROI) within one representative frame of the image sequence displayed in a viewport of the Image Display & Processing Tool and then to get a graphical presentation of the mean values of the image intensities for the currently defined ROI as a function of the image number within the sequence or the corresponding variable as well as a table of the mean values and their statistical errors. While modifying the ROI interactively the data point display and the statistics table are updated in real-time. Defining a second, third,... ROI allows the user to freeze the curve and statistics display status of the already defined ROIs and to add further sets of data points to the plot and further columns to the statistics table.

The resulting plot of data points (and fit / data analysis curves) can be printed as a postscript file, the statistics table can be exported in ASCII format to other programs.

ISA fit functions

After inspecting the pure data points typically the user will apply some data analysis algorithm(s) to the image sequence. In many cases one of the predefined sequence analysis functions will fulfil his/her requirements. If not, the ISA framework offers the user the possibility to provide an own analysis function (in "C"), link it to the running program and integrate it into the standard GUI.

Writing ISA functions is perhaps the most difficult topic in ISA. While it is quite
easy to create something reasonable and even fittable from some analytically

described function (provided that it is possible to determine good starting val-

ues for the parameters), it can be far more complex to develop a function com-

patible with virtually any data, or some special function which is not

analytically presentable. Nonlinear least squares fitting can be rather tricky.

Moreover, if somebody wishes to adapt the ISA mode selector he has to be

familiar with the parameter overlay to be able to implement such new ISA

methods without influencing other ISA and PVM methods.

Abbreviations

Abbreviations 9.2

ROI- Region of Interest
ISA Tool - Image Sequence Analysis Tool
XTIP- Image Display & Processing Tool
GUI - Graphical User Interface
ISA sequence- Selected frames defined by ISA_first_image, ISA_num_images,
ISA_image_incr

The Function Menu 9.3

The Function menu allows the advanced user to work with non-standard
ISA functions conveniently.

To be able to understand the following rather complex sections the user must
be familiar with the UNIX operating system and with the C programming lan-
guage, know how to compile user automation (AU) programs and have some
experience in writing some AU programs of at least medium complexity. The
Function drop-down menu allows the user to perform the following operations:

Select ISA function - Shows a list of available ISA functions and allows the user
to select one to be used in subsequent ISA operations. One may enter a name
which does not exist in the list to begin to edit a new function later.
The name selected is automatically assigned to the ISA_func_name parameter.

Edit ISA function - Calls the external editor to edit the source text of the current
user ISA function. The editor command is taken from the $UXNMR_EDITOR
environment variable (kwrite by default if $UXNMR_EDITOR is empty).

Copy ISA function - Copy one ISA function into another one.

Rename ISA function - Rename an ISA function.
Delete ISA function - Delete the selected user ISA function.

Detach ISA function - Detach the current ISA function shared object from the Image Display & Processing executable. The function will be attached automatically if when performing the next ISA operation and the initialization phase of the ISA function is performed.

Initialize ISA - Reinitialize the ISA framework. If the ISA function was detached it is now explicitly attached and the initialization phase for the ISA function is executed.

Edit mode selector - Call the external editor ($UXNMR_EDITOR or kwrite) to edit the ISA mode selector source.

Delete mode selector - Delete the user ISA mode selector.

Detach mode selector - Detach the ISA mode selector, similarly to Detach ISA function.

The ISA functions reside in two locations. Initially they reside in `<PvInstDir>/exp/stand/nmr/isa/src` (the sources, names without any suffix) and in `<PvInstDir>/exp/stand/nmr/isa/obj` (the compiled shared objects having the suffix .so). These are the standard system-wide ISA functions. User defined ISA functions may also exist and each ParaVision user may have different ones or nothing at all. The user’s ISA functions reside in `<PvInstDir>/prog/curdir/<User>/ParaVision/isa/src` (the sources) and in `<PvInstDir>/prog/curdir/<User>/ParaVision/isa/obj` (the shared objects). For each user only his/her own and system-wide ISA functions are visible; he/she cannot see the ISA functions belonging to other users. The ISA mode selector is a very special kind of ISA function: it is responsible for automatic recognition of the appropriate ISA mode and the associated ISA function and other ISA parameters from the acquisition parameters of the dataset. The name of the mode selector is setmode; its shared object resides also in the obj subdirectory, but its source is not found in the subdirectory src as ordinary ISA functions rather than in the upper directory isa. If the same ISA function name exists in both places the user’s ISA function has higher priority over a system-wide one. Before executing any part of a function the ISA package checks the ages of the ISA function source and shared object and, if the source is found to be newer, recompiles and reattaches it automatically.

The user can edit only his/her own ISA functions. If the Edit ISA function operation is started while some system-wide function is selected, this function is implicitly copied into the user’s directory, and this local copy will be edited. Correspondingly, nothing can be copied, renamed or deleted in the system-wide
directory; all these operations work with user’s functions only. The only exception is the ability to copy some system ISA function into the user’s ones under a different name. The practical consequence is that the simplest way to recover from the trouble after incorrect editing of a local copy of some system ISA function is to delete and detach it - the system function will be used next time. This applies also to the ISA mode selector. Editing ISA functions and/or the mode selector will block the Image Display and processing Tool until the editor is exited.

Prototype for user-defined fit function

The fit function developer’s to do list

1. Set number of parameters
2. Define function description text for user interface
3. Define x-axis description text for user interface
4. Define parameter description texts for user interface
5. Define initial value and limits for each parameter
6. Define default values for parameter images to be used if no fit result available
7. Define x-axis points for given function values
8. Improve parameter start values on the basis of the data points (x,y)
9. Specify fit function
10. Specify first partial derivative of fit function for all fit parameters

The prototype fit function

/*****************************/
/** Standard T2 Relaxation Decay: y = A + C * exp (-t/T2) **/
/*****************************/
/** Before PARAMETERS () static and global variables may be declared **/
/*****************************/
/******************** This line must be the first ! ********************/
PARAMETERS ( 3 ) /** Total number of parameters: 3 **/
/*****************************/
/** After PARAMETERS () local variables may be declared **/
int i;
double xmin, ymax, ymin;
/*****************************************************************************/
/*** The following options ***/
/*** override the function defaults. ***/
/*** They may appear in any order; if ***/
/*** they are absent defaults are used ***/
/*****************************************************************************/
/*** Short summaries about the function and each parameter ***/
FUNCTION ( "T2 relaxation: y=A+C*exp(-t/T2)" )
XAXIS ( "echo time [msec]" )
/*** Param# description ***/
PARDESC ( 0, "absolute bias" )
PARDESC ( 1, "signal intensity" )
PARDESC ( 2, "T2 relaxation time [msec]" )
/*** Initials: # low init. high (default: -MAXFLOAT..0..MAXFLOAT) ***/
PARINITLIMITS ( 0, 0, 0, MAXINT )
PARINITLIMITS ( 1, X_EPS, 1, MAXINT )
PARINITLIMITS ( 2, X_EPS, 1, MAXINT )
/*** Default values for parameter images ***/
PARDFLT ( 0, 0 )
PARDFLT ( 1, 0 )
PARDFLT ( 2, X_EPS )
/*** Initialise x-values reading the value from the ***/
/*** the array parameter named ACQ_echo_time ***/
INITPARXREADX ( ACQ_echo_time" )
/*** Try to predict better initial values ***/
ymax = X_EPS;
ymin = MAXINT;
for (i=0; i<NPOINTS; i++)
if (YPoint(i) > ymax && XPoint(i) >= 0)
    ymax = YPoint(i);
if (ymax > X_EPS) PARINIT ( 1, ymax )
ymax /= 2;
for (i=0; i<NPOINTS; i++)
if (YPoint(i) > 0 && XPoint(i) >= 0 && fabs(YPoint(i)-ymax) < ymin)
{
    ymin = YPoint(i);
xmin = XPoint(i);
}
ISA Function Programming

```c
if (ymin < MAXINT) PARINIT ( 2, xmin/M_LN2 )
/*** Fixing parameter 0 gives better convergence ***/
FIXED ( 0 )
******************************************************************************
/*** Body of the function (each line must be present) ***/
******************************************************************************
IMPLEMENTATION /*** Must come before the value ***/
/*** After IMPLEMENTATION local variables may be declared ***/
double expval;
expval = exp (-ARG/PAR(2));
/*** Function value (must be located before any derivative) ***/
VALUE ( PAR(0)+expval*PAR(1) )
/*** Derivatives on each parameter (the order is not important) ***/
DERIVATIVE ( 0, 1 )
DERIVATIVE ( 1, expval )
DERIVATIVE ( 2, expval*ARG*PAR(1)/PAR(2)/PAR(2) )
END
/*** After END static and global functions may be declared ***/
******************************************************************************
/*** End of file ***/
******************************************************************************
```

Compiling ISA functions

The process of compiling ISA functions or the mode selector is in many respects similar to the process of compiling AU programs. There exists a special header which is automatically placed at the beginning of each ISA function to produce valid C source code. This header is `<PvInstDir>/exp/stan/nmr/isa/start`. In this header a lot of useful macros are defined including the proper templates for initialization and implementation parts of ISA functions, and for the mode selector. If the user knows the C programming language they can easily understand the ISA program structure by inspecting the contents of this start file.

**ISA environment variables**

The user has some control over the compiling process via special environment variables. These environment variables are:

- **CC** - The name of the ParaVision C compiler executable. The default is `gcc`, the GNU C compiler.
ISA Function Programming

CFLAGS - The extra flags passed to the C compiler. Among them there may be, for example, “-O3” to produce highly optimized code. Which flags may be used depends on the C compiler.

LDFLAGS - The extra flags passed to the linker. The default is nothing as up to now there exists no ISA function which could need them. One flag which is absolutely necessary, -shared, which means to produce a shared object, is always implicitly added by ISA.

LDLIBS - The extra libraries to link the ISA functions with. The default is none. As any dynamically linked ISA function has access to virtually any system function. Such extra libraries will hardly be needed. However, if the user has some specific mathematical library they wish to use they should set its name here.

If there was some syntax error while compiling an ISA function it can be shown to the user whether the commands of supervised step-by-step analysis were used. It cannot be done during interactive manipulations because aggressive mousing grabs the pointer which would conflict with the modal error viewer.

There are two possible methods to debug an ISA function. The first one is to switch ISA_debug to Yes and to set ISA_debug_file to something meaningful in the ISA function. Then a text file with that name will be created in the user’s home directory during function fitting and all computed function values and derivatives for each variable parameter dumped to it. Such a file can then be inspected in order to check the numerical values for accuracy. This method is not applicable to the ISA mode selector as it cannot be fitted, as well as for other non-fittable functions which set ISA_bypass_fit to Yes. Another method is to debug using printf() calls whose output will appear in the window from which ParaVision had been started. The printf() calls can be used anywhere. After the function has been debugged they can either be removed completely or included into #if DEBUG...#endif blocks which will be either compiled or not depending on whether -DDEBUG=1 was used in CFLAGS or not (another example of the CFLAGS usage). Experienced users can use source level debuggers that are able to load symbols dynamically (e.g. gdb with share command).

ISA functions

Each ISA function has the same basic structure needed for correct referencing it from the main Image Display & Processing program:

<global user declarations> - Optional definitions of user’s global variables and functions.
PARAMETERS(<number of parameters>) - Required, defines the total number of
parameters, both variable and fixed, marks the beginning of the initialization phase.

<local user declarations for initialization> - Optional definitions of local variables visible inside initialization phase

<initialization code> - Optional but highly recommended code to preset parameter limits, starting values, setting various descriptive things, specifying function specialities, etc.

IMPLEMENTATION - Required, marks the end of the initialization phase and beginning of the implementation phase.

<local user declarations for implementation> - Optional definitions of local variables visible inside the implementation phase.

<implementation code> - Often required to compute the function value.

VALUE(<value>) - Required to set the function value for given argument.

<more implementation code> - Often required to compute the derivatives.

DERIVATIVE(<parameter>,<derivative>) - Used to set the function derivative for the given parameter at a given argument; the DERIVATIVE() macros must be specified at least once for each parameter that may be varied.

END - Required, marks the end of the implementation phase.

<more global user declarations> - Optional definitions of the user’s global functions.

**ISA mode selector**

The ISA mode selector automatically executed while entering and reentering ISA has a different structure:

<global user declarations> - Optional definitions of the user’s global variables and functions.

SETMODE - Required, marks the beginning of the setmode code.

<local user declarations> - Optional definitions of local variables visible inside setmode.

<setmode code> - The setmode code itself; strictly speaking optional but with an empty setmode there is no possibility to automatically select an appropriate ISA function depending on the acquisition method; usually presets at least the ISA function name and the X coordinates of the experimental data points, often some descriptive features such as the X axis labeling etc.

END - Required, marks the end of the setmode code.

<more global user declarations> - Optional definitions of user’s global functions.
ISA Function Programming

ISA macros

A number of macros are defined for more convenient access to ISA internals (all in upper case to avoid ambiguity). Each macro has its defined scope. Wrong macro use can lead either to syntax errors while compiling an ISA function or to incorrect function behavior. These macros are:

ERROR("<text>") - Abort ISA function or setmode with error message.

INITFIRST - For initialization phase; synonym INIT_FIRST_PASS; a boolean variable which can be tested to determine whether the function initialization phase has been entered for the first time after attaching it.

MODENAME("<name>") - For the mode selector only; sets ISA mode name, which must be unique and is freely defined by the user. Initially it is preset to an empty string; if the user changes it Reenter ISA will implicitly be executed with the changed mode name. Reenter ISA will be repeated until the mode name between the last two setmode calls is not changed. Using MODENAME() could be a bit tricky; it is intended for those datasets to which more than one ISA function can be applied.

FUNCNAME("<name>") - For the mode selector only; sets the name of the ISA function to be used for analysis. The corresponding ISA function must exist. The name will be assigned to the parameter ISA_func_name.

FUNCTION("<text>") - For initialization phase or mode selector; sets a short textual ISA function description to be displayed on the plot. The text will be assigned to the parameter ISA_func_descr.

XAXIS("<text>") - For initialization phase or mode selector; sets a short textual description of the X axis to be output on the plot. The text will be assigned to the parameter ISA_x_axis.

PARDESC(n,"<text>") - For initialization phase only; sets a short textual description of the function parameter number n (n is an integer). Recommended to describe the physical meaning of a parameter. The parameter numbers start from zero.

PARINIT(n,c) - For initialization phase only; sets the initial value c to the parameter number n (n is an integer, c is a double precision number). Quite often has a tremendous influence on the speed and quality of the fit convergence. Must be set with due care.

PARDFLT(n,d) - For initialization phase only; sets the default value d to the parameter number n (n is an integer, d is a double precision number). The default value is used while computing parameter images to assign it to the
poorly converged pixels or to those pixels which were not included in the ROI mask. Default is zero.

PARLOW\(n,l\) - For initialization phase only; sets the lower limit \(l\) to the parameter number \(n\) (\(n\) is an integer, \(l\) is a double precision number).

PARHIGH\(n,h\) - For initialization phase only; sets the upper limit \(h\) to the parameter number \(n\) (\(n\) is an integer, \(h\) is a double precision number).

PARLIMITS\(n,l,h\) - For initialization phase only; equivalent to PARLOW\(n,l\); PARHIGH\(n,h\). The ISA function writers are encouraged to set reasonable limits to all function parameters to either produce correct results or to indicate unacceptable fit quality. The latter is found by ISA software if any variable parameter reaches its lower or upper limit and the default parameter value for parameter images is taken in this case.

PARINITLIMITS\(n,l,c,h\) - For initialization phase only; equivalent to PARLIMITS\(n,l,h\); PARINIT\(n,c\).

REQUEST\(n\) - For initialization phase only; requests the value of the parameter number \(n\) (\(n\) is an integer) via a special dialogue. As the parameter description is normally an output in the dialogue it is desirable to set PARDESC\(n\) for this parameter before issuing REQUEST\(n\). However, as this dialogue will be shown at each initialization including implicit initialization it can interfere with interactive mousing. Therefore it is highly recommended not to use REQUEST\(n\), as well as other macros for dialogue communication with the user in ISA functions intended for routine use. Instead the ISA function writers are encouraged to precompute the approximated parameter value and set it via PARINIT\(n,c\). However, in rare cases such macros could be convenient while developing new functions.

INITREQUEST\(n\) - Same as REQUEST\(n\) but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

REQUESTX - Similar to REQUEST\(n\); requests the abscissa values for all data points. Can be used not only in the initialization phase of ISA functions but also for the mode selector. Analogously to REQUEST\(n\), it is undesirable to use this macro in ISA functions. However, it may be freely used in mode selector because it is executed rather rarely. Nevertheless, computing the right X values from appropriate acquisition parameters (if they do exist) is a far more elegant solution. The data points indices start from zero.
INITREQUESTX - Same as REQUESTX but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXEDIT("<text>","<names>") - Calls the standard PARX editor in synchronous mode (i.e., the function waits until exit from the editor, and it is not possible to execute further commands before the modal editor window is dismissed) to edit any PARX parameters whose names are listed in the second macro parameter delimited by spaces, commas or slashes. Unqualified, partially or fully qualified parameter names as well as class names are allowed. The text used as the first macro parameter is displayed in the dialogue title bar. Technically it is possible to call this macro in any place. However, because of the synchronous behavior it may be acceptable in mode selector only as it is executed rarely.

INITPARXEDIT("<text>","<names>") - Same as PARXEDIT("<text>","<names>") but only to be performed while going through initialization phase for the first time after attaching the ISA function. Strictly speaking never to be performed at all (see preceding considerations). Implemented mainly for completeness.

PARXREADSTR(s,"<name>") - Read the value of the PARX parameter <name>, format it in a similar way as in the spectrometer control tool and assign it the string variable s. This macro offers a convenient way to read values of string and enumerated parameters (in the latter case the enumerated value is converted to the string presentation automatically). It has no modal behavior and can in principle be used not only in mode selectors but also in ISA functions. However, as accessing PARX parameter spaces costs a lot of time it is important to avoid using PARX related macros and functions in those parts of ISA function code which are executed often, i.e., they may be quite desirable in mode selector or inside blocks of if(INITFIRST){} and very annoying in IMPLEMENTATION.

INITPARXREADSTR(s,"<name>") -
Same as PARXREADSTR(s,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function. This macro is harmless as it is executed rarely.

PARXREADVAR(v,"<name>") - Assign the value of the PARX parameter <name> to the double precision variable v. The parameter must be reduced to a scalar of either integer or double precision type.
INITPARXREADVAR(v,"<name>") -
Same as PARXREADVAR(v,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXREAD(n,"<name>") - Assign the value of the PARX parameter <name> to the ISA function parameter number n. The parameter must be reduced to a scalar of either integer or double precision type.

INITPARXREAD(n,"<name>") - Same as PARXREAD(n,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXREADX("<name>") - Assign the values of the PARX parameter <name> to the data points abscissa. The parameter must be reduced to a one-dimensional array of either integers or double precision number.

INITPARXREADX("<name>") - Same as PARXREADX("<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXWRITESTR(s,"<name>") - Opposite of PARXREADSTR(s,"<name>"); converts the string s to the appropriate parameter value and assigns it to the parameter, analogous to setting PARX parameters in the spectrometer control tool. Especially convenient for assigning string and enumerated parameters.

INITPARXWRITESTR(s,"<name>") - Same as PARXWRITESTR(s,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXWRITEVAR(v,"<name>") - Assign the value of the double precision variable v to the PARX parameter <name>. The parameter must be reduced to a scalar of either integer or double precision type.

INITPARXWRITEVAR(v,"<name>") - Same as PARXWRITEVAR(v,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

PARXWRITE(n,"<name>") - Assign the value of the ISA function parameter number n to the PARX parameter <name>. The parameter must be reduced to a scalar of either integer or double precision type.

INITPARXWRITE(n,"<name>") - Same as PARXWRITE(n,"<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.
ISA Function Programming

PARXWRITEX("<name>") - Assign the values of the data points abscissa to the PARX parameter <name>. The parameter must be reduced to a one-dimensional array of either integers or double precision numbers.

INITPARXWRITEX("<name>") - Same as PARXWRITEX("<name>") but only to be performed while going through the initialization phase for the first time after attaching the ISA function.

VARIABLE(n) - For initialization phase only; make the ISA function parameter number n variable. By default all parameters are variable.

FIXED(n) - For initialization phase only; make the ISA function parameter number n fixed. Opposite of VARIABLE(n).

IFVARIABLE(n)...ELSE...FI - For any ISA function phase; marks two blocks of code to be executed depending on whether the ISA function parameter number n is variable or not. The keyword ELSE and the block of code following ELSE is optional. The keyword FI is required. Can be used to test if there is an attempt to vary some by definition fixed parameter illegally.

IFFIXED(n)...ELSE...FI - For any ISA function phase; opposite of IFVARIABLE(n)....; the order of code blocks is reversed.

IFPLOT...ELSE...FI - For implementation phase only; similarly to IFVARIABLE(n)... marks two blocks of code to be executed depending on whether the ISA function is called while plotting (i.e. the derivatives are not needed) or while fitting (the derivatives must also be computed). Can be used to test if some non-fittable function was illegally used in Execute ISA.

IFFIT...ELSE...FI - For implementation phase only; opposite of IFPLOT...; the order of code blocks is reversed.

TOLERANCE(v) - For initialization phase only; assign the value of the variable v to the fit tolerance which will be set later to the parameter ISA_tolerance.

MAXITER(n) - For initialization phase only; assign the value of the integer variable n to the maximum number of iterations, which will be set later to the parameter ISA_max_iter.

NPOINTS - Synonym NDATA; the total number of the experimental data points. Can be used anywhere. Can also be used as the argument of the PARAMETERS() macro.

XPOINT(n) - Synonym XDATA[n]; the X value of the data point number n. Can be used anywhere but not in the PARAMETERS() macro.

YPOINT(n) - For ISA functions only; synonym YDATA[n]; the Y value of the data point number n.
STD(n) - For initialization phase only; synonym SIG[n]; the standard deviation of the Y value of the data point number n.

USED(n) - For initialization phase or for the mode selector; take the data point number n to be used in fitting and shown on the plot. By default all data points are used.

UNUSED(n) - For the initialization phase or the mode selector; removes the data point number n from fitting and from the plot (the point is left in the table). Opposite of USED(n).

IFUSED(n)...ELSE...FI - Similar to IFVARIABLE(n)...; marks two blocks of code to be executed depending on whether the data point number n is used or not.

IFUNUSED(n)...ELSE...FI - Opposite of IFUSED(n)...; the order of code blocks is reversed.

FORCE_REINIT - For initialization phase only; specifies that ISA must be implicitly initialized at each Execute ISA or Plot ISA function. May be used to compute parameters of non-fittable ISA functions from the complete set of the data points which is not available in the implementation phase.

BYPASS_FIT - For initialization phase only; specifies that the ISA function must not be fitted. It is often used with FORCE_REINIT.

ARG - For implementation phase only; synonym X; the double precision number value of the argument with which the function must compute its value and derivatives.

PAR(n) - For implementation phase only; synonym A[n]; the value of the function parameter number n.

VALUE(v) - For implementation phase only; sets the value of the ISA function from the double precision variable v.

DERIVATIVE(n,v) - For implementation phase only; sets the derivative of the ISA function for the parameter number n from the double precision variable v.

Rules for own variables

All symbols used internally in ISA functions and in the mode selector and defined in the start header have their names completely in UPPER CASE. Therefore, to define own variables or functions the user may freely use lower or mixed case so as not to interfere with the predefined symbols.

The following predefined variables can be used by ISA function writers directly (i.e. without the previously described macros):

In the mode selector: char MODE[], FUNC_NAME[], FUNC_DESCR[],
ISA Function Programming

X_ANNOT[]. They should normally be set with the macros MODENAME(), FUNCNAME(), FUNCTION(), XAXIS() correspondingly but may also be used directly, for example for comparisons.

In the initialization phase: char FUNC_DESCR[], X_ANNOT[]; int MA, *MFIT, *MAXIT, *REINIT, *BYPASS; double *TOLER. FUNC_DESCR[] and X_ANNOT[] can be set by macros FUNCTION() and XAXIS() correspondingly, *MAXIT and *TOLER - by MAXITER() and TOLERANCE(). *REINIT and *BYPASS are set by the macros FORCE_REINIT and BYPASS_FIT and cleared by the ISA package (they must not be cleared by the ISA function). MA and *MFIT may be used for comparisons only; they must not be altered by the ISA function. MA contains the total number of function parameters, *MFIT - the number of variable parameters.

In the implementation phase: int MA, *MFIT. They must not be altered, just as in the initialization phase.

In the argument of the PARAMETERS() macro it is allowed to use arithmetic expressions based on NPOINTS or NDATA.

Globally accessible are int pvIsaPsid - the PARX parameter space ID for the ISA dataset - and char PvIsaParxErrstr[] - error text (if any) filled by PARX related ISA functions. Both variables must not be altered.

External functions for use in user’s ISA functions

The following external functions are intended for use in user’s ISA functions and mode selector:

PvIsaParxEditSync (title, pvIsaPsid, argc, argv);
PvIsaVaParxEditSync (title, pvIsaPsid, ...);
PvIsaStrParxEditSync (title, pvIsaPsid, “<names>”);
PvIsaAskForItem (prompt, title, must_match, buffer, buffer_len, argc, argv);
PvIsaVaAskForItem (prompt, title, must_match, buffer, buffer_len, ...);
PvIsaStrAskForItem (prompt, title, must_match, buffer, buffer_len, “<names>”);
PvIsaAskForItemNumber (prompt, title, number, argc, argv);
PvIsaVaAskForItemNumber (prompt, title, number, ...);
PvIsaStrAskForItemNumber (prompt, title, number, “<names>”);
PvIsaAskForString (prompt, buffer, buffer_len);
PvIsaAskForInteger (prompt, ivalue, imin, imax);
PvIsaAskForDouble (prompt, dvalue, dmin, dmax);
ISA Function Programming

The parameters are:

char title[], prompt[] - Dialogue title and prompt.
int argc, char *argv[] - List of names similar to the argument list of the C main() function.
... - Variable list of names (char[]), terminated by NULL.
"<names>" - Single string (char[]) of names delimited by spaces, commas or slashes.
int must_matchif must_match = 1 - The item entered by the user must exist in the list of items.
char buffer[] - On enter - the default item value, after exit - the item value returned.
int buffer_len - The size of buffer in characters.
int *number - On enter - the default item number, after exit - the item number returned.
int *ivalue - On enter - the default integer value, after exit - the integer value returned.
int imin, imax - Integer bounds for *ivalue.
double *dvalue - The same as *ivalue but double.
double dmin, dmax - Double bounds for *dvalue.

All these functions open an appropriate dialogue to let the user enter or select something. They all return 0 if OK was pressed and -1 on Cancel or in case of error. All functions are synchronous, i. e. the user can do nothing outside the dialogue window and the program waits until the dialogue is dismissed.
PvIsaParxEditSync, PvIsaVaParxEditSync, PvIsaStrParxEditSync open the PARX editor window to edit specified list of PARX parameters and/or classes.
PvIsaAskForItem, PvIsaVaAskForItem, PvIsaStrAskForItem open a list window to select or enter literally some element belonging to the list. They return the string value of the item.
PvIsaAskForItemNumber, PvIsaVaAskForItemNumber, PvIsaStrAskForItemNumber are similar to the preceding three but return the item number in the list instead of the item value itself.
PvIsaAskForString requests to enter an arbitrary string.
PvIsaAskForInteger and PvIsaAskForDouble request an integer or double value correspondingly which must lie inside specified bounds.
ISA Function Programming

ISA functions for reference

Any ISA function can technically reference virtually anything which is defined globally in the system C library (libc), mathematical library (libm), X11 and Motif libraries (libXm, libXt, libX11), PARX libraries (libParx, libParxEd), many other Bruker libraries, Image Display & Processing internal functions, other libraries if they are linked to ISA function explicitly via LDLIBS. It is technically possible, for example, to issue any command to CPR (with CPR_exec()) or to the ParaVision command dispatcher (with PvCmdExecute()), any Unix command (with system()), open graphics window, it is technically even possible to start acquisition, perform an experiment, process it and visualize the result in some way, everything from within ISA function. However, the users are encouraged not to do such actions. The purpose of the ISA functions is the analysis of image sequences, they do not have to anything extra than to achieve their main goal, image sequence analysis, in a way as efficient and robust as possible. Moreover, improper use of some system functions can lead to very surprising effects. For example calling exit(0) from an ISA function will cause the system to perform a normal exit from the main Image Display & Processing program which would be almost certainly not what the developer expected. Explicit returns are also forbidden because proper return value and consistent values of ISA function parameters are later checked in the package in many respects.

How to write user defined ISA function

To write user defined ISA function from scratch is not an easy task. It is usually more convenient to copy some already working ISA function similar to that which the user wants to implement and then edit it to adapt it to the actual experiment properties. This is also true for the mode selector. Moreover, the user should edit the system mode selector in order to add his new ISA methods so that the preceding methods continued to work.

However, if the user wishes to write an ISA function from scratch the following sequence is suggested:

1. Specify PARAMETERS() and IMPLEMENTATION. For each parameter assign PARDESC() and PARLIMITS(). If some parameters are to be fixed, specify them. Initialize X points. In the IMPLEMENTATION compute VALUE(). Define ROI, do Calc. Pnts for ROI. Try Initialize ISA, check if the syntax was correct. Try Plot ISA Curve, check if the function values are correctly computed.

2. In the IMPLEMENTATION compute DERIVATIVE() for each variable parameter. Do again Initialize ISA. Try Execute ISA, see how well the curve fits and
how quickly the fit converges. If there is any doubt switch ISA_debug to Yes or insert debugging printf() into the function code.

3. Try fitting with various starting parameter values and investigate the fitting behavior. Develop the initialization phase to approximate the starting parameter values realistically so that this computation is fast (it is technically possible to perform the complete fit inside initialization phase but it would waste CPU time) and in the same time could produce results which were reasonably near to the expected values. Try Initialize ISA followed by Plot ISA Curve (without fitting) to test the robustness of the initialization for various experimental data.

4. Introduce more descriptions such as FUNCTION() and XAXIS() and add the appropriate code to the mode selector so that the new ISA function is correctly recognized. Test it.

5. Final testing should include repetitive Reenter ISA, interactive mousing, Calculate parameter images and Generate artificial images when applicable. The program should not crash even with noise data.

For such datasets which can be analysed with more than one ISA method it may appear that the wrong ISA sequence is selected. If this is the case edit the ISA parameter class, where ISA_start_image, ISA_num_images and ISA_image_incr are set explicitly, then perform Reenter ISA. If the current active image belongs to the specified ISA sequence these parameters will not be reinitialized while reentering. The mode selector for such “multiISA” datasets works in a similar way where necessary reentering is induced by explicit MODENAME() switching.

For more information about writing ISA functions and mode selector the user should inspect Bruker supplied example sources found in the system ISA directory.

**Standard ISA fit function examples**

The following more or less standard functions are supplied by BRUKER:

setmode - ISA mode selector.
t2 - T2 fit for MSME_TOMO datasets (IMND).
t2log - Same as t2 but for linearized fit (ISA_scaling == Log_Scaling).
t2vtr - T2 fit for MSME, MGE datasets via ACQ_echo_time.
t2blmodes - T2 fit for blipmode methods.
ISA Function Programming

\texttt{tlinv} - T1 inversion recovery fit via IMND\_low\_value - IMND\_range\_value.
\texttt{tlepiat} - T1 fit for T1\_EPI (Look-Locker method) via ACQ\_inversion\_time.
\texttt{tlinvacq} - T1 inversion recovery fit via ACQ\_inversion\_time.
\texttt{tlsat} - T1 saturation recovery fit via ACQ\_recov\_time.
\texttt{tlmod} - Same as \texttt{tlinv} but for MSME\_MOD datasets fit via ACQ\_vd\_list.
\texttt{tblmodes} - T1 inversion recovery fit for blipmode methods.
\texttt{T1InvRephFisp} - Monoexponential fit of T1* and calculation of T1, T2 and Proton Density for TRUE FISP.
\texttt{diffread} - Diffusion fit for read diffusion gradient direction (according to the Stejskal-Tanner formula, fitting via IMND\_diff\_Strength).
\texttt{diffslice} - Diffusion fit for slice diffusion gradient direction.
\texttt{diffb} - Same as \texttt{diffslice} but fitting via IMND\_diff\_b\_value instead of gradient strength.
\texttt{diffb\_pvm} - Diffusion fit with variable gradient strength fitting via M\_DIFFSE\_DiffGradB
\texttt{dtraceb} - Diffusion fit with variable gradient strength fitting via PVM\_DwEffBVal
\texttt{dynst} - Dynamic studies (no fit at all; the data points are connected with straight lines).
\texttt{dynstsub} - Same as \texttt{dynst} but the very first (reference) image intensities are subtracted from the others; when one calculates parameter images with this function the pictures with the reference image subtracted are obtained.
\texttt{evolution} - Same as \texttt{dynst} for dynamic studies. X-axis values (in sec) are loaded from ACQ\_time\_points.
\texttt{evolutionsub} - Same as \texttt{dynstsub} for dynamic studies. X-axis values (in sec) are loaded from ACQ\_time\_points.
\texttt{gammavariat} - 5 parameters fit function for bolus dynamic studies from ACQ\_time\_points.

Such a wide variety of ISA functions doing practically the same things is explained by the fact that the acquisition methods use parameters with the same names in very different ways so that practically each actual acquisition method requires specific ISA function designed especially for it.
The information described in this section can be used to play with macros executing various ISA functions in batch mode. To understand the following information adequately one has to have some basic knowledge of the ParaVision Command Dispatcher and of writing shell scripts (see the corresponding Unix manpage, man sh).

The following commands drive the ISA package:

pv IsaEnter <modename> - Enter or reenter ISA. Mode name, if specified, can be used by the mode selector if more than one mode for the current dataset is known.

pv IsaExit - Exit from ISA.

pv IsaRoiDialog - Open ROI dialogue from ISA.

pv IsaRoiAdd - Add new ISA ROI followed by mousing.

pv IsaRoiClone - Add cloned ISA ROI followed by mousing.

pv IsaRoiModify - Modify current ISA ROI via mousing.

pv IsaRoiRefresh <label> - Refresh ROI with the specified label, all ISA_* ROIs if the label is "**" or the current ROI if the label is not specified.

pv IsaRoiRm <label> - Remove ROI with the specified label, all ISA_* ROIs if the label is "**" or the current ROI if the label is not specified.

pv IsaClean - Remove all curves.

pv IsaRoiAddConsist - Add ROI consistency callback - for test purposes only.

pv IsaRoiRmConsist - Remove ROI consistency callback - for test purposes only.

pv IsaEditList <names> - Edit list of parameters - for test purposes only.

pv IsaEditSingle <name> - Edit single parameter - for test purposes only.

pv IsaEditArray <name> - Edit single array parameter - for test purposes only.

pv IsaRoiEval - Calculate mean values of data points for ROI.

pv IsaFuncSelect <name> - Select specified ISA function or select interactively if the name is not specified.

pv IsaFuncEdit <name> - Edit specified ISA function or edit the current ISA function if the name is not specified.

pv IsaModEdit - Edit mode selector.
ISA Function Programming

pvIsaFuncCop <from> <to> “Yes” - Copy from-function into to-function. Missed arguments are requested interactively. If “Yes” is not specified confirmation is requested.

pvIsaFuncRen <from> <to> “Yes” - Rename from-function into to-function. Missed arguments are requested interactively. If “Yes” is not specified confirmation is requested.

pvIsaFuncDel <name> “Yes” - Delete specified ISA function. Missed argument is requested interactively. If “Yes” is not specified confirmation is requested.

pvIsaModDel “Yes” - Delete mode selector. If “Yes” is not specified confirmation is requested.

pvIsaFuncDetach - Detach ISA function.

pvIsaModDetach - Detach mode selector.

pvIsaParxRead <filename> - Read ISA parameter class from the specified file. If the filename is not specified it is asked for interactively.

pvIsaParxWrite <filename> - Write ISA parameter class into the specified file. If the filename is not specified it is asked for interactively.

pvIsaParxValue <name> <value> - Assign specified value to the parameter <name>. Missing arguments are requested interactively.

pvIsaParxEdit <names> - Open editor dialogue to edit list of parameters/classes. If the list is not specified it is asked for interactively.

pvIsaParxEditSync <names> - Edit list of parameters/classes synchronously. If the list is not specified it is asked for interactively.

pvIsaParxEditPrefs - Edit ISA user preferences.

pvIsaParxEditor <type> <names>... - Edit list of parameters - for test purposes only. The <type> may be “Single”, “Array” or “List”, the latter being the default.

pvIsaFuncInit - Initialize ISA function.

pvIsaFuncPlot - Plot ISA function.

pvIsaFuncExec - Execute (fit) ISA function for ROI data.

pvIsaImageFuncExec “ROI” “Yes” - Execute (fit) ISA function for image pixels (calculate parameter images). If ROI is specified, the current ROI is valid and the current masking mode is Cut Away or Overlay calculating images is performed only inside that ROI, otherwise for the whole image. The specification of the optional argument Yes prevents the user from the dialog confirmation.
pvIsaAllFuncExec “ROI” <inpath> <outpath> -
“NO_OVER” <ISA mode>: As pvIsalimageFuncExec, but calculates parameter images for all the slices of the entire dataset. This command does not require the ISA dialog to be opened before issuing the command, so it is usable in automatic postprocessing scripts. By default, fitting is done for all pixels of the image, and the resulting dataset <outpath>, if already existing, is automatically overwritten. If “ROI” is specified, fitting is performed inside the current ROI only. If “NO_OVER” is finished, the output dataset, if existing, is not overwritten, but the nearest free procno is used. An optional ISA mode name should be given, if the input dataset defines more than a single usable ISA mode.

pvIsalimageFuncPlot “ROI” “Yes” - Plot the ISA function for image pixels (generate artificial images). If ROI is specified, the current ROI is valid and the current masking mode is Cut Away or Overlay generating images is performed only inside that ROI, otherwise for the whole image. The specification of the optional argument Yes prevents the user from the dialog confirmation.

pvIsaTableMenu <cmd> - ISA table interactions. The <cmd> argument may be:
vp - Vary selected parameters.
fp - Fix selected parameters.
cp - Edit selected parameters.
adp - Take back selected data points.
rdp - Remove selected data points.
ep - Edit selected data points.
edp - Edit selected data points.
rc - Remove all curves.
dc - Deselect everything.

pvIsaExport <type> <file> - Export ISA data depending on the <type> argument into the specified file. If the filename is not specified it is asked for interactively. The <type> may be:
PSBW - Black and white PostScript.
PSColor - Color PostScript.
TextData - ASCII text (this is the default if no type is specified).

pvIsaOrigDataset - Switch to the original dataset from which the current image was calculated.

pvIsaShowMore <state> - Set the current state of the More buttons toggle. <state> is either “No” or “Yes”; if it is not specified it is taken from the current toggle.
ISA Function Programming

state which is not changed in this case.

`pvIsaSetExec <state>` - Set the current state of the Points Only/Pnts & Curves mousing option menu. `<state>` is either “Eval” or “Exec”; if it is not specified it is taken from the current mousing mode which is not changed in this case.
ISA Function Programming