1. Introduction ........................................................................................................5
  1.1 Conventions................................................................................................5
2. Editor and Linker..................................................................................................7
3. Programming ......................................................................................................10
  3.1 Structures ....................................................................................................10
  3.2 Global SCRIPT System Variables .................................................................11
  3.3 Protecting Your Individual SCRIPT Program ............................................11
  3.4 Programming Elements..............................................................................12
  3.5 Labels ........................................................................................................12
    3.5.1 SCRIPT System Labels ...........................................................................13
  3.6 Setting up the ECW ..................................................................................14
4. SCRIPT System Variables ............................................................................17
5. Macro Commands ............................................................................................20
  5.1 Control Parameters ....................................................................................20
  5.2 Setting Parameters ....................................................................................21
  5.3 Test Sampling ............................................................................................22
  5.4 Starting a measurement ...........................................................................22
  5.5 Saving Measured Data ............................................................................23
6. Commands for Analysis ................................................................................25
  6.1 Loading Measurement Data into the Analysis Program .........................25
  6.2 Saving Analysed Data ..............................................................................26
  6.3 Initialization of SIM ................................................................................27
  6.4 Setting up SIM ........................................................................................28
  6.5 Fitting an Impedance Spectrum .................................................................28
7. Acquisition and Control of External Signals .............................................30
  7.1 Analog Control Output ............................................................................32
8. Data Output ......................................................................................................34
8.1 Output to Screen ............................................................................ 34
8.2 Graphic Data Export and Paste to CAD ....................................... 34
8.3 Output of Data to an ASCII-File..................................................... 35

9. Document Handling (Graphics & Text) ....................37
9.1 Loading & Saving of Graphics & Texts ........................................ 37
9.2 Printing a Document .................................................................... 38
9.3 Comments....................................................................................... 38
  9.3.1 File Description Blocks (a Short Reminder).................................39
9.4 Inserting Text Into a Graphical Document ................................. 40
9.5 Text Attributes................................................................................ 40

10. Message Boxes ............................................................................43
10.1 Resetting the MESSAGE-Function ............................................. 43
10.2 Message Box Text........................................................................ 43
10.3 Display Message Box................................................................... 44
10.4 Message Box Specials................................................................. 45
10.5 Progress Indicator Functions ..................................................... 45

11. File Handling.................................................................................46
11.1 General.......................................................................................... 46
11.2 Directories..................................................................................... 46

12. Communication- and Dialogboxes .......................51
12.1 Input Box....................................................................................... 51
12.2 Checkbox ...................................................................................... 54
12.3 Toggle Switches........................................................................... 56
12.4 Listbox .......................................................................................... 58
12.5 Pop-Up Menus ............................................................................. 60

13. Graphics.........................................................................................61
13.1 The Structure of the Graphical Resources in SCRIPT .......... 62
13.2 Buttons.......................................................................................... 63
13.3 Embedding User-Defined Graphics............................................ 66
14. Creation and Export of Graphics Created by SCRIPT .................................................................67

15. Advanced Programming: Script Functions, Special Labels, Menu Techniques ..........................80
   15.1 Script Functions .................................................................................................................. 80
   15.2 Complex Menus .................................................................................................................. 82
1. Introduction

The THALES-software package offers a broad variety of tools to provide nearly any technique of measurement and analysis relevant in electrochemical investigations. It is quite easy to manually select a method for measurement, analysis and documentation which fits the application, but often it is useful to create a procedure which automatically runs a sequence of measurement, analysis and documentation routines. This makes measurements comfortable and easy to handle – even for untrained staff.

SCRIPT enables each user to create his own procedure which exactly does what he needs. SCRIPT sequences are written in an easy-to-learn BASIC-like language which uses a lot of powerful macro-commands. If needed, the complete set of andiBASIC commands is available. It allows a more detailed programming. The SCRIPT-Editor offers a versatile Help function and a highlighting of commands, variables and remarks.

Zahner has written a lot of scripts for special applications such as COLT or CIMPS. They come together with the corresponding option. In the standard Thales package a few demo sequences are included. They will help you getting an idea how to program with SCRIPT. You can find the demo scripts in the folder c:\thales\script of your PC hard drive.

This manual uses the following colors to highlight macro commands, SCRIPT system variables, andiBASIC commands, data statements, string constants and software remarks in the same way ZEdit does.

- **Macro commands**: red + bold
- **SCRIPT system variables**: black + bold
- **andiBASIC commands**: lilac
- **Software remarks**: green
- **String constants**: blue
- **Data**: cyan

1.1 Conventions

SCRIPT is sensitive to capital and lower case letters. All macro commands are written in capital letters only, all andiBASIC commands in lower case letters only. Most system variables have the first letter in upper case, the rest in lower case letters. User variables may be defined as needed.

The ECW file system software distinguishes between lower and upper case letters whereas Windows does not. Therefore, it was necessary to find conversion rules between the two systems. If a filename defined in Thales contains an upper-case letter, it is passed to Windows with an extension containing “0”s and “1”s. The extension represents the file name where a “0” stands for a lower-case letter and a “1” stands for an upper-case letter. Even though this system works perfectly, the filenames do not look
too good under Windows. Therefore, we recommend to use lower-case letters only for filenames.

The SCRIPT-Editor allows to write programs without line numbers as you may be used to do from other editors. You are allowed to write more than one statement in one line. Then, the statements are separated by a colon (:).

SCRIPT as well as andiBASIC variables have not to be defined. The type is defined by their ending character (identifier):

<table>
<thead>
<tr>
<th>Type</th>
<th>Identifier</th>
<th>Range</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>-</td>
<td>-10^{-308} to +10^{306} (+-10^{-308})</td>
<td>a, A, Value, sun</td>
</tr>
<tr>
<td>Integer</td>
<td>%</td>
<td>-2.147<em>10^9 to +2.147</em>10^9</td>
<td>a%, A%, Value%, sun%</td>
</tr>
<tr>
<td>Integer Array</td>
<td>%()</td>
<td>-32768 to +32767</td>
<td></td>
</tr>
<tr>
<td>Complex</td>
<td>&amp;</td>
<td>equal to real numbers</td>
<td>a&amp;, A&amp;, Value&amp;, sun&amp;</td>
</tr>
<tr>
<td>Logical</td>
<td>,%, %(), &amp;</td>
<td>=0 =&gt; no / false&lt;br&gt;&amp;&lt;0 =&gt; yes / true</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>$</td>
<td>All characters</td>
<td>a$, A$, Value$, sun$</td>
</tr>
</tbody>
</table>

SCRIPT shares its variables with the SCRIPT runtime software. To avoid interference with existing variables, use for instance leading characters such as SC_. Example: SC_a, SC_Value%, SC_sun$.
2. Editor and Linker

SCRIPT procedures are written and edited in the ZEdit text editor program. You get there by the following steps:

To start writing a new script:
**Thales Main Menu -> Script -> Edit Script -> Edit Text**

To edit an existing script:
**Thales Main Menu -> Script -> Edit Script -> File Operations (select device/path/file) -> Edit Text**

With the ZEdit editor you are able to write a SCRIPT text as you are used to do in a standard text editor. Line numbers are not necessary. Furthermore, ZEdit is able to highlight different text elements such as macro commands, strings, remarks, etc. This helps to have a clearly arranged program text.

While editing a script we recommend to keep the ZEdit open. Simply change the task by clicking into the Thales window. Here you may link your script by clicking on the **LINK** button. The script text is automatically checked for erroneous statements and typing errors before it is linked. Logical errors or violations of parameter ranges are not detected by the checking procedure. You may also call the check procedure by clicking on the **CHECK** button.
In order to exit ZEdit click on the close button of the ZEdit window or use the middle mouse button.

After linking a new button is created on the SCRIPT page. The script is executed by clicking on this new button.

An empty SCRIPT page looks like this:

![SCRIPT page]

The page is roughly divided into three sections. At the top of the screen the title of the program is shown. At the right hand side, five buttons offer the basic control functions of SCRIPT. The area at the left (empty in the picture) is reserved for SCRIPT icons.

Scripts are written and edited in ZEdit. An entry to the editor main menu has been provided by the button Edit SCRIPT on the SCRIPT page. Here you may open and save script source files. Enter the editor page via the “edit text” button. Alternatively you can open the ZEdit directly via the pull down menu by touching the left side of the Thales task bar.

Edit your script in ZEdit. You may keep the ZEdit window open while testing the script. Up to nine scripts may be held in one script text.

In order to link a new script text (make a runtime file out of the
(script text) click on the button *Link SCRIPT*. If the linking was successful the message “Link successful” is displayed. Prior to the linking procedure the script text is checked for syntax errors.

Each script text may be saved to the hard drive as a text file using the disk operation functions of the editor main menu or by means of the file -> “export” function. By default, the script texts are saved to the folder, selected for THALES text-files (initially c:\thales\examples\texts). The folder may be changed by the user as needed. You then may load a script text each time you need it. It has to be linked anew after loading to create a runtime version.

The Linker integrates the script command sequence in a reserved user program area of the SCRIPT software so that a new version of SCRIPT is generated in the RAM. If you do not save this version, it will be lost after the Thales software is restarted (cold-start). If you want to have the linked scripts available directly after stating up Thales (without having to load and link them manually), you may save it to the hard drive by clicking on the *Save Program* button.

Saves the runtime version of the SCRIPT routines to the hard drive. Keep the file path / name in mind - all linked scripts are available then directly after starting the Thales software by browsing through the “EXE” function and selecting the application desired.

By default, the SCRIPT software is saved to:

- **c:\thales\examples\applications** (PC hard drive, if no ECW hard drive is installed) or
- **root directory** (ECW hard drive, if installed)

SCRIPT allows to have up to nine scripts linked at the same time. The same number of buttons are available in the SCRIPT page. The buttons are implemented automatically. How to create and edit user-defined buttons is described later.

Multiple scripts may be written/edited and checked/linked either separately or as one script text. Each script has to get its individual number. For each script number a certain program area is reserved. Therefore, if you link a script with a number which already exists in memory, the old script will be replaced. Thus, the programmer has to organize the script numbers carefully.
3. Programming

3.1 Structures

The SCRIPT technique has been developed to combine different measurements and analysis methods in one script procedure. This provides a high comfort and an easy handling. With SCRIPT you are able to execute even complex methods at the push of a button.

A script is a sequence of macro commands and/or andiBASIC commands written and edited within a ZEdit text. The first line of each script must read

\[
\text{SCRIPT}\ n \quad (\text{SCRIPT1}…9)
\]

\(n\) represents a numeric value in the range of 1 to 9. It identifies the scripts 1 to 9. The last line of each script must read

\[\text{SCRIPT\_END}\]

Thus, a script text has to have the following structure:

\[
\begin{align*}
\text{SCRIPT1} & \quad \text{start of script 1} \\
\text{Command 1} & \\
\text{Command 2} & \\
\text{.......} & \\
\text{Command N-1} & \\
\text{Command N} & \\
\text{SCRIPT\_END} & \quad \text{end of script 1}
\end{align*}
\]

Up to nine scripts corresponding to up to nine graphic buttons can be linked to one script text file. Write the complete description in one text file or use the \textit{Append} option of \textit{ZEdit} to concatenate separate text files and yield the structure shown below.

| SCRIPT System statements | \\
|-------------------------| \\
| **SCRIPT1** | Definition of script1 \\
| \textit{.} | \\
| \textit{..} | \\
| \textit{SCRIPT\_END} | \\
| **SCRIPT2** | Definition of script2 \\
| \textit{.} | \\
| \textit{..} | \\
| \textit{SCRIPT\_END} | \\
| \textit{.} | \\
| \textit{..} | Definition of further scripts |
3.2 Global SCRIPT System Variables

SCRIPT provides three *SCRIPT system statements* which are valid for all scripts. These statements have to be placed before the first script text starts.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SCRIPT_DEVICE</code></td>
<td>sets the number of the device where the script files (buttons, graphics, etc.) are located</td>
</tr>
<tr>
<td><code>SCRIPT_PATH$</code></td>
<td>sets the path where the script files (buttons, graphics, etc.) are located</td>
</tr>
<tr>
<td><code>SCRIPT_PLOT$</code></td>
<td>defines a list of graphic files used in the script. Each element of the list can be plotted from the script using the <code>SCREEN_PLOT()</code> macro.</td>
</tr>
</tbody>
</table>

The global SCRIPT system variables have to be placed before the first script.

3.3 Protecting Your Individual SCRIPT Program

As described above the script graphic resources are saved by default in the c:\Thales and c:\Thales\Script folders on the PC hard drive or in the Root directory and the Script folder of the ECW hard drive (if installed). These folders are deleted during an update procedure. Therefore it is a good idea to use specific data paths by assigning the `SCRIPT_PATH$` and `SCRIPT_PLOT$` variables accordingly. Also you may backup the default folders before updating the Thales software.

To protect your individual SCRIPT we suggest:

*When you test a script make sure to save the corresponding text source to a file before you switch off the system.* Whenever the runtime version of your SCRIPT will get lost you may easily reconstruct that SCRIPT by linking the corresponding text source(s) again.

*Changes of the elementary graphic files stored in the directory 'script' will be lost during an installation or update.* If you decide to create your individual graphics, save the graphic files in a different subdirectory (not in the subdirectory 'script'). Create a new subdirectory, for instance 'my script'. Load, edit and save all individual graphics from/to 'my script'. In order to link your individual graphics to the runtime program use the global SCRIPT variables defining the graphic's path

`SCRIPT_DEVICE=65`
There are three types of elements available for SCRIPT programming:

1. SCRIPT macro commands
2. SCRIPT system variables
3. andiBASIC commands

By the macro commands predefined complex procedures are carried out whereas andiBASIC commands allow to program with advanced features. Macro commands are very easy to use and may be taken from the help list of ZEdit. Most of them use a couple of parameters which have to be defined in the script routine.

The macro MEAS_OPEN_EIS(Device, Path$, File$) for example opens an EIS spectrum file from the hard drive. The parameters Device, Path$ and File$ define the unit, the partition, the path name and the file name, respectively. All parameters are needed to clearly identify the file.

Other macros do not need additional parameters. SETUPECW for example applies the full set of (edited) ECW parameters to the system. The parameters are edited separately through system variables.

SCRIPT system variables represent the value of a certain parameter. Some of them are set by the system and can be read by the user (e.g. Pact returns the current potential), others are defined by the user to set a certain parameter (e.g. Pset defines the output set-potential of the potentiostat). All SCRIPT system variables allow values only in a certain range. Some have only a range of two numbers (0 and –1) to define a state (e.g. Pot defines the state of the potentiostat, where Pot = 0 means potentiostat off and Pot = -1 means potentiostat on). SCRIPT uses the standard logic states used for programming:

\[
\begin{align*}
0 & = \text{off} \\
-1 & = \text{on}
\end{align*}
\]

To be able to program scripts using andiBASIC commands, a more detailed knowledge of the system and some programming skills is needed. AndiBASIC commands are described in the andiBASIC Programming Manual which is available optionally.

**3.5 Labels**

ANDI-Basic allows the use of labels. Label names are a sequence of characters (e.g. START). To define a label use the name plus a double colon “::” (e.g. START::). Labels are mainly used to define jump destinations but also to optically structure a script.
Labels are used in common with the SCRIPT environment. To avoid a double definition, user-defined labels should start with the leading characters \texttt{Sn_}, where \( n \) is the number of the current script.

Example: \texttt{S1(ERR1::}

<table>
<thead>
<tr>
<th>not recommended</th>
<th>better</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{SCRIPT1}</td>
<td>\texttt{SCRIPT2}</td>
</tr>
<tr>
<td>onBRANCH%gotoFIT_END</td>
<td>onBRANCH%gotoS1_FITEND</td>
</tr>
<tr>
<td>FIT_END::</td>
<td>S1_FITEND::</td>
</tr>
<tr>
<td>SCRIPT_END</td>
<td>SCRIPT_END</td>
</tr>
<tr>
<td>\texttt{SCRIPT2}</td>
<td>\texttt{SCRIPT2}</td>
</tr>
<tr>
<td>onBRANCH%gotoFIT_END</td>
<td>onBRANCH%gotoS2_FITEND</td>
</tr>
<tr>
<td>FIT_END::</td>
<td>S2_FITEND::</td>
</tr>
<tr>
<td>SCRIPT_END</td>
<td>SCRIPT_END</td>
</tr>
</tbody>
</table>

### 3.5.1 SCRIPT System Labels

Some labels are reserved for special purposes within the script environment. They can be handled in the same way as all other labels but they work slightly different. In the current version of SCRIPT there are two SCRIPT system labels, \texttt{START} and \texttt{SUB\#}.

**START::**

The sub-routine following the \texttt{START::} label is executed automatically, as soon, as the script menu is entered. It can be implemented in any of the scripts \texttt{SCRIPT1} to \texttt{SCRIPT9}. The sub-routine is terminated by a \texttt{return} statement. The programmer must take care, that this procedure is not entered by accident during the script-execution. A \texttt{goto} statement, for instance pointing to the regular script end directly before the \texttt{START} label can avoid this.

**Example**

\begin{verbatim}
SCRIPT1
.. 'normal script execution entered by button1'
.. goto SCRIPT1\_END
START:\::
'program performing the startup procedure'
\end{verbatim}
SUB1:: The procedure following the labels SUB1:: ... SUB9:: are able to run in parallel to a main routine in the foreground

SUB9::

SUB%

Enables the corresponding sub-routine

This feature is useful for instance, if actual measuring data should be displayed and refreshed, while the script main menu is open. To activate one of the nine possible sub-routines you have to set the SCRIPT system variable SUB% to the corresponding value.

The following example demonstrates, how the START procedure plots a instrument display and activates the task SUB1 by setting the reserved variable SUB% to one. The program after SUB1 performs then measuring data acquisition and display refresh after each second.

Example

```script
SCRIPT1
.. 'normal script execution by push button'
.. goto SCRIPT1_END

START::
DISPLAY(100,360,"Temperature","C") 'x, y-position, header left, right'
SUB%=1 'enables the execution of the SUB1:: procedure'
return

SUB1::
if ti<tref then return 'reads time ti & returns, smaller reference'
tref = ti + 1 'sets next display time'
ACQUIRE(1) 'reads measuring channel #1 to ACQ_VAL'
DISPVAL(100,360,fnSTR$(ACQ_VAL)) 'displays ACQ_VAL'
return

SCRIPT1_END::
SCRIPT_END
```

3.6 Setting up the ECW

The ECW parameters can be set up in different ways:

1. You may set parameters manually in the TestSampling page. These settings will be kept until they are edited or changed by the procedures described in 2. and 3.
2. You can define single parameters like Pset (DC potential), Cset (DC current), Pot (state of the potentiostat) and many others. This method is good to change only one or a few parameters.
3. You can load a previously configured and saved measurement or analysis file to set up all parameters of a method in only one step. All relevant parameters of a
method are saved along with the data. After loading, the parameters are automatically set to the values stored in this file. This method is very comfortable if you want to set up the complete set of parameters in only one step. MEAS_OPEN_EIS will load a set of EIS measurement parameters, ANA_OPEN_MOD will load a pre-defined model file to SIM, etc. If you now perform the corresponding measurement or analysis procedure, it is done with the parameters loaded.

Ways 2. and 3. will set the SCRIPT system parameters, but do NOT automatically set up the ECW hardware. This is done by the macro command SETUPECW. It sends the complete set of parameters to the operating system of the ECW which sets up the hardware accordingly.

The standard procedure to set up a script is to setup each single method contained in the script, do a measurement or analysis routine and save the data as an instruction file. Now you only have to load these instruction files and start the measurement. You may do this for a complete sequence of measurement and analysis methods in one script:

```
: MEAS_OPEN_EIS(instruction 1)
: MEAS_EIS
: MEAS_SAVE_EIS(experiment 1)
: ANA_OPEN_EIS(experiment 1)
: ANA_PLOT_EIS(number)
: MEAS_OPEN.CV(instruction 2)
: MEAS.CV
: MEAS_SAVE.CV(experiment 2)
```

This is only a schematic of a script structure. For the correct macro parameters please refer to the corresponding chapters.

If you need to edit parameter settings often (e.g. to adapt the experiment to changed conditions), it is not necessary to modify the SCRIPT source file. Instead, the instruction file components can be replaced by adapted ones. For instance, within an existing script an equivalent circuit has to be fit to experimental impedance data. If it seems appropriate to replace the equivalent circuit by an modified one, the new model should be tested in direct mode and stored using the file name of the former one. The new model file will replace the old model file and the new model is automatically used in the script from now on.

Other ways to include a new model are:
> Rename the old model and save the new model with the file name of the old one (which is used in the script).
> Replace the file name in the script with the file name of the new model.

SCRIPT provides a set of macro commands which are used to pass and get parameters to and from the operating system/hardware. They allow to get information about measured and set parameters and to send the current set of parameters to the system. If, for instance, depending on the actual rest-potential different actions should take place, process data and system variables like rest potential, cell current, EIS-frequency, etc. can be read and set by SCRIPT.

The macro commands **CURRENT** and **POTENTIAL**, e.g., are getting the currently measured current and potential data from the *Electrochemical Workstation* (ECW). **SETUPECW** is sending the currently edited parameter set to the ECW. Therefore, edited parameters will only become effective after the command **SETUPECW** was executed.

The usage of SCRIPT system routines like **SETUPECW** is very easy but it has to be used carefully. Thales takes care that *instruction files* created in direct mode do not contain invalid parameters. In SCRIPT, on the other hand, the user has to take care of the range limits of each parameter he sets. E.g.: Trying to input DC-potentials beyond the range of ± 4V in the *TestSampling* page are not allowed (if no U-buffer is selected) and therefore will be refused by the Thales parameter check routines. If, however, the user is defining a parameter out of range in SCRIPT, this faulty setting will be accepted while editing and will only be detected when executing the script. In that case the script execution will be terminated.
4. SCRIPT System Variables

SCRIPT provides a set of variables which are used to control the hardware parameters. The names of these variables are protected keywords and must not be used as user-defined variable names. These SCRIPT system variables may be set or read, as needed. Please remember that the programmer has to take care of the parameter ranges of each variable when setting it.

### DC-potential & DC-current

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pact</td>
<td>Actual DC-potential</td>
<td>Volt</td>
</tr>
<tr>
<td>Cact</td>
<td>Actual DC-current</td>
<td>Ampere</td>
</tr>
<tr>
<td>Pset</td>
<td>New set potential</td>
<td>Volt</td>
</tr>
<tr>
<td>Cset</td>
<td>New set current</td>
<td>Ampere</td>
</tr>
<tr>
<td>Bfact</td>
<td>Gain factor of buffer amplifier</td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>Potential of reference electrode</td>
<td>Volt</td>
</tr>
</tbody>
</table>

### Potentiostat Modes

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pot</td>
<td>Nominal state of potentiostat. Pot stores the</td>
<td>Pot = 0 OFF</td>
</tr>
<tr>
<td></td>
<td>actual state of the potentiostat and will be used</td>
<td>Pot = -1 ON</td>
</tr>
<tr>
<td></td>
<td>to switch ON/OFF the potentiostat.</td>
<td></td>
</tr>
<tr>
<td>Gal</td>
<td>Mode setting of potentiostat selecting</td>
<td>Gal = 0 potentiostat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gal = -1 galvanostat</td>
</tr>
<tr>
<td>GAL</td>
<td>Mode setting pseudogalvanostatic or</td>
<td>See list below</td>
</tr>
<tr>
<td></td>
<td>galvanostatic mode</td>
<td></td>
</tr>
<tr>
<td>PFree%</td>
<td>Flag to activate the rest potential mode.</td>
<td>PFree% = 0 -&gt; OFF</td>
</tr>
<tr>
<td></td>
<td>OFF = fixed potential</td>
<td>PFree% = -1 -&gt; ON</td>
</tr>
<tr>
<td></td>
<td>ON = rest potential mode</td>
<td></td>
</tr>
</tbody>
</table>

### Setting of potentiostatic, galvanostatic and pseudo-galvanostatic mode

<table>
<thead>
<tr>
<th>Gal</th>
<th>GAL=1</th>
<th>GAL=0</th>
<th>GAL=-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>potentiostatic</td>
<td>pseudo-galvanostatic</td>
</tr>
<tr>
<td>-1</td>
<td>galvanostatic</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

! When the rest potential mode will be activated the mode setting must be selected Gal=0 / Potentiostat.

! Please note, that in contrast to all other unit definitions, the AC amplitude is defined in one thousandth (1/1000 = “milli”) units.
### AC-settings

<table>
<thead>
<tr>
<th><strong>Ampl</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC-amplitude [milliVolts] in potentiostatic, rest potential and pseudo-galvanostatic mode, AC-amplitude [milliAmpere] in true galvanostatic mode</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Frq</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency of excitation signal [Hertz]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Nw</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count of measuring periods</td>
<td></td>
</tr>
</tbody>
</table>

### Setup of potentiostats without RMux

<table>
<thead>
<tr>
<th><strong>DEV%</strong></th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets and returns the number of the selected device</td>
<td>DEV% = 0</td>
<td>main potentiostat</td>
</tr>
<tr>
<td></td>
<td>DEV% = 1</td>
<td>EPC40 channel1</td>
</tr>
<tr>
<td></td>
<td>DEV% = 2</td>
<td>EPC40 channel2</td>
</tr>
<tr>
<td></td>
<td>DEV% = 3</td>
<td>EPC40 channel3</td>
</tr>
<tr>
<td></td>
<td>DEV% = 4</td>
<td>EPC40 channel4</td>
</tr>
</tbody>
</table>

### Setup of potentiostats with RMux

<table>
<thead>
<tr>
<th><strong>DEV%</strong></th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets and returns the number of the selected device</td>
<td>DEV% = 0</td>
<td>main potentiostat</td>
</tr>
<tr>
<td></td>
<td>DEV% = 1</td>
<td>RMUX channel1</td>
</tr>
<tr>
<td></td>
<td>DEV% = 2</td>
<td>RMUX channel2</td>
</tr>
<tr>
<td></td>
<td>DEV% = 16</td>
<td>RMUX channel 16</td>
</tr>
<tr>
<td></td>
<td>DEV% = 17</td>
<td>EPC40 channel1</td>
</tr>
<tr>
<td></td>
<td>DEV% = 18</td>
<td>EPC40 channel2</td>
</tr>
</tbody>
</table>

### Selection of Connection Scheme

<table>
<thead>
<tr>
<th><strong>SC%</strong></th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selects connection scheme (see ECW, chap2)</td>
<td>SC% = 0</td>
<td>ECW connection scheme1</td>
</tr>
<tr>
<td></td>
<td>SC% = 1</td>
<td>ECW connection scheme2</td>
</tr>
<tr>
<td></td>
<td>SC% = 2</td>
<td>ECW connection scheme3</td>
</tr>
<tr>
<td></td>
<td>SC% = 3</td>
<td>ECW connection scheme4</td>
</tr>
<tr>
<td></td>
<td>SC% = 4</td>
<td>ECW connection scheme5</td>
</tr>
<tr>
<td></td>
<td>SC% = 5</td>
<td>ECW connection scheme6</td>
</tr>
</tbody>
</table>
5. Macro Commands

5.1 Control Parameters

The following functions offer selective control of the electrical parameters of the electrochemical workstation.

**POTENTIAL**
Returns the current DC-potential of the connected cell to the variable $P_{act}$

**CURRENT**
Returns the current DC-current through the connected cell to the variable $C_{act}$

**IMPEDEANCE($f$, $n$)**
Executes a single frequency impedance measurement at the frequency $f$ and by integrating $n$ sine waves. The measured impedance is stored in the variable $Z\&$. The quality of the measurement will be returned in the variable $E$. $E$ will take values between 0 and 1, where:
- 0 = worst case
- 1 = best case
- 0.99 corresponds to an impedance modulus uncertainty of about 1%.

**GETPARAM**
Returns the current settings of the control parameters to SCRIPT.

**SETUPECW (SETUPIM6)**
Passes the control parameters to the measuring program and will set-up the potentiostat to the newly defined state.

---

After loading an *instruction file* (MEAS\_OPEN\_*) the routine **GETPARAM** is called automatically. The potentiostat will be set in accordance with the loaded parameter set and that set will be passed to SCRIPT.

---

The user has to call **GETPARAM** if the potentiostat is to be used before loading an *instruction file*. After starting SCRIPT, the variables of the control parameters have been reserved, but no values are assigned. Using **SETUPECW** before calling **GETPARAM** will load the ECW hardware with invalid values and an erroneous behavior of the hardware might occur.
5.2 Setting Parameters

As described above the ECW hardware is totally set up when loading an *instruction file* (in principle any measurement data file). The only exception is: the *potentiostat* is *ALWAYS SWITCHED OFF* after loading. This is done for the safety of the hardware. If you want the measurement start at the set potential, the potentiostat must be switched on by the SCRIPT command

\[ \text{Pot} = -1. \]

If the potentiostat is not switched on prior to a measurement, the measurement will not be started at the set potential of the *instruction file*, but at the current open circuit potential.

The general syntax for setting control parameters by means of an example files is:

\[ \text{MEAS\_OPEN\_***}(\text{Device}, \text{Path$}, \text{File$}) \]

‘***’ stands for the measurement method.

\[ \text{Device} \]
specifies the logical drive where the example file is stored. Usually Device is always 65. In elder system configurations Device may have other values.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

If Device=65 (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

\[ \text{Path$} \]
string defining the data path of the file to be loaded

The subdirectories must be separated by a ‘\’ (backslash).

**Examples**

Device=2:Path$="test\test1"
selects the subdirectory *test1* of subdirectory *test* on the ECW hard drive (on elder systems).

Device=65:Path$="c:\thales\script"
selects the subdirectory *Script* of the subdirectory *Thales* on partition C of the PC hard drive.

\[ \text{File$} \]
string defining the file name
SCRIPT provides the following measurements methods to be loaded as instruction files:

- **MEAS_OPEN_EIS** ( Loads impedance spectrum data)
- **MEAS_OPEN_CV** ( Loads cyclic voltammogram data)
- **MEAS_OPEN_IE** ( Loads stationary I/E-curve data)
- **MEAS_OPEN_POL** ( Loads polarization measurement data)
- **MEAS_OPEN_ACQ** ( Loads data-acquisition set-up data)

**Examples**

- **MEAS_OPEN_EIS(65,"c:\thales\script","test1")**
  Loads the file `test1` located in the `thales\script` folder of the c-partition of the PC hard drive.

- **MEAS_OPEN_CV(2,"examples","test1")**
  Loads the file `test1` located in the `examples` folder of the ECW hard drive.

### 5.3 Test Sampling

The setting and the response of the potentiostat can best be checked on the TestSampling page of the EIS-program. This page can also be called from SCRIPT by means of the command `MEAS_TEST`. Once on the TestSampling page you can check the measurement as well as edit the parameters manually.

**MEAS_TEST** calls the TestSampling page of Thales

### 5.4 Starting a measurement

After setting the ECW hardware correctly you may like to start a measurement. This is done by a group of macro commands starting with “MEAS_”:

**MEAS_***

‘***’ stands for the measurement method to be started. SCRIPT is providing the following measurement methods:

- **MEAS_EIS** Starts an impedance measurement
- **MEAS_CV** Starts an cyclic voltammogramm
- **MEAS_IE** Starts an I/E-measurement
- **MEAS_POL** Starts an polarization measurement
5.5 Saving Measured Data

If a measurement was successful you may want to save the measured data along with the measurement parameters. You can do that with the command:

\[
\text{MEAS\_SAVE\_***}(\text{Device},\text{Path$},\text{File$},\text{Savemode})
\]

`***` stands for the measurement method. The parameters Device, Path$ and File$ are used in the same way as described in chapter 5.1 Setting Parameters.

**Device** specifies the logical drive where the example file is stored.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

If **Device=65** (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

**Path$** string defining the data path of the file to be loaded

The subdirectories must be separated by a ‘\’ (backslash).
If you want to overwrite an existing file, use the prefix ‘@’ in the pathname or in the filename (Path$="@test\test1" or File$="@measure1").

**Examples**

Device=2:Path$="test\test1"
selects the subdirectory *test1* of subdirectory *test* on the ECW hard drive (on elder systems).

Device=65:Path$="c:\thales\script"
sselects the subdirectory *script* of the subdirectory *thales* on partition c of the PC hard drive.

**Savemode** numerical value setting save-options

The value of Savemode is calculated by adding the values of the three lower bits. The following options are available:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 =&gt; manual input of comments enabled</td>
</tr>
<tr>
<td>Bit</td>
<td>Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>manual input of comments enabled</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>input of filename not enabled</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>ESC function enabled</td>
</tr>
</tbody>
</table>

Example

_Savemode=_5

Usage of the <ESCAPE>-function should be enabled if loops are used to avoid “never-ending” loops.

SCRIPT provides the following measurement methods to be saved:

- **MEAS_SAVE_EIS(** Saves an impedance spectrum
- **MEAS_SAVE_CV(** Saves a cyclic voltammogram
- **MEAS_SAVE_IE(** Saves an I/E-curve
- **MEAS_SAVE_POL(** Saves a polarization measurement
- **MEAS_SAVE_ACQ(** For future use
6. Commands for Analysis

6.1 Loading Measurement Data into the Analysis Program

After measuring and saving you may want to analyze your data. You can do this using the corresponding macro command of the family

\[ \text{ANA_OPEN}_*** (Device, \text{Path$}, \text{File$}) \]

‘***’ stands for the type of data (method) to be loaded. SCRIPT provides the following data types:

- \text{ANA_OPEN_EIS} \quad \text{Loads an impedance spectrum}
- \text{ANA_OPEN_MOD} \quad \text{ Loads a model file to SIM}
- \text{ANA_OPEN.CV}\quad \text{ Loads a CV-measurement}
- \text{ANA_OPEN_IE}\quad \text{ Loads an I/E-measurement}
- \text{ANA_OPEN_SETUP}\quad \text{ Loads a SIM instruction file (see chapter 6.3)}

The corresponding analysis program will load the measurement defined by the parameters Device, Path$, and File$.

Device specifies the logical drive where the example file is stored. Usually Device is always 65. In elder system configurations Device may have other values.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

If Device=65 (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

Path$ string defining the data path of the file to be loaded

The subdirectories must be separated by a ‘\’ (backslash).

Examples
Device=2:Path$="test\test1"
selects the subdirectory test1 of subdirectory test on the ECW hard drive (only in elder systems).

Device=65:Path$="c:\thales\script"
selects the subdirectory Script of the subdirectory Thales on partition c of the PC hard drive.

File$ string defining the file name

6.2 Saving Analysed Data

The save function writes the analyzed data to the hard drive. This function group reads:

\textbf{ANA\_SAVE\_***}(Device,Path$,File$,Savemode)

\textit{Device} specifies the logical drive where the example file is stored. Usually Device is always 65. In elder system configurations Device may have other values.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

\textbf{!} If Device=65 (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

\textit{Path$} string defining the data path of the file to be loaded

The subdirectory levels are separated by a ‘\’ (backslash).

\textit{File$} string defining the name of the file

If you want to overwrite an existing file, use the prefix '@' in the pathname or in the filename (Path$="@test\test1" or File$="@measure1").

\textbf{Examples}

Device=2:Path$="test\test1"
selects the subdirectory test1 of subdirectory test on the ECW hard drive.

Device=65:Path$="c:\thales\script"
selects the subdirectory Script of the subdirectory Thales on partition c of the PC hard drive.

\textit{Savemode} numerical value setting save-options
The value of Savemode is calculated by adding the values of the three lower bits. The following options are available:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1 =&gt; manual input of comments enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 =&gt; manual input of comments disabled</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1 =&gt; input of filename via input box enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 =&gt; input of filename via input box disabled</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1 =&gt; break by ESC function enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 =&gt; break by ESC function disabled</td>
</tr>
</tbody>
</table>

Example
Savemode=5

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>manual input of comments enabled</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>input of filename not enabled</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>ESC function enabled</td>
</tr>
</tbody>
</table>

Usage of the <ESCAPE>-function should be enabled if loops are used to avoid “never-ending” loops.

The following ANA_SAVE functions are available:

ANA_SAVE_EIS( Save an impedance spectrum
ANA_SAVE_MOD( Save an impedance model file
ANA_SAVE_CV( Save a cyclic voltammogram
ANA_SAVE_IE( Save a stationary I/E curve

6.3 Initialization of SIM

CLRSIM initializes SIM program

The evaluation program will be reset to the default parameters and all present data will be deleted. This function enables the user to reset SIM at any point during a running script and may prevent an uncontrolled overflow of SIM.
6.4 Setting up SIM

**ANA_OPEN_SETUP**(*Device, Path$, File$*)

The complete set of user settings such as display parameters (Bode, Nyquist, auto/manual scaling, plot modes, etc.), mathematical parameters (smooth parameter, fit convergence, etc.) may be selected for each application individually by loading the appropriate *instruction files*. All SIM parameters are set after loading automatically.

In order to create an instruction file call SIM, make your settings and save them to the hard drive. Files created in this way can be loaded as *instruction files* from SCRIPT.

*Device* specifies the logical drive where the example file is stored. Usually *Device* is always 65. In elder system configurations *Device* may have other values.

The following entries for ‘*Device*’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

! If *Device*=65 (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

*Path$* string defining the data path of the file to be loaded

The subdirectories must be separated by a ‘\’ (backslash).

**Examples**

*Device*=65:*Path*="c:\thales\script"

selects the subdirectory *Script* of the subdirectory *Thales* on partition C of the PC hard drive.

*File$* string defining the file name

---

6.5 Fitting an Impedance Spectrum

**FITEIS**(*Fmin, Fmax, Sdens, Ssource*)

**FITEIS()** runs a fit. The frequency range to be fitted and the number of fitting points are defined by the parameters in between the brackets (*Fmin, Fmax, Sdens, Ssource*).

*Fmin* lower end of frequency range to be fitted
$F_{max}$ upper end of frequency range to be fitted

$S_{dens}$ number of fitting point samples to be selected automatically

$S_{source}$ Optional parameter to indicate the sample point source:
0 (or missing parameter): Sample extraction from the smoothed curve
1: Sample extraction from the original measuring data
2: Sample extraction from the ZHIT curve
7. Acquisition and Control of External Signals

If your ECW is equipped with a data acquisition hardware module (TEMP/U, FE42, etc.), or if you have installed ‘Thales TCP/IP Net Instruments’ you are able to measure additional physical quantities like temperature, voltage, etc. along with the immediate electrochemical data. Before you use this feature check, if the appropriate additional channels in the ECW’s signal acquisition setup menu are activated as active logical displays. You can change the activation by clicking on the corresponding entry in the column ‘Display’ of the channel list. Note that the active logical display numbers (rightmost column nr) are starting with ‘0’. They may be arranged in any order, independent from the physical channel number (left column) by toggling them off and on in the appropriate sequence. Physical input channel numbers => 4096 correspond with NET input channels. They will be activated automatically after the registration of an NET input instrument.

If you intend to read an additional channel besides electrochemical data within the SCRIPT environment, use the ACQUIRE function. After reading you are able to plot the channel data in your SCRIPT window using the DISPVAL command. With the DISPLAY macro you can define an output box.

ACQUIRE(no)

Acquisition of additional channels

no = number of the logical display
The logical display number corresponds to the physical channel listed in the same line of the Signal Acquisition Setup page.

After the acquisition, the SCRIPT system variable ACQ_VAL is loaded with the value measured before with the ACQUIRE macro. If the acquired channel is a NET channel, the original data string sent by the NET device is available through the SCRIPT system variable ACQ_VAL$.
**DISPLAY(Xpos,Ypos,Text1$,Text2$[,Color [,Size]])**

Displays an output window with up to two headlines (Text1$ and Text2$).

- **Xpos** = X-coordinate of the output window
- **Ypos** = Y-coordinate of the output window
- **Text1$** = Header of the output window (part1)
- **Text2$** = Header of the output window (part2)
- **Color** = optional header text color information (see also chapter 9.5, ‘Text Attributes’). Default is Color =15 (white). Only the integer value of Color is used. Take Color = 0.1 for text in black color.
- **Size** = optional size information (default is 1, size 1, 0.75 and 0.5 is recommended)

**DISPVAL(Xpos,Ypos,Text$[,Color] ,Size])**

Prints the value of ACQ_VAL (or any other) into the window opened by DISPLAY(. The value has to be sent as a string. Therefore, it is usually necessary to convert the numerical value of ACQ_VAL into the string format, first (see example, and chapter ‘script functions’).

- **Xpos** = X-coordinate of the output window
- **Ypos** = Y-coordinate of the output window
- **Text$** = Text which is printed into the output window
- **Color** = optional 7-segment display color information (see also chapter 9.5, ‘Text Attributes’). Default is Color =3 (yellow).
- **Size** = optional size information (default is 1, size 1, 0.75 and 0.5 is recommended)

**Example**

In the following example, the temperature is measured until it exceeds a value of 30 degrees. The statement packa$,using"-###.#",ACQ_VAL converts the content of ACQ_VAL (a floating point variable) into the string a$ with four significant digits (accuracy one tenth of a degree) and adds a minus sign, provided the temperature is less than 0 degrees. An alternative representation is proposed as a remark line.

```plaintext
DISPLAY(200,316,"Temp/C","TC 0: NiCr/Ni") 'Define output-box'
DISP_LOOP:
  ACQUIRE(0) 'Label DISP_LOOP'
  packa$,using"-###.#",ACQ_VAL 'Measure temperature'
  DISPVAL(200,316,a$) 'Format result (ACQ_VAL)'
  'DISPVAL(200,316,fnVAL$(ACQ_VAL))' 'Plot the formatted value'
  pause 500 'Wait 500 milliseconds'
  if ACQ_VAL <30 goto DISP_LOOP 'Continue until T >= 30C'
```
7.1 Analog Control Output

If your ECW is equipped with an analog control output hardware module (DAC16/4, etc.), or if you have installed ‘Net Instruments’ with output feature, you are able to control additional physical quantities like temperature, pressure, etc. For instance, you can control the revolutions of an RDE. Before you use this feature check, if the appropriate additional channels in the ECW’s ACQ output control setup menu are present as active outputs. You can change the activation by clicking on the corresponding control panels after pressing the “edit output” button. Note that the active logical output controls are ordered according to their physical channel number, starting with ‘1’ for the first output of a DAC16/4. Selecting a physical channel number ‘0’ will close the corresponding panel. A new panel can be assigned by editing the (dimmed) empty panel located on the bottom of the list.

A physical channel => 4096 corresponds with NET output channels. The corresponding control panels will be activated automatically after the registration of an NET output instrument.

Syntax

`OUTA(Value,Channel)`

Sends the set-value ‘Value’ to the analog output available at the address ‘Channel’.
Example
In the following example, the revolutions of an RDE, coupled via TCP/IP is set between zero and 2000 upm. At every step of 100 rpm an EIS spectrum is recorded and saved. The function fnTNAME$( is used to create an automatic filename out of time and date.

Channel=4096  'a TCP/IP NET output shall be addressed corresponding to an RDE'

CONTROL_LOOP::
  for revolutions = 0 to 2000 step 100  'set up a for-next loop for revolutions'
    OUTA(revolutions,channel)          'put out the set value via TCP/IP'
    pause 1000                            'measure spectrum'
    MEAS_EIS
    MEAS_SAVE_EIS(65,“data”,fnTNAME$(0)) 'save the data under an automatic file name'
  next                                     'see chpt. “script functions” for fnTNAME$'
8. Data Output

Measured data and fitted data may be put out as graphics to different devices using output macros.

8.1 Output to Screen

A group of output macros display a graphical plot on the screen. The previously loaded data files are used. With the Cnt parameter you can define how many of them shall be plotted.

\[
\text{ANA_PLOT}_{***(Cnt)}
\]

Cnt defines the number of data sets to be plotted:

<table>
<thead>
<tr>
<th>Cnt</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>last file (=actual) being loaded</td>
</tr>
<tr>
<td>2</td>
<td>last two files</td>
</tr>
<tr>
<td>3</td>
<td>last three files</td>
</tr>
<tr>
<td>:</td>
<td>etc.</td>
</tr>
<tr>
<td>:</td>
<td>etc.</td>
</tr>
</tbody>
</table>

The maximum number of plots is defined in SIM Setup. How to load files into the memory is described later.

The following macros for a screen output are available:

- **ANA_PLOT_EIS** (Plot impedance measurement)
- **ANA_PLOT_MOD** (Plot impedance model)
- **ANA_PLOT_CV** (Plot CV-diagram)
- **ANA_PLOT_IE** (Plot I/E-diagram)

8.2 Graphic Data Export and Paste to CAD

\[
\text{ANA_XGRAPH}(x,y) \text{ or } \text{ANA_HPPLOT}(x,y[\text{Orientation,Size,Flag}])
\]

**ANA_XGRAPH**(x,y[,Orientation,Size,Flag]) is a graphical interface macro which is used to create an meta-file which is transferred to CAD. It is equivalent to the I/O-function 'Export Graphic' you can find generally in the graphic output pages of Thales. The parameters x and y define the offset in [mm] of the lower left edge of the graphic from the lower left edge of the document.
Optionally the graphic may be turned and re-sized by the parameters “Orientation” and “Size”:

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>straight</td>
</tr>
<tr>
<td>1</td>
<td>left turned</td>
</tr>
<tr>
<td>2</td>
<td>Upside down</td>
</tr>
<tr>
<td>3</td>
<td>right turned</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Size</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unchanged</td>
</tr>
<tr>
<td>&gt;1</td>
<td>up scaled</td>
</tr>
<tr>
<td>&lt;1</td>
<td>down scaled</td>
</tr>
</tbody>
</table>

The optional flag “Flag” specifies in particular, which parts of the standard graphic in the screen are excluded from the export function. “Flag” may be built also from a binary combination (Or-function) of the flag-table-values:

<table>
<thead>
<tr>
<th>Flag value</th>
<th>Bit set</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>none</td>
<td>all components present</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>curve graphic suppressed</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>curve legend suppressed</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>legend frame suppressed</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>circuit scheme suppressed</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>circuit table suppressed</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>measurement comment suppressed</td>
</tr>
<tr>
<td>2,4,8</td>
<td>2,3</td>
<td>unused</td>
</tr>
</tbody>
</table>

### 8.3 Output of Data to an ASCII-File

**ANA_XASCII(Device,Path$,File$)**

- **Device** specifies the logical drive where the example file is stored. As export data normally are destined to the PC, only Device =65 makes sense.

- **Path$** string defining the data path of the file to be loaded

The subdirectory levels are separated by a ‘\’ (backslash).

- **File$** string defining the name of the file

If you want to overwrite an existing file, use the prefix ‘@’ in the pathname or in the filename (Path$="@test\test1" or File$="@measure1").

**Examples**

Device=65:Path$="c:\thales\script"

selects the subdirectory *Script* of the subdirectory *Thales* on partition C of the PC hard drive.
The **ANA_XASCII** macro creates an ASCII-list of the measured data and saves it to the folder defined by the `Path$` parameter of the hard drive defined by the `Device` parameter with the filename `File$`. The ASCII-list will be formatted corresponding to the current *display setting* of SIM:

<table>
<thead>
<tr>
<th>Display setting</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bode</td>
<td>Data will be listed as impedance (modulus) and phase (argument).</td>
</tr>
<tr>
<td>Nyquist</td>
<td>Data will be listed as real part and imaginary part</td>
</tr>
<tr>
<td>Smooth active</td>
<td>smoothed data set will be appended to the data list</td>
</tr>
</tbody>
</table>
9. Document Handling (Graphics & Text)

Loading a predefined graphic or text file initializes the CAD-program and the ZEdit, respectively. Previously opened data in CAD or ZEdit will be deleted.

9.1 Loading & Saving of Graphics & Texts

The following open and save macros are available:

- **CAD_OPEN()** Loads a CAD-file
- **CAD_SAVE()** Saves data to a CAD-file
- **TEXT_LOAD()** Loads a text file
- **TEXT_SAVE()** Saves text to a text file

In order to load a graphic file into CAD or to load a text file to ZEDIT use:

```plaintext
###_OPEN_*** (Device,Path$,File$)
```

To save a graphic as a CAD file or a text to a text file use:

```plaintext
###_SAVE_*** (Device,Path$,File$)
```

### stands for CAD or TEXT
*** stands for OPEN or SAVE

Device specifies the logical drive where the example file is stored.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

- **Path$** string defining the data path of the file to be loaded
- **File$** string defining the name of the file
If you want to overwrite an existing file, use the prefix '@' in the pathname or in the filename (Path$="@test\test1" or File$="@measure1").

**Examples**
Device=2:Path$="test\test1"
selects the subdirectory test1 of subdirectory test on the ECW hard drive.

Device=65:Path$="c:\thales\script"
selects the subdirectory Script of the subdirectory Thales on partition C of the PC hard drive.

### 9.2 Printing a Document

If you like to print an opened document to the printer use:

\[ \text{CAD\_PLOT}() \]

The actual graphics is printed to the selected printer.

See \text{DOC\_OUT}() for detailed description.

### 9.3 Comments

The ANA\_REM\_*** macro command copies the *file description block* of the data loaded into SIM to the *SCRIPT system array text$()*.

This may be necessary if you want to integrate a description or parts of it into a graphics.

\[ \text{ANA\_REM\_***}(m, c) \]

** *** = file type (method)  
** m = number of the files loaded (last loaded = 1)  
** c = column of the comment

The following data types are available:

\[ \text{ANA\_REM\_EIS(} \]
gets comment from EIS spectrum data

\[ \text{ANA\_REM\_MOD(} \]
gets comment from SIM model data

\[ \text{ANA\_REM\_CV(} \]
gets comment from CV data

\[ \text{ANA\_REM\_IE(} \]
gets comment from IE data

Because it is possible to load more than one file into SIM, you have to specify which file you want to work with. The \( m \) parameter specifies the file counting from the file loaded last (=1) to the file loaded first (=n).
The columns $c$ are defined as follows:

<table>
<thead>
<tr>
<th>column1</th>
<th>column2</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g.: System:</td>
<td>dummy cell</td>
</tr>
<tr>
<td>Potential:</td>
<td>-1.000V</td>
</tr>
<tr>
<td>Current:</td>
<td>-1.0e-3A</td>
</tr>
<tr>
<td>.....</td>
<td>.......</td>
</tr>
</tbody>
</table>

Each index of the `text$()` array contains one line of the copied file description block.

**Examples**
- **ANA_REM_EIS(1,1)** Copies the descriptor titles of the last EIS-spectrum being loaded.
- **ANA_REM_MOD(3,2)** Copies the comments of the third last model being loaded to SIM.

### 9.3.1 File Description Blocks (a Short Reminder)

**EIS data**

<table>
<thead>
<tr>
<th>Line</th>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line1</td>
<td>System: Si/SiO2-NaCl</td>
<td></td>
</tr>
<tr>
<td>Line2</td>
<td>Potential: -1000mV</td>
<td></td>
</tr>
<tr>
<td>Line3</td>
<td>Current: -3.25e-10A</td>
<td></td>
</tr>
<tr>
<td>Line4</td>
<td>Temperature: RT</td>
<td></td>
</tr>
<tr>
<td>Line5</td>
<td>Time: 08:20:15-08:26:31</td>
<td></td>
</tr>
<tr>
<td>Line6</td>
<td>Comment 1: passivated layer</td>
<td></td>
</tr>
<tr>
<td>Line7</td>
<td>Comment 2: 2cm*cm</td>
<td></td>
</tr>
<tr>
<td>Line8</td>
<td>Comment 3: 1mol NaCl in H2O</td>
<td></td>
</tr>
<tr>
<td>Line9</td>
<td>Comment 4:</td>
<td></td>
</tr>
<tr>
<td>Line10</td>
<td>Comment 5:</td>
<td></td>
</tr>
</tbody>
</table>

**CV/IE data**

<table>
<thead>
<tr>
<th>Line</th>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line1</td>
<td>System: Si/SiO2-NaCl</td>
<td></td>
</tr>
<tr>
<td>Line2</td>
<td>Temperature: 45C</td>
<td></td>
</tr>
<tr>
<td>Line3</td>
<td>Time: 15:20:15-15:30:15</td>
<td></td>
</tr>
<tr>
<td>Line4</td>
<td>Scan rate: 10mV/s</td>
<td></td>
</tr>
<tr>
<td>Line5</td>
<td>Comment1: passivated layer</td>
<td></td>
</tr>
<tr>
<td>Line6</td>
<td>Comment2: 2cm²</td>
<td></td>
</tr>
<tr>
<td>Line7</td>
<td>Comment3: 1mol NaCl in H2O</td>
<td></td>
</tr>
<tr>
<td>Line8</td>
<td>Comment4</td>
<td></td>
</tr>
<tr>
<td>Line9</td>
<td>Comment5</td>
<td></td>
</tr>
<tr>
<td>Line10</td>
<td>Comment6</td>
<td></td>
</tr>
</tbody>
</table>
9.4 Inserting Text Into a Graphical Document

Via the ANA_REM_*** macros the file description block is copied to the array named text$. You may place one or more lines of the description (= indices of the array) e.g. into a graphic or a form in CAD. To do that use the macro:

\[ \text{CAD\_TEXT}(x,y,count,arr$()) \]

\( x \) : x-position (mm) of the lower left edge of the string  
\( y \) : y-position (mm) of the lower left edge of the string  
\( count \) : number of lines (indices) to be plotted  
\( arr$() \) : string array (e.g. the SCRIPT system array text$()).

The macro plots a number (\( count \)) of lines (indices) of the string array \( arr$() \) to an existing CAD graphic at the position \( x, y \) relative to the lower left corner of the graphics. \( x \) and \( y \) are to be input in the unit of [pixel] (= [mm] in print-out).

We recommend the use of the pre-defined SCRIPT system array text$(99)$ which is able to store 100 ASCII strings.

**Example 1**

\[ \text{CAD\_TEXT}(110,220,5,text$()) \]

Plots the first five lines (indices 0 to 4) of the string array \( text$() \) into the recent document at the position \( x=110\text{mm} \) and \( y=220\text{mm} \)

**Example 2**

\[ \text{ANA\_REM\_EIS}(1,1) \]
\[ \text{CAD\_TEXT}(30,200,10,text$()) \]
\[ \text{ANA\_REM\_EIS}(1,2) \]
\[ \text{CAD\_TEXT}(60,200,10,text$()) \]

This sequence first copies the file description's titles of the file loaded last to the text array text$. The first 10 elements of text$() are plotted to the graphic starting at position \( x=30\text{mm} \), \( y=200\text{mm} \). Finally, the file description's comments are copied to text$() and are plotted to the right of the corresponding titles.

9.5 Text Attributes

Text attributes are defined with the macro command

\[ \text{CAD\_TTYP}(size,font,direction,color,justx,justy,vmotion) \]

\( size \) : Defines the height of a font. It is roughly calculated by the formula \( \text{height} \approx Size \times 2\text{mm} \)  
For the font 7-Segment the size is limited to the values 0.5, 1 and 2, standard is 2.
font: You have access to a couple of Thales specific fonts as well as to a couple of Windows fonts. They are coded by numbers.

The Thales fonts are:
- 0 CG-Times
- 1 Universe
- 2 Courier
- 3 Symbol / Greek character set
- 4 Helvetica
- 5 Times Roman
- 6 7-Segment

The Windows® fonts are:
- 32 Times New Roman
- 33 Universe
- 34 Courier
- 35 Symbol / Greek character set
- 36 Arial
- 37 Times Roman

In order to select a font attribute add the following values to the font no.:

<table>
<thead>
<tr>
<th>Add value</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 / 8</td>
<td>German / US ASCII</td>
</tr>
<tr>
<td>0 / 32</td>
<td>normal / italic</td>
</tr>
<tr>
<td>0 / 64</td>
<td>normal / fat</td>
</tr>
<tr>
<td>0 / 128</td>
<td>normal / underline</td>
</tr>
</tbody>
</table>

If you intend to print or export the data to a HPGL device, it is recommended to use Universe or CGTimes.

If you intend to print to the Windows® printer or export via EMF meta-files, it is recommended to use the Windows® fonts. The best position and scale correspondence can be achieved with Windows® Arial (Code 20 or 28).

direction: The direction of the text is defined using the following values:

- Direction
  - Direction=0
  - Direction=1
  - Direction=2
  - Direction=3

color: This parameter sets the color of the text. The color coding is as follows:
<table>
<thead>
<tr>
<th>Color no.</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>1</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>yellow</td>
</tr>
<tr>
<td>4</td>
<td>dark blue</td>
</tr>
<tr>
<td>5</td>
<td>lilac</td>
</tr>
<tr>
<td>6</td>
<td>light blue</td>
</tr>
<tr>
<td>7</td>
<td>gray</td>
</tr>
<tr>
<td>8</td>
<td>cyan</td>
</tr>
<tr>
<td>9</td>
<td>dark brown</td>
</tr>
<tr>
<td>10</td>
<td>light brown</td>
</tr>
<tr>
<td>11</td>
<td>orange</td>
</tr>
<tr>
<td>12</td>
<td>pink</td>
</tr>
<tr>
<td>13</td>
<td>violet</td>
</tr>
<tr>
<td>14</td>
<td>middle blue</td>
</tr>
<tr>
<td>15</td>
<td>white</td>
</tr>
</tbody>
</table>

*justx, justy*: This parameter is used to adjust the text as follows:

**Alignment**

<table>
<thead>
<tr>
<th>Justx=0</th>
<th>Justx=1</th>
<th>Justx=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Justy=0</th>
<th>Justy=1</th>
<th>Justy=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*vmotion*: The vertical spacing between two lines of text is

\[ D \approx \text{vmotion} \times 1.42 \times \text{height} \]

If a parameter *vmotion* is omitted it is set to 1 by default.
10. Message Boxes

MESSAGE boxes may be implemented in a SCRIPT in order to inform the user about an action to be done, etc. This can be a warning, an error message or a request to the user to react in a certain way.

10.1 Resetting the MESSAGE-Function

Prior to the use of a Message Box function it has to be initialized using the macro:

```
MESSAGE(0)
```
initializes the Message Box macro routine

10.2 Message Box Text

A Message Box is able to display up to 20 lines of text with the following structure:

```
message caption line
empty line
line 1
line 2
........
........
line 10
```

The different texts will be passed to the output routine via the function

```
MESSAGES$(text$)
```
send a text line to the currently opened Message Box

`text$` is a string keeping the text string to be sent.

**Example**

```
t3$="will put out"
t4$="4 lines of text"
MESSAGES$("Message test")
MESSAGES$("This message")
MESSAGES$(t3$)
MESSAGES$(t4$)
```
10.3 Display Message Box

**MESSAGE**(wait, x, y)

The predefined message is displayed by calling the MESSAGE macro. Depending on the parameters sent with the MESSAGE command, different actions can be initiated. *wait* defines the time the Message Box is displayed whereas *x* and *y* define the position of the box on the screen.

**MESSAGE**(wait)

This macro displays a message on the screen for *wait* milliseconds and then automatically removes it. Example: **MESSAGE**(5000) displays the Message Box with the defined text lines, beeps and waits for 5 seconds before the box is removed and the program is continued.

Optionally you may define the position of the message box on the screen. The position is restricted to the following coordinate range:

0<=x=79
0<=y<=31

- *x*=0 or *y*=0 will select a random position of the box on the screen
- *x*=1 or *y*=1 will select the center position of the box on the screen
- *x*=2,3,. & *y*=2,3,…sets the position relative to lower left corner of the screen.

The size of the message box will automatically be set depending on the number of lines and the length of the longest text line:

X-width \( XW = \max(\text{length(message-texts)}) + 12 \)
Y-height \( YW = \text{count of messages} + 2 \)

If the maximum length of the message(s) is higher than the remaining width 80-\(XW\) the defined coordinates are omitted and the message box is displayed centered.
If the height of the message window (= \text{count of lines} + 2) is higher than the remaining height 32-\(YW\) the defined coordinates are omitted and the message box will be displayed centered.
10.4 Message Box Specials

**MESSAGE(-2)** will prompt the message and ring the bell once. The program is forced to wait for the user pressing a key to remove the message and to continue program execution.

**MESSAGE(-1)**

The *Message Box* is displayed while the program is continued. The box has to be closed by the script program using the andiBASIC command `plotpull`. Each individual message `MESSAGE(-1)` will open its own window on the screen and will occupy graphics memory. If too many *Message Boxes* are open you may encounter problems with the graphic memory running short. When a *Message Box* is needed no longer, it should be closed with a `plotpull`.

10.5 Progress Indicator Functions

To display the actual state of a time consuming operation (like file copying), script offers two functions to indicate the progress of the operation:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WAITDISP</strong> (x, y, &quot;string-expression&quot;)</td>
<td>displays a window at the screen position (x) and (y) with the header &quot;string-expression&quot; and a progress bar, denoted with “0” and “100” percent at the left and right margin respectively.</td>
</tr>
<tr>
<td><strong>WAITBAR</strong> (pr)</td>
<td>Updates the progress-bar. The value ‘(pr)’ must be in the range between 0 and 1 and must be updated increasingly</td>
</tr>
</tbody>
</table>
11. File Handling

11.1 General

From October 2004 on, all ECW systems are shipped without an internal hard drive. They use the PC hard drive as mass storage device for programs and data. Older ECW systems have got an internal hard drive where the Thales software as well as the data was stored. SCRIPT has access to both, the ECW hard drive and the PC hard drive.

The ECW file handling system is slightly different from the WINDOWS® file handling. The restrictions of the ECW file handling system are:

- the maximum number of files within a sub-directory is 508
- maximum file names length is 14 characters
- maximum subdirectory-levels is 8
- the upper most directory is called "maindir" and stores all programs and parameter files to run the THALES®-software.

The THALES file manager is case sensitive regarding the filenames. As described above, Thales adds an extension consisting “0” (lower case letter) and “1” (upper case letter) to achieve compatibility.

Example for a file name using upper and lower case letters stored from Thales and browsed with the Windows Explorer:
“ThalesName.1000001000.bin”
To avoid the numerical extensions we recommend to only use lower case letters for file names.

11.2 Directories

A directory is a list of the file within a folder. For some purpose it is useful to have access to these names. A directory is read and its elements are transferred to the SCRIPT system arrays Name$( ), Date$( ) and Len ( ). The maximum number of elements the DIR_***() functions can handle is 512. If you are dealing with directories which have got more entries please use the search-string option for reduce the amount of elements.

These functions offer a wide variety of applications. Directories can be checked for existing file names and files may be selected which shall be evaluated in an automatic routine.

\[ \text{DIR}_***( Device,Path$, Search$, Mode ) \]

The DIR_***() macro may handle all files or only the files created by a specific method, which is specified in the place of the ***.
The available macros are:

- **DIR_ALL**$( ) Checks for all types of files
- **DIR_EXP**$( ) Check for the export (MS-DOS) directory
- **DIR_CAD**$( ) Checks for 'cad'-files
- **DIR_TEXT**$( ) Checks for 'text'-files
- **DIR_EIS**$( ) Checks for impedance measurement data files
- **DIR_MOD**$( ) Checks for model files
- **DIR_CV**$( ) Checks for cyclic voltammogram data files
- **DIR_IE**$( ) Checks for current-potential-curve files
- **DIR_POL**$( ) Checks for polarization curve data files

**Device** specifies the logical drive where the example file is stored. Usually Device is always 65. In elder system configurations Device may have other values.

The following entries for ‘Device’ are defined:

<table>
<thead>
<tr>
<th>Device no.</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device = 1</td>
<td>ECW floppy disc</td>
</tr>
<tr>
<td>Device = 2</td>
<td>ECW internal hard disc</td>
</tr>
<tr>
<td>Device = 65</td>
<td>PC file system</td>
</tr>
</tbody>
</table>

If **Device=65** (PC file system) is selected the partition of the PC hard drive has to be defined with the Path$ string.

**Path$** string defining the data path of the file to be loaded
Subdirectory levels are separated by a ‘\’ (backslash).

**Search$** Search string for the selection of specific directory entries. If all files are to be listed use an asterisk (“*”) as search string.

**Mode** The Mode parameter enables the user to select directory entries from the list. This parameter may be omitted if the routines shall run with the default settings (no file selection)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>file selection</td>
</tr>
<tr>
<td>0</td>
<td>no file selection</td>
</tr>
</tbody>
</table>
The **DIR_***$()** functions will return the following data to SCRIPT:

- **Filecount**  : number of files found
- **Name$()**    : the filenames of the files found
- **Date$()**    : date of creation of the files found
- **Len ()**     : length of the files found

The arrays are dimensioned for 512 elements. Longer directories cannot be processed.

The following examples will show application of the **DIR_***$()**-functions:

NOTE! **Filecount** will return the actual count of files that have been found. These file data are transferred to the arrays starting at index 0 up to index **Filecount-1**. Each use of the **DIR_***$()**-function will overwrite the existing data stored in the arrays **Name$()**, **Date$()** and **Len ()**.

**Example 1: Load data and fit**

```plaintext
SCRIPT1
SRC_PATH$="examples"
DES_PATH$="examples"
DIR_EIS$(2,SRC_PATH$,"pser*")
LOOPEND%=Filecount-1
CLRSIM
ANA_OPEN_MOD(2,"script","pserie")
    for FILE%=0 to LOOPEND%
        ANA_OPEN_EIS(2,SRC_PATH$,Name$(FILE%))
        FITEIS(0.1,100000,20)
    next FILE%
next FILE%
CLRSIM
for FILE%=0 to LOOPEND%
    ANA_SAVE_MOD(2,DES_PATH$,"@"+Name$(FILE%))
next FILE%
CAD_OPEN(2,"script","scr_for1")
ANA_PLOT_MOD(LOOPEND%+1)
ANA_HPPLOT(20,40)
SCRIPT_END
```

The example above loads 6 spectra named 'pserieXY' from the 'examples' subdirectory and fits the measured data with the 'pser' model. The fitted data will be saved as models named 'pserieXY'. Finally, all fitted data will be loaded and a graph is drawn into the form named 'scr_for1'. The result of this procedure will be resident in the CAD program after the sequence has been finished. A **Mode** setting of "1" in the directory function **DIR_EIS$(2,SRC_PATH$,"pser*",1)** will enable file selection.

**Example 2: Checking hard disk entries**

```plaintext
SCRIPT1
SRC_PATH$="examples"
DES_PATH$="examples"
DIR_ALL$(2,DES_PATH$,"*")
FREE%=508-Filecount
DIR_EIS$(2,SRC_PATH$,"pser*", 0)
```

The `Dir_ALL$()` function returns the number of files in the specified directory, whereas the `Dir_EIS$()` function searches for files matching the specified pattern.
The example above will give hints on how to avoid errors. The free entries of the destination directory will be checked by the directory function. If the condition (SRC%>FREE%) will become true the program will branch to the label S1_ERR1:: and prompt an error message. Otherwise the program will continue as shown in 8.3.0. After the plot the sequence will branch to the label S1_END::, and end at the usual ending descriptor SCRIPT_END.

Example 3: Fitting a complete subdirectory

```plaintext
SCRIPT1
DAT_PATH$="examples" 'Mark start of script'
MOD_PATH$="examples" 'Path of measure data'
DES_PATH$="examples" 'Path of model file'
DIR_ALL$(2,DES_PATH$,"*") 'Destination path'
FREE%=508-Filecount 'How many files in des-dir'
FILE%=Filecount 'Free entries'
DIR_EIS$(2,SRC_PATH$,"*") 'Search for all EIS-data'
BRANCH%=0 'Store Filecount'
if (FILES%>FREE%) trueif 'More files selected than'
   BRANCH%=1 'free entries? If YES'
endif 'continue at S1_ERR1'
on BRANCH% goto S1_ERR1 LOOPEND%=FILEcount-1 'Initialize SIM'
CLRSIM
ANA_OPEN_MOD(2,"script","pserie") 'Load model: scr_demo to SIM'
for FILE%=0 to LOOPEND% 'Start LOOP'
   ANA_OPEN_EIS(2,SRC_PATH$,Name$(FILE%)) 'Load spectrum'
   FITEIS(0.1,100000,20) 'Fit spectrum'
   ANA_SAVE_MOD(2,DES_PATH$,"@"+Name$(FILE%)) 'Save fit and overwrite'
next FILE% 'Do next spectrum'
CLRSIM 'Init SIM'
for FILE%=0 to LOOPEND% 'Start LOOP'
   ANA_OPEN_MOD(2,SRC_PATH$,Name$(FILE%)) 'Load model'
next FILE% 'Do next model'
CAD_OPEN(2,"script","scr_for1") 'Load formulary'
ANA_PLOT_MOD(LOOPEND%+1) 'Plot models'
ANA_HPLLOT(20,40) 'Create cad plot'
goto S1_END S1_ERR1::
MESSAGE(0) 'Reset Message function'
MESSAGE$("disc error!") 'Define Message$1'
MESSAGE$("too many files selected") 'Define Message$2'
MESSAGE$("too less free entries") 'Define Message$3'
MESSAGE(-1) 'Output of message, wait for key strike'
S1_END::
SCRIPT_END 'End of Script1'
```
All measurements being stored within the specified subdirectory DAT_PATH$ will be fitted. The fitted data then will be stored to a free subdirectory defined in DES_PATH$.
12. Communication- and Dialog boxes

If you have to control a procedure manually, you have to communicate with the system. In SCRIPT you can do this using input boxes. They allow to define parameters which you input via the keyboard. The values may be of string (e.g. file name, comment, etc.) or numerical type (e.g. amplitude, current, etc.). In some cases it is more convenient to have check boxes to select from given choices. For all these purposes, SCRIPT offers several procedures to input data at runtime.

12.1 Input Box

SCRIPT offers the INBOX-procedure to insert text- or numerical data into an input-box at runtime. In detail, the following SCRIPT-procedures are involved:

**INBOX(0)** Initialization of the input-box array iS()

**INBOX$(Info$, Default$, Type, Min, Max)*** Setup of the box parameters

**INBOX(Icount, Tcount[,x,y])** Displays the input-box

**iS()** String-array for the input-strings

The iS() array has got a dimension of 20 elements. Thus, a maximum of 20 inputs are allowed for one box. The input strings iS() are arranged in increasing order starting with the index “1”, i.e. the first input string is available in iS(1), the second in iS(2), and so on.

Prior to the use of an input box the corresponding variables must be initialized. This is done with the **INBOX(0)** macro. Now the parameters of the input box such as box comment string and input parameter range has to be defined by the **INBOX$(..)$** macro. The following parameters may be defined (parameters in [ ] are optional):

- **Info$** defines an info text which is placed directly before the input
- **Default$** default input string which can be confirmed or edited by the user
- **Type** specifies type and format of the input. Its value is composed of the following elements:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Explanation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>(LSB)</td>
<td>String or numerical input</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Sign-handling</td>
<td>0: signed numerical input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(for numerical input only)</td>
<td>1: calculate absolute value</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>1: calculation for numerical input</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>Type of variable (for numerical input only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: floating-point</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: integer</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>Forces a different value compared to the previous line, for instance for a set of starting and ending values.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>Special format (for numerical input only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: numerical format</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: time format (hh:mm:ss)</td>
<td></td>
</tr>
<tr>
<td>7 (MSB)</td>
<td>128</td>
<td>Enable limits</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: use min, max as range limits</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: do not use range limits (min, max)</td>
<td></td>
</tr>
</tbody>
</table>

### Examples

<table>
<thead>
<tr>
<th>Value of typeflag</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>1: numerical input</td>
</tr>
<tr>
<td></td>
<td>16: integer values</td>
</tr>
<tr>
<td>129</td>
<td>1: numerical input</td>
</tr>
<tr>
<td></td>
<td>: (Bit 4 = off =&gt; floating point variable)</td>
</tr>
<tr>
<td></td>
<td>128: disable check of limits</td>
</tr>
<tr>
<td>0</td>
<td>input of text and check of length of the text</td>
</tr>
</tbody>
</table>

**Min** sets minimum range limit (for numerical input referred to the numerical value, for a string literal input referred to the count of characters)

**Max** sets maximum range limit (for numerical input referred to the numerical value, for a string literal input referred to the count of characters)

Now you are ready to open the input box using the macro `INBOX()`. Its parameters have the following meanings:

- **Icount** left hand side display width for the descriptor strings `Info$` (in count of characters)
- **Tcount** right hand side display width for the default strings `Default$` (in count of characters)
- **x,y** defines the lower left corner of the INBOX-window. One coordinate step has the dimension of 8x16 pixel. Example: x=10 and y=10 means 80 pixels to the right and 160 pixel to the top.

#### Example

```script
SCRIPT
SC_STRING$="any string"
INBOX(0) 'Initialize Inbox'
INBOX$("Demo for inbox") 'Caption of the Inbox-header'
INBOX$("String",SC_STRING$,0,3,40) 'Info,Default;0=Type is string, '
                   '3=minimum, 40=max. length of input'
INBOX$("Real",fnSTR$(SC_REAL),1,-1e39,1e39) 'input-type = floating point, '
                   'minimum and maximum value of input'
INBOX$("Integer",fnSTR$(SC_INT%),17,-10,10) 'type=integer,minval,maxval'
                   'minimum and maximum value of input'
INBOX(10,30,25,50) 'call inbox. Parameters are: '
```
Besides the `INBOX()` macro, the following `andiBASIC`-routines and variables may be useful for the handling of inputs:

**FL%** System variable, containing the status of the input-box
- 0: OK, user confirmed data (RETURN key)
- 1: User break (close button clicked or escape key pressed)
- 2: OK button was clicked (see below)

**fnSTR$(val)** function converting a numerical value into a string. `val` : numerical expression. It corresponds to the following `andiBASIC` function `str$(())`, except from suppressing a leading space for positive signed numbers.

**str$(val)** `andiBASIC` function converting a numerical value into a string

**val(string$)** `andiBASIC` command converting a string into a numerical value

**fnVAL(string$)** function converting a string containing a formatted numerical value using engineering prefixes into the corresponding numerical value. The following prefixes are recognized:

<table>
<thead>
<tr>
<th>prefix</th>
<th>Name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>atto</td>
<td>$10^{-18}$</td>
</tr>
<tr>
<td>f</td>
<td>femto</td>
<td>$10^{-15}$</td>
</tr>
<tr>
<td>p</td>
<td>pico</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>n</td>
<td>nano</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>u</td>
<td>micro</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>m</td>
<td>milli</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>K</td>
<td>kilo</td>
<td>$10^3$</td>
</tr>
<tr>
<td>M</td>
<td>mega</td>
<td>$10^6$</td>
</tr>
<tr>
<td>G</td>
<td>gig</td>
<td>$10^9$</td>
</tr>
<tr>
<td>T</td>
<td>tera</td>
<td>$10^{12}$</td>
</tr>
</tbody>
</table>

**fnVAL$(val[,n])** function converting a numerical value into a string using the engineering prefixes above. The function was established to format and truncate numerical values to a significant digit count. The typical 16-bit accuracy of the Thales-environment is forced by `n`=0 or by omitting `n`.

`val` : numerical value; `n` : digits used for conversion (optional)
A maximum of 5 digits are converted. If \( n \) is used, the number of digits is \((5-n)\). If \( n \) is omitted 5 digits are converted. Examples (with \( X=+1.277518e-11 \)):

\[
\begin{align*}
\text{fnVAL}(X,-1) & \Rightarrow 12.7752p \\
\text{fnVAL}(X) & \Rightarrow 12.775p \\
\text{fnVAL}(X,0) & \Rightarrow 12.775p \\
\text{fnVAL}(X,1) & \Rightarrow 12.78p \\
\text{fnVAL}(X,2) & \Rightarrow 12.8p \\
\text{fnVAL}(X,3) & \Rightarrow 13p \\
\text{fnVAL}(-X,4) & \Rightarrow -10p \\
\text{fnVAL}(X,4) & \Rightarrow 10p \\
\text{fnVAL}(X,3.5) & \Rightarrow 10p \\
\text{fnVAL}(-X,3.5) & \Rightarrow -10p
\end{align*}
\]

The integer value of \( n \) is utilized to define the significant digit count. Apart from that, the last two example lines demonstrate, how the leading space holder for positive sign may be forced, if a non-integer value is used for \( n \).

### 12.2 Checkbox

In some cases the user has to select from a number of pre-defined options. For example, a script might present a group of check boxes from which the user can select error conditions that produce warning messages. For such purposes, script offers the checkbox-procedure. In detail, the following SCRIPT-procedures are involved:

- **CHECKBOX\( (0) \)**: Initializes the checkbox result array **CHECKBOX\%()**
- **CHECKBOX\$(String\$)**: Defines a heading line for the checkbox window as well as the names of each checkbox.
- **CHECKBOX\( (\text{Count},x,y,\text{Width},[\text{fontcolcode}]) \)**: Displays the checkbox window and enables the checking procedure

- **String\$**: string expression. The first call sets the headline of the checkbox window, subsequent **CHECKBOX\$(\) calls will set the names of subsequent check boxes.
- **Count**: number of checkboxes in the checkbox window.
- **x,y**: defines the lower left corner of the CHECKBOX window. One coordinate step has the dimension of 8x16 pixel. Example: \( x=10 \) and \( y=10 \) means 80 pixels to the right and 160 pixel to the top.
- **Width**: width of the checkbox window.
**fontcolcode**  
Font and color of the strings to be displayed (omitted = default ASCII Arial, black). This parameter is optional. The value contains the font information in the integer part, the color information in the fractional part: \( \text{fontcolcode} = \text{font} + \text{color}/16 \). See chapter 9.5 CAD_TTYP for details of font and color.

**CHECKBOX%()**  
Array which represents the states of all checkboxes defined in a checkbox window. The values may be set to a default setting before calling the checkbox window. After editing and confirming the checkbox window with the \( \checkmark \) button the current (edited) states can be found in this array. After closing the window with the \( \times \) button all changes are rejected and the array stays unchanged.

The states are represented as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Checkbox is not checked</td>
</tr>
<tr>
<td>0</td>
<td>Creates an empty line</td>
</tr>
<tr>
<td>1</td>
<td>Checkbox is checked</td>
</tr>
</tbody>
</table>

To set up a checkbox window proceed as follows:

1. **CHECKBOX(0)** clears the CHECKBOX%() array
2. **CHECKBOX$(\text{String1$})** defines the heading line of the checkbox window
3. **CHECKBOX%(\text{1})** keeps the state of checkbox 1
4. **CHECKBOX%(\text{2})** keeps the state of checkbox 2

**Example**

```script
mat CHECKBOX%() = -1  
' reset the flag array CHECKBOX% to standard values'  
' -1 : the corresponding box is enabled, default is not checked'  
' 0 : the corresponding box is disabled'  
' 1 : the corresponding box is enabled, default is checked'  
CHECKBOX%(1) = 1  
' enable CHECKBOX%(1) and set to checked'  
CHECKBOX%(2) = 0  
' disable CHECKBOX%(2)'  

CHECKBOX(0)  
' Initialize the checkbox function'  
CHECKBOX$("CHECKBOX DEMO"+CR$+"Sub-Heading")  
' Header and optional Sub-heading'  
CHECKBOX$("1: 1st box (checked)")  
' 6 descriptive strings.'  
CHECKBOX$("2: 2nd box (suppressed)")  
' default settings as defined'  
CHECKBOX$("3: 3rd box (not checked)")  
' above'  
CHECKBOX$("4: 4th box (not checked)")  
CHECKBOX$("5: 5th box (not checked)")  
CHECKBOX$("6: 6th box (not checked)")  

CHECKBOX(6,7,4,30,44+15/16)  
' activate the checkbox-procedure'  
' 6 : count of strings/checkboxes'  
' 7,4 : X and Y-position to display'  
' 30 : width of the checkbox'  
' 44 : Font ASCII Arial, 15=color white'  
' => flag array CHECKBOX%() now'  
' contains the actualized box states'  
plotpull  
' deletes the checkbox window'  
' (DON'T FORGET !!)'  
```

**SCRIPT_END**
In the example script above, two *andiBASIC* procedures and a system variable are used. You may learn more about their functions in the *andiBASIC* manual. Here a short description:

<table>
<thead>
<tr>
<th>Procedure/Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mat</strong> array()=a</td>
<td>Fills the <em>array()</em> with the value <em>a</em></td>
</tr>
<tr>
<td><strong>plotpull</strong></td>
<td>In this context, removes the checkbox-window</td>
</tr>
<tr>
<td><strong>CR$</strong></td>
<td>Defines a carriage return character (ASCII 13). DON'T EDIT THIS VARIABLE !!!!</td>
</tr>
</tbody>
</table>

### 12.3 Toggle Switches

In SCRIPT-applications the user sometimes has to select a single setting from a set of not independent options. For example, if you want to print out a graphic, you can select a printer, the clipboard as output device or you can create a file - but you can select only one output device at once. For such situations, SCRIPT offers the toggle group (=TGROUP)-procedure. In detail, the following SCRIPT-procedures are involved:

**TGROUP(0)** Initialization of the toggle-group procedure

**TGROUP$(String$)** Defines headline and names of the toggle entries

**TGROUP(Count,x,y,Width[,fontcolcode])** Opens the toggle-group window

**TGROUP%** Represents activated toggle entry (set default and read result)
String$ string expression. The first call sets the headline of the checkbox window, subsequent TGROUP$() calls define the names of subsequent toggle entries.

Count number of toggle entries

x, y defines the lower left corner of the toggle-group window. One coordinate step has the dimension of 8x16 pixel. Example: x=10 and y=10 means 80 pixels to the right and 160 pixel to the top.

Width width of the toggle-group window

fontcolcode font and color of the strings to be displayed (omitted = default ASCII Arial, black). This parameter is optional. The value contains the font information in the integer part, the color information in the fractional part: fontcolcode = font + color/16. See chapter 9.5 CAD_TTYP for details of font and color.

Note: The first String$ is used as a headline, all subsequent String$ definitions are used as names for the toggle entries in increasing order. The number of the toggle entry selected after the TGROUP window is closed you can find in the SCRIPT system variable TGROUP%. Before opening a TGROUP window you may set TGROUP% in order to define a default selection.

Example

```script
SCRIPT
TGROUP%=1
TGROUP(0)
TGROUP$("TGROUP DEMO"+CR$+"Sub-Heading")
TGROUP$("weather is very fine")
TGROUP$("weather is fine")
TGROUP$("weather is not fine")
TGROUP$("weather will change")
TGROUP(4,7,5,24,44)
plotpull
SCRIPT_END
```

plotpull ' deletes the checkbox window'

' (DON T FORGET ! !) '

' set default index to the first entry'
' in the system-variable TGROUP%'
' initialize togglegroup. '
' Header and optional Sub-heading'
' 4 descriptive strings for 4 items/'
' entries in the toggle-group-window'
' activate the checkbox-procedure'
' 4 : number of strings/items'
' 7,5 : X and Y-position to display'
' 24 : width of the checkbox'
' 44 (optional) typeface, default=Arial'
' ==> : TGROUP% now contains the'
' index of the selected item '
' deletes the checkbox window'

plotpull ' deletes the checkbox window'

' (DON T FORGET ! !) '

12.4 Listbox

If the user wants to choose one or more data entries within a list of data (for instance, select file names for file operations or choose from a range of names), the LISTBOX function is recommended.

**LISTBOX(0)**

Resets the listbox procedure.

**LISTBOX$(WinHead$)**

The first call defines the headline of the Listbox window.

*WinHead*$

String defining the header of the Listbox window.
Example: *WinHead*$=“Listbox no. 1”

**LISTBOX$(ColHead$)**

The second call defines the sub-header(s), displayed in bold letters on top of each column.

*ColHead*$

String defining the headers of the list columns. The headers of the columns are separated by colons (,).
Example: *ColHead*$=“column 1,column 2,column 3”

**LISTBOX$(Line$,Act)**

All subsequent calls of the LISTBOX$(()) macro transfer the contents of each subsequent line.

*Line*$

String defining the content of one line. The columns are separated by colons (,). The number of string sections separated by commas must equal the number of sections defined in *ColHead*$.
Example: *Line*$=“data 1,data 2,data 3”

*Act*

defines whether a Listbox entry is pre-selected (0 = not selected, -1 = selected). If you use the LISTBOX%() elements in that place, you create a listbox with the current settings.

**LISTBOX%()**

*SCRIPT system array* representing the current selections (0 = deselected, -1 = selected).

The LISTBOX%() entries remain unchanged, if the user escapes the Listbox window by clicking on the CLOSE button or by clicking the middle mouse button. Only a click on the check button transfers the edited entries to the array.

If more than one listbox shall be provided within the script, use individual arrays for the results and use the array LISTBOX%() only locally.

In order to open a listbox window use the macro:
LISTBOX(Select, x, y, dx, dy, Fixed, Height, Alignment):

Select 1 = only one entry is selected, all others are deactivated
2 = enables multi-selection

x, y coordinates of the upper left corner of the listbox window on screen. The coordinates are set in pixel!

dx, dy window width and height (pixel)

fixed 0 = true-typeface Arial
1 = fixed typeface ‘Courier New’ for labeling

height count of vertical pixels reserved for each line

alignment 0 = second column right justified (valid only for two columns)
1 = left justified.

Example

SCRIPT

LISTBOX(0) 'initialize'
LISTBOX$("Header") 'task bar header'
LISTBOX$("Left,MiddleL,MiddleR,Right") 'subheader'
' define eleven data lines'
for i=0 to 10
    LISTBOX$("Line"+str$(i)+",Column2,Column3,Column4",LISTBOX%(i))
next
' call LISTBOX, multiselect, truetype, height 12, left justified columns'
LISTBOX(2,200,300,300,200,0,12,1)

SCRIPT_END
12.5 Pop-Up Menus

The menu macros create a pop-up menu where you may make a decision about the further proceeding of the process. It returns the selection in the form of a numerical value which you may use in a branch list in order to jump to the corresponding routine. The programming is very similar to that of a TGROUP.

**MENU(0)**
Initialization of the menu procedure

**MENUS(String$)**
Defines headline and names of the menu entries

**MENU(Count,Active,xpos,ypos,xwidth)**
Executes the menu procedure.

At the character-positions *xpos*, *ypos* a menu with *Count* lines and the character width *xwidth* will pop up. The mouse cursor position is set to the menu point # *Active*.

- **String$** string expression. The first call sets the headline of the menu window, subsequent **MENUS()** calls define the names of subsequent menu point entries.
- **Count** number of menu point entries
- **x,y** defines the lower left corner of the menu-window. One coordinate step has the dimension of 8x16 pixel.
- **xwidth** width of the menu window

The **MENU** macro return the variables *FL%* and *MENU%*:

- **FL%** flag indicating whether the user escaped (no selection) or not
  - *FL%<>0* => user escaped without a selection
  - *FL%=0* => user selected and escaped
- **MENU%** returns the number of the selected menu entry

**Example**

```
MENU(0)  'reset MENU parameters'
MENUS(“X-Data”)  'define MENU entries'
MENUS(“X-Range”)
MENUS(“Y-Data”)
MENUS(“Y-Range”)
MENUS(“Color”)
MENUS(“Symbols”)

MENU(6,GRAPH1%,35,8,25)  'MENU(Count=6,Aktive #,xpos,ypos,xwidth), '
onFL%gotoMUXMENU  'FL%<>1 : user escaped from menu -> no action'
gotoMUXMENU
```

**MUXMENU::end**
13. Graphics

The THALES-software provides all necessary tools to create a company specific graphical environment of the SCRIPT main menu. Graphical files as well as example files can be found within the subdirectory `c:\thales\script` on your PC. The following files should be found in this folder:

**SCRIPT system files**

- `scr_m` main menu
- `scr_ce` help file
- `scr_pe` help file
- `scr_ed` active button `<edit script>`
- `scr_ck` active button `<check script>`
- `scr lk` active button `<link script>`
- `scr_sv` active button `<save program>`

**SCRIPT user files**

- `scr_logo` background including passive buttons & logo
- `scr_b1` active button `<user 1>`
- `scr_form` graphical form
- `scr_for1` graphical form

---

The **SCRIPT system files** are defining the graphical environment of SCRIPT and must not be changed by the user.

An individual appearance of SCRIPT is offered by editing the **SCRIPT user files**. You may add your company's logo, individually styled buttons, graphics, etc. to this group of files.

**SCRIPT demo files**

- `exa830` SCRIPT-example file
- `exa831` SCRIPT-example file
- `exa832` SCRIPT-example file
- `exa91` SCRIPT-example file
- `f pserie` model demo file
- `but_exa1` button's graphical demo1
- `but_exa2` button's graphical demo2
- `but_exa3` button's graphical demo3
- `but_exa4` button's graphical demo4
A software installation or software update will overwrite the SCRIPT graphic files being stored in the directory `c:\thales\script`. To protect your user files, create a new subdirectory and save your graphics to that directory. Finally, use the SCRIPT system variable `SCRIPTPATH$` to assign that directory during the linking procedure.

13.1 The Structure of the Graphical Resources in SCRIPT

The SCRIPT menu consists of an overlay of several graphic elements.

First of all, the main menu `scr_m` is plotted on the screen. Next, the file `scr_logo` is plotted. It defines the appearance of the script desktop. The files `scr_bx` finally are needed to highlight an activated button.

As mentioned above the file `scr_m` must not be edited. The individual appearance will be obtained by `scr_logo`. To create a new logo use the CAD program and edit the file named `scr_logo` in the subdirectory `c:\thales\script`. 
13.2 Buttons

For each of the linked scripts, a graphical file `scr_bx` should exist, which represents the appearance of the active button. The appearance of the passive button should be integrated in the `scr_logo` file. The graphic data of the active button must be plotted at the same position of the passive view in the `scr_logo` and should be of the same size.

To highlight an activated button, you may use inverted frames, background and/or texts in different color, etc.
All graphics `scr_m`, `scr_logo`, `scr_b1`, `scr_b2`, etc. are plotted within a rectangle of the size (width=160mm/height=128mm). The origin is at position (X0=0mm/Y0=0mm) of the CAD program. Due to the digitized screen, the user area is (0,0) to (159.75/127.75). For editing use scale=1.

maximum visible area x = 159.75, y = 127.75

origin x = 0, y = 0
Within SCRIPT, nine script sequences may be defined which usually are represented by nine buttons. The buttons are automatically implemented by the linking procedure.

Please comply with the following restriction when creating buttons:

1. A button should be of square or rectangular shape.
2. The different buttons must not overlap.
13.3 Embedding User-Defined Graphics

To enhance the individual outfit of your SCRIPT main page, you may add your own graphics (user defined drawings). Before you can add a drawing, you must create it in CAD and save it to the folder of your current script (see also SCRIPT_PATH). In order to implement your graphics, two SCRIPT macros are available:

**SCRIPT_PLOT=graph$** Sets the name of the graphic file to be plotted by SCREENPLOT(). More than one file name may be defined by subsequent assignments. They are numbered from “0” on upwards.

*graph$* name of a graphic file (string expression)

**SCREENPLOT(no)** Plots the drawing with the subsequent number *no* defined by SCRIPT_PLOT earlier

*no* number of the graphic file defined by SCRIPT_PLOT

**Example**

```plaintext
SCRIPT_PATH$="script\pmux"  'Defines the path for SCRIPT import-files'
SCRIPT_PLOT$="setoff"    '=> Graphic-file that will be plotted by SCREENPLOT(0)'
SCRIPT_PLOT$="seton"     '=> Graphic-file that will be plotted by SCREENPLOT(1)'

SCRIPT1
  SCREENPLOT(0)  'plots the first graphic on the SCRIPT-screen'
  SCREENPLOT(1)  'plots the second graphic on the SCRIPT-screen'
SCRIPT_END
```

**Note**

- The order of the SCRIPT_PLOT-definitions determines the number which has to been used by SCREENPLOT(nr), i. e. the first defined drawing gets the number=0, the second the number=1 …

- The position where the drawing is plotted at runtime is exactly that where you have designed it using CAD. If you want to place a drawing, defined by SCRIPT_PLOT at a different site, use the plot“sk”,Xpos,Ypos-command to set the plot origin to the correspondent position.
14. Creation and Export of Graphics Created by SCRIPT

It may be useful to create, plot or export graphs of measured or simulated data. For this purpose, SCRIPT offers some powerful tools. Moreover, the SCRIPT allows a mathematical treatment of (e.g. noisy) data, such as smoothing, regression and splines and others. Although a mathematical treatment of measured data seems to be quite complex in detail, in SCRIPT it takes only a few commands to create custom-treated diagrams.

For the handling of user defined graphics, the following SCRIPT macros are available:

- **PLOTFRAME()** Defines the area and background parameters of a graph
- **PLOTTYPE()** Defines point- and smoothing parameters
- **PLOTXYZ()** Plots data- and calculated points of a graph
- **PLOTAXIS()** Plots the axis of a graph
- **PLOTLABEL()** Plots the label of an axis
- **PLOTTEXT()** Plots a text string at certain position in a document
- **PLOTLEGEND()** Plots a graph curves legend box
- **PLOTLEGENDS()** Defines graph curve legend lines
- **DOC_OUT()** Exports a CAD-document
- **XYZ()** SCRIPT system array containing the data to be plotted; the dimension of the array is: XYZ(9999,2)

Detailed description of the graph macros:

**PLOTFRAME(x0,y0,dx,dy,color,thickn,expflg,framena,framcol,framflg)**

- Defines the active area (frame) of a diagram. Simultaneously, the 4 (5) lines which surround the active area, define the sites of the diagram axes. Please note that ‘distance values’ are given in pixels for the output on the screen, respectively in millimeters for the export to the CAD program.

  - **x0** coordinate of the start of the X-axis (lower left corner)
  - **y0** coordinate of the start of the Y-axis (lower left corner)
  - **dx** length of the X-axis
  - **dy** length of the Y-axis (upward direction)
  - **color** bounding frame color (possible values: 0 to 15)
thickn  thickness of the bounding frame in Pixel

expflg  0 = output to screen (default); 1= export to CAD

framena for a value > 0, framena pixels are reserved (extra-frame) outside the area defined by x0, y0, dx, dy. (especially useful for plotting axis-labels outside the frame)

framcol background color of the extra-frame (provided framena > 0)

framflg flag for deleting the background (active area and extra-frame)

PLOTTYPE(lines,symbols,calccnt,smoothw,splinep,splinexy,errtype)

Defines the line- and dot-type of the plot. In addition, parameters for smoothing and/or regression are set.

lines Defines the type and the thickness of the line which connects the data-points (see PLOTXYZ below). The default value is: lines = 0.5. Formally, lines is a floating point number where the integer part defines the line-type and the fractional part defines the line-width.

lines = linetype.linewidth

For linetype, the following types are available:

<table>
<thead>
<tr>
<th>linetype</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>solid line</td>
</tr>
<tr>
<td>1</td>
<td>narrow dotted line</td>
</tr>
<tr>
<td>2</td>
<td>dashed line</td>
</tr>
<tr>
<td>3</td>
<td>dashed-dotted line</td>
</tr>
<tr>
<td>4</td>
<td>wide dotted line</td>
</tr>
</tbody>
</table>

linewidth is a relative measure for the thickness of the line (in pixels). A value of 0.5 corresponds to an absolute linewidth of two pixels, whereas a value of 0.99 corresponds to a width of about 8 pixels (the higher the value the thicker the line, but not linear).

symbols Defines the symbol shape and the symbol size of the data points to be plotted (see PLOTXYZ() below). The default value is: symbols = 0.5. Formally, symbols is a floating point number where the ‘integer part’ defines the shape (i. e. symbol type) and the ‘fractional part’ defines the symbol size according to

symbols= symboltype,symbolsize
        0 <= symboltype <= 15
        0 < symbolsize < 1
The coding of *symboltype* is as follows:

<table>
<thead>
<tr>
<th>symboltype</th>
<th>shape</th>
<th>symboltype</th>
<th>shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Hollow) circle</td>
<td>8</td>
<td>down triangle</td>
</tr>
<tr>
<td>1</td>
<td>(Hollow) square</td>
<td>9</td>
<td>Rhombus</td>
</tr>
<tr>
<td>2</td>
<td>(Hollow) up triangle</td>
<td>10</td>
<td>filled circle</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>11</td>
<td>filled square</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>12</td>
<td>filled up triangle</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>13</td>
<td>filled slanted square</td>
</tr>
<tr>
<td>6</td>
<td>(Hollow) slanted square</td>
<td>14</td>
<td>filled down triangle</td>
</tr>
<tr>
<td>7</td>
<td>(not used)</td>
<td>15</td>
<td>filled rhombus</td>
</tr>
</tbody>
</table>

As explained above for the parameter *linewidth*, the fractional part *symbolsize* is a relative measure for the size of the plotted shape. For details, please refer to the example at the end of this chapter.

If you want to process the original data set and add the processed data to your graph, the following parameters define some important settings for this operation. If you do not want so, set these parameters to their default values. For processing details, refer to the `PLOTXYZ()` procedure.

- **calccnt**: Defines the number of data points to be generated by the processing operation, with $16 \leq \text{calccnt} \leq 1025$
  
  The default is $\text{calccnt} = 65$.

- **smoothw**: If data smoothing is selected, *smoothw* defines the width (the number of adjacent datapoints with respect to the middle point of interest) of the gaussian-shaped ‘smoothing window’ as a fraction of the total extent of the graph (total extent = 1), with $0.05 < \text{smoothw} < 1$
  
  The default is $\text{smoothw} = 0.1$ (= 10% of the total extent of the graph).

- **order**: Defines the order of the polynomial for smoothing as well as the order of an $n$'th order polynomial regression, if one of these operations is selected. $1 \leq \text{order} \leq 9$
  
  The default is $\text{order} = 2$.

- **splinep**: If splining is selected, *splinep* selects the type for the interpolating (cubic) spline.
  
  $\text{splinep} = 0$ selects the nonperiodic spline-type
  
  $\text{splinep} = 1$ selects the periodic spline-type
  
  The default is $\text{splinep} = 0$.

- **splinexy**: If splining is selected, *splinexy* defines which part of the data is splined.
  
  $\text{splinexy} = 1$ => x-values are splined
  
  $\text{splinexy} = 2$ => y-values are splined
  
  $\text{splinexy} = 3$ => both, x- and y-values are splined
  
  Default is $\text{splinexy} = 1$. 
errtype

If the error-display mode in PLOTXYZ is activated, this parameter defines the type of symbols that shall be used. The following values are possible:

<table>
<thead>
<tr>
<th>errtype</th>
<th>shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vertical bar (default)</td>
</tr>
<tr>
<td>2</td>
<td>(Hollow) square</td>
</tr>
<tr>
<td>3</td>
<td>Filled square</td>
</tr>
</tbody>
</table>

PLOTXYZ

Plots a curve from the data of the SCRIPT system array XYZ(). You are free to define start and end index so that sections of the array can be plotted too.

Data must be filled into the rows of XYZ() in ascending order as XYZ(row order index, column meaning index). The four columns are reserved as:

<table>
<thead>
<tr>
<th>index</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X data</td>
</tr>
<tr>
<td>1</td>
<td>Y data</td>
</tr>
<tr>
<td>2</td>
<td>Z data</td>
</tr>
<tr>
<td>3</td>
<td>Error data</td>
</tr>
</tbody>
</table>

sindx

start index referring to YXZ()

eindx

end index referring to YXZ()

curvtyp

the following curve and symbol types are available:

<table>
<thead>
<tr>
<th>curvtyp</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Original data are plotted using the symbols defined in PLOTTYP</td>
</tr>
<tr>
<td>1</td>
<td>Original data points are connected using the linetype defined in PLOTTYP</td>
</tr>
<tr>
<td>2</td>
<td>Combination of 0 and 1</td>
</tr>
<tr>
<td>4</td>
<td>Smoothed curve (line) and original data (symbols) are displayed</td>
</tr>
<tr>
<td>5</td>
<td>Only smoothed curve is displayed</td>
</tr>
<tr>
<td>8</td>
<td>Regression curve (line) and original data (symbols) are displayed</td>
</tr>
<tr>
<td>9</td>
<td>Only regression curve is displayed</td>
</tr>
<tr>
<td>12</td>
<td>Splined curve (line) and original data (symbols) are displayed</td>
</tr>
<tr>
<td>13</td>
<td>Only splined curve is displayed</td>
</tr>
<tr>
<td>16</td>
<td>Error symbol display: differential mode, X-direction</td>
</tr>
<tr>
<td>17</td>
<td>Error symbol display: differential mode, Y-direction</td>
</tr>
<tr>
<td>18</td>
<td>Error symbol display: differential mode, Z-direction</td>
</tr>
<tr>
<td>32</td>
<td>Error symbol display: weighting mode, all directions</td>
</tr>
</tbody>
</table>

The two error-display modes differ with respect to the interpretation of the error information in the curve data respectively error data column: in differential mode, the values from the curve column together with the error data column directly define start- and endpoint of the error bar. Selectively the X-, Y-, or the Z-data can be affected.

In weighting mode, the curve data define the center value, and the error data the significance. A significance of one means infinitely small, a significance of zero means infinitely high absolute uncertainty of X- as well as Y- [as well as Z-] data. This mode is
appropriate to display the uncertainty of a two- or three-dimensional vector when a common uncertainty exists, like it is the case for measured impedance values displayed in a Nyquist diagram.

color
color of the curve and the point symbols (possible values: 0 to 15)

logf
coloring type: linear or logarithmic

<table>
<thead>
<tr>
<th>logf</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>linear scale for X-and Y-axis</td>
</tr>
<tr>
<td>1</td>
<td>logarithmic scale for X-axis, linear scale for Y-axis</td>
</tr>
<tr>
<td>2</td>
<td>linear scale for X-axis, logarithmic scale for Y-axis</td>
</tr>
<tr>
<td>3</td>
<td>logarithmic scale for X- and Y-axis</td>
</tr>
</tbody>
</table>

libf
flag to determine or exclude gaps between active curve area and frame in case of auto scaling active (see below).

<table>
<thead>
<tr>
<th>libf</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>In X- as well as in Y direction the curve touches the frame</td>
</tr>
<tr>
<td>1</td>
<td>In X-direction a 5% gap is reserved between curve limits and frame</td>
</tr>
<tr>
<td>2</td>
<td>In Y-direction a 5% gap is reserved between curve limits and frame</td>
</tr>
<tr>
<td>3</td>
<td>In X- as well as in Y direction a 5% gap is reserved between curve limits and frame</td>
</tr>
</tbody>
</table>

xmin, xmax, ymin, ymax
xmin and xmax define the lowest and the highest values of the x-scale.
ymin and ymax have the same function for the y-scale.
Setting xmin=0 & xmax=0 forces auto scaling in x-direction: x-scaling endpoints are calculated from the data set.
Setting ymin=0 & ymax=0 forces auto scaling in y-direction: y-scaling endpoints are calculated from the data set. libf function is enables in case of auto scaling.

The function PLOTXYZ() may also be used to display real time data in a convenient way. Setting sindx equal to eindx will let the curve degenerate to a single point. An effective way for both actual real time data display as well as later display of data sets already acquired is the following: Store the actual real time result in ascending order in the array XYZ() and apply PLOTXYZ() for every data point arriving with both sindx as well as eindx pointing to the actual value. Please recognize, that xmin, xmax, ymin, ymax values have to be set accordingly (no auto-scaling possible) and only curvtyp values between 0 and 2 are available in this mode. After completion of a record, data are already present in XYZ() for further processing.

PLOTAXIS(axindex,axcolor[,grid,gridcol,textsize,texttype,mode,min,max])

Plots the axis scaling and grid given by the parameter axindex and scales the axis.

axindex defines which axis is to be scaled. The following table lists the assignment of axindex to the corresponding axis.
If you intend to plot the scaling outside the frame (extra-frame, see the PLOTFRAME-command), you have to add a value of 8 to the value of \textit{axindex}. Otherwise the scaling is plotted inside the frame.

<table>
<thead>
<tr>
<th>axindex</th>
<th>axis</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bottom x-axis</td>
<td>Scaling inside frame</td>
</tr>
<tr>
<td>1</td>
<td>left y-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>2</td>
<td>top x-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>3</td>
<td>right y-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>4</td>
<td>z-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>8</td>
<td>bottom x-axis</td>
<td>Scaling outside frame</td>
</tr>
<tr>
<td>9</td>
<td>left y-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>10</td>
<td>top x-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>11</td>
<td>right y-axis</td>
<td>&quot;</td>
</tr>
<tr>
<td>12</td>
<td>z-axis</td>
<td>&quot;</td>
</tr>
</tbody>
</table>

\textit{axcolor} color of axis labeling and ticks (possible values: 0 to 15)

The following optional parameters are substituted by defaults, if omitted.

\textit{grid} \hspace{1cm} 0 = no grid (default), 1 = grid enabled

\textit{gridcol} grid color if grid is enabled (possible values: 0 (default) to 15)

\textit{textsize} size for the axis-labeling according to the information under CAD\_TTYP. The typical value is \textit{textsize} = 1.5 (to 2.5). The default is \textit{textsize} = 2.

\textit{texttype} font for the axis-labeling according to the information under CAD\_TTYP. The default value is \textit{texttype} = 44 for standard Windows ASCII Arial font.

\textit{mode} defines user- (\textit{mode}=0), linear- (\textit{mode}=1) or logarithmic scaling (\textit{mode}=2). Default is the corresponding curve plotting mode used in a previously executed \texttt{PLOTXYZ\texttt{}}-command.

\textit{min} Starting value for the axis, referred to the logical start of the plot (left hand end for X-, lower end for Y-axes).

\textit{max} Ending value for the axis, referred to the logical end of the plot (right hand end for X-, upper end for Y-axes).

\textbf{Auto-scaling}: If omitted, both \textit{min} as well as \textit{max} are substituted by the defaults, calculated from the corresponding minimal- and maximal values, which occurred during a previously executed \texttt{PLOTXYZ\texttt{}}-command. In this case, the \textit{libflag} is taken into account.

\texttt{PLOTLABEL(axindex,color,label$,unit$,distance,alignment,orientation,arrow, textsize, texttype)}
Plots the label- and the unit-string of an axis. In the following, the concatenation of the label- and the unit-string will be simply abbreviated as the ‘label’.

**position** position of the ‘label’ as a function of the axis-index (restricted to 0 to 4).

**label$** text string for the axis

**unit$** (physical) unit of the data referring to the axis, string type; unit may be an empty string (““). The plotted unit$- string is automatically extended by the order of magnitude of the corresponding axis in engineering units (for instance mV, µA or KΩ), if the axis labeling has to be normalized to a base value.

**distance** distance of the label (in pixel or millimeters respectively) to the corresponding frame (axis) line

**alignment** alignment of the label. Alignment = 0 leads to a left-, alignment =1 to a centered and finally, alignment =2 to a right aligned label.

**orientation** orientation of the label with respect to the corresponding axis. According to
0 = standard
1 = reverse (or upside down)

**arrow** 0 = no arrow symbol
1 = arrow symbol behind the axis label descriptor.

**textsize** size for the axis-labeling according to the information under CAD_TTYP. The typical value is textsize = 1.5 (to 2.5). The default is textsize = 2.

**texttype** font for the axis-labeling according to the information under CAD_TTYP. The default value is texttype = 44 for standard ASCII Windows Arial font.

Sometimes the labeling of an axis is intended to be done not at the site of the axis, for instance at the opposite site of the drawing, referred to the corresponding axis. In this case, the value of position has to be modified, in order to signalize the PLOTLABEL()-procedure, from which of the plotted axis the base unit (see unit$) must be taken. In this case, use the following formula to calculate value of position:

$$\text{position} = \text{label\_position} + \text{reference\_axis\_number} \times 16$$

**Example**

An Y-axis was plotted on the left hand side of the drawing as second axis after the initializing PLOTFRAME(-command: reference\_axis\_number = 2

The label\_position should be the right hand side Y-axis: label\_position=3. Then the final value for position should be position = 3 + 2*16 = 35.

PLOTTEXT(x0,y0,text$,color, textsize, texttype, orientation, alignment)
This command allows to print the content of a text string in a document, independent from a graph frame or axis.

\( x0 \)  X-coordinate reference for the text to plot.

\( y0 \)  Y-coordinate reference for the text to plot.

\( \text{text$} \)  String-expression for the text to plot.

\( \text{color} \)  color of the curve and the sample symbols (possible values: 0 to 15).

\( \text{textsize} \)  size for the text to plot according to the information under CAD_TTYP. The typical value is \( \text{textsize} = 1.5 \) (to 2.5). The default is \( \text{textsize} = 2 \).

\( \text{texttype} \)  font for the text to plot according to the information under CAD_TTYP. The default value is \( \text{texttype} = 44 \) for standard ASCII Windows Arial font.

\( \text{orientation} \)  Possible values from 0 to 3 indicate the 4 orthogonal plot orientations:

0 = left to right
1 = upward
2 = right to left
3 = downward

\( \text{alignment} \)  alignment of the text to plot referred to \( x0,y0 \). Alignment = 0 leads to a left-, alignment =1 to a centered-, alignment =2 to a right aligned text.

Example

\( \text{color}=15 \)'(0=black, 1=red.. 15=white)'
\( \text{size}=2.5 \)'(ca. 12 pt. ')
\( \text{type}=44 \) ' (ASCII Arial' )

\( \text{PLOTTEXT} \( (464,384,\"Plot this centered at 464, 384\",\text{color},\text{size},\text{type},0,1) \)

\'text$, color, textsize, typeface, plot from left to right, centered'

\( \text{PLOTLEGEND()} \) \( \text{PLOTLEGEND$()} \)  These commands allow to plot a graph curve legend to characterize a curve or a set of curves by means of labeling strings. Color (and optionally used line type and symbol marks) can be chosen accordingly.

\( \text{PLOTLEGEND( 0)} \)  initializes the command.

\( \text{PLOTLEGEND$()} \) \( \text{legend$ [,color, linetype, symboltype]} \)  sends the legend content. The \( \text{PLOTLEGEND$(-command may repeated n-times (usually identical to the count of curve graphs).

\( \text{legend$} \)  text label for the curve

\( \text{color} \)  text color - usually the color of the curve graph and symbols (possible values: 0 to 15).

\( \text{linetype} \)  Defines the type of the line displayed in the legend (usually identical to the linetype used for the graph curve).
For **linetype**, the following types are available:

<table>
<thead>
<tr>
<th>linetype</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>solid line</td>
</tr>
<tr>
<td>1</td>
<td>narrow dotted line</td>
</tr>
<tr>
<td>2</td>
<td>dashed line</td>
</tr>
<tr>
<td>3</td>
<td>dashed-dotted line</td>
</tr>
<tr>
<td>4</td>
<td>wide dotted line</td>
</tr>
</tbody>
</table>

The line width is taken according to the **textsize** parameter.

**lines** = - 1 will suppress the line plot output in the box, independent from the selection by the PLOTLEGEND(-argument **flag**).

**symboltype** Defines the symbol shape displayed in the legend. (usually identical to the type of the data points of the graph curve plotted, see also PLOTTYPE() above).

The coding of **symboltype** is as follows:

<table>
<thead>
<tr>
<th>symboltype</th>
<th>shape</th>
<th>symboltype</th>
<th>shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(Hollow) circle</td>
<td>8</td>
<td>down triangle</td>
</tr>
<tr>
<td>1</td>
<td>(Hollow) square</td>
<td>9</td>
<td>Rhombus</td>
</tr>
<tr>
<td>2</td>
<td>(Hollow) up triangle</td>
<td>10</td>
<td>filled circle</td>
</tr>
<tr>
<td>3</td>
<td>+</td>
<td>11</td>
<td>filled square</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>12</td>
<td>filled up triangle</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>13</td>
<td>filled slanted square</td>
</tr>
<tr>
<td>6</td>
<td>(Hollow) slanted square</td>
<td>14</td>
<td>filled down triangle</td>
</tr>
<tr>
<td>7</td>
<td>(not used)</td>
<td>15</td>
<td>filled rhombus</td>
</tr>
</tbody>
</table>

The size of the symbols is taken according to the PLOTLEGEND(-**textsize** parameter.

**symboltype** = - 1 will suppress the symbol plot output in the box, independent from the selection by the PLOTLEGEND(-argument **flag**).

**PLOTLEGEND**(true,x,y[flag,textsize,texttype,fbcolor,orientation,alignment])
Finalizes and executes the command.

**true** A value unequal zero (i.e. true = 1), to signalize the command-execution.

**x, y** X- resp. Y-coordinates of the upper left corner of the legend - box.

**flag** signals the usage of additional line- and symbol marks within - and of a frame around the box. The value of **flag** has to be coded as:

flag=enable line+2*enable symbol+4*enable frame+8*enable background

Each “enable” parameter may be 0 (off) or 1 (on).

**textsize** size for the text to plot according to the information under CAD_TTYP. The typical value is **textsize** = 1.5 (to 2.5). The default is **textsize** = 2.
texttype  font for the text to plot according to the information under CAD_TTYP. The default value is texttype = 44 for standard ASCII Windows Arial font.

fbcolor  Frame- and background colors. The value of fbcolor has to be coded as:

fbcolor = color of frame + color of background*256.

If flag signals frame or background usage, but fbcolor is omitted, the frame - and background colors of the last PLOTFRAME(-command are used as default.

orientation  Possible values from 0 to 3 indicate the 4 orthogonal plot orientations:

0 = left to right
1 = upward
2 = right to left
3 = downward

alignment  When omitting this parameter, the legend box is “fixed” at the leftmost upper edge of the text box (res. at the text start) relative to x, y. Other values will code different possible alignments in X- as well as Y-direction:

1 = X-centered
2 = X-right aligned
4 = Y-centered
8 = Y-bottom aligned

Use the sum of X- and Y-alignment values to align in both directions, for instance use alignment = 1 + 4 = 5 for XY-centered boxes.

Example

PLOTLEGEND(0)  'PLOTLEGEND(-command initialized'
for k=1ton
  'filling in n legend lines'
  PLOTLEGEND$(LEGEND$(k-1),k,k-1)
  '(legend[,color,symboltype,linetype])'
  'sends the string LEGEND$((), the colors and symboltypes'
  'from 1 to n, no linetypes are sent: default 0=solid is used)'
next
PLOTLEGEND(1,482,362,15,2,44,8+256*14,0,5)
  '(true, x, y[,linena*1 + symbena*2 + framena*4 + bgndena*8, size=2, '
  'Arial, framecol=8, bgndcol=14, from left to right, XY centered])'

CAD_DOC(flag[,file$])  With the CAD_DOC-function it is easy to export high quality vector graphics.

flag  Depending on the parameter ‘flag’, the following output devices are selected:

<table>
<thead>
<tr>
<th>flag</th>
<th>output device</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>Windows standard printer; in addition, the value of flag gives the number of copies to be printed.</td>
</tr>
<tr>
<td>= 0</td>
<td>An EMF-file will be exported</td>
</tr>
<tr>
<td>= -1</td>
<td>The CAD-drawing is exported into the Windows clipboard (in the EMF-format)</td>
</tr>
<tr>
<td>= -2</td>
<td>The CAD-drawing is plotted on the screen (= preview)</td>
</tr>
</tbody>
</table>
`file$` defines the filename under which the data will be saved in case that `flag`=0. It may be either a string expression. If a complete path is defined in `file$`, the file is saved there, if not, the default path for graphics, defined in the UTIL section of Thales is used. Example: `file$="c:\thales\examples\test.emf"`

If an error occurs during the export, i.e. the file cannot be saved on the selected device, the graphic is copied into the clipboard, instead.

We recommend to check the example below. Therefore, select and copy the following text into the clipboard and insert the text into the (plain) script editor (= ‘copy and paste’). Then, check and link the script by pressing the <link script> button. Finally run the script.

In addition, you can modify some of the parameters and check the influence on the output during the runtime of the script.

Example

```plaintext
SCRIPT1
'fori=0to100:XYZ(i,0)=i:XYZ(i,1)=(rnd(i)-.5)*10:next'
fori=0to10:XYZ(i,1)=sin(i/10)+cos(i/3)*.3:XYZ(i,0)=i*1e-5:next
'PLOTTYPE(lines,symbols,calccnt,smoothw,order,splinep,splinexy,errtype)
'lines format: type.size. ' '��������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������������
framena: Enables outer frame area (e.g. reserved area for labeling).
0: no envelope frame, >0: envelope frame width (pixel resp. mm).
framcol: Background color of envelope frame, if enabled.

PLOTFRAME(100,100,320,256,15,1,0,48,14,1)

PLOTXYZ(sindx,eindx,curvtyp,color,logflg,libflg,XMin,XMax,YMin,...)
PLOTXYZ() plots the curve contained in the float array XYZ(). in the area
reserved by PLOTFRAME XYZ(). The first column
(n,0) contains the x-data, the second (n,1) y-data and the third z-data.

If a column (e.g. XYZ(n,0) for x-values) is cleared, it is assumed as
implicit and is automatically filled with its indices.

sindx: start index of XYZ() rows for data to plot (typically zero).

eindx: end index of XYZ(), last row to plot.

curvtyp: coding for the type of curve/plot. It contains the'
information, if curve and/or symbols shall be plotted'
or if fitted curve data shall be used.

value  meaning
0  curve represented by symbols for the samples.
1  curve linear interpolated between samples.
2  curve represented by both symbols as well as.
   linear interpolated lines.
4  smoothed curve and symbol smoothed samples are displayed.
5  only smoothed curve is displayed.
8  regression curve and symbol samples are displayed.
9  only regression curve is displayed.
12  splined curve and symbol samples are displayed.
13  only splined curve is displayed.

color: curve and symbol color'

logflg: code for lin/log display:
          bit 0 reset/set: x-axis linear/logarithmic
          bit 1 reset/set: y-axis linear/logarithmic

libflg: code for liberation space between frame and curve
          bit 0 reset/set: x-axis full/shrink
          bit 1 reset/set: y-axis full/shrink

XMin,XMax,YMin,YMax... Fixed scaling limits for the curve. Omitted (or set to equal)
limits causes auto scaling.

PLOTXYZ(0,10,4,2,0,3)

PLOTAXIS(Position,Color,Grid,Gridcol,Textsize,Texttype)
PLOTAXIS() scales and labels the axes of the plot frame'
Position: 0=bottom x, 1=left side y, 2=top x, 3=right side y, 4=z-axis.
   scale is placed inside the frame. Add a value of 8 to place'
   the scale outside of the frame.

Color: axis scale and numbers color.

Grid: enable (Grid=1) or disable (Grid=0) grid scale.

Textsize, Texttype: same as in "selection of text attributes CAD_TTYP"
described. Default is size 3, type 28 = Windows Arial ASCII.

PLOTAXIS(0,3,1,15)
PLOTAXIS(1,2,1,15)

PLOTLABEL(Position,"Label","Unit",distance,justification,turn,arrow)
PLOTLABEL() labels the axis at the position like described above.
Label and unit may be set to empty strings ="", if only the order of'
magnitude must be displayed.

distance: distance of the label to the according frame outline.

justification: 0=left-, 1=centered-, 2=right justified.

turn: 0=along the frame line oriented in read direction,
     1=turned.

arrow: 0=no, 1 optional add of a direction arrow.

PLOTLABEL(2,3,"Time / \"s\",2,1)
Instead of exporting vector graphics by means of the CAD_DOC-function, the complete actual Thales-window or a certain part of it may be exported as a bitmap.

**flag** Depending on the parameter 'flag', the following output modes are selected:

<table>
<thead>
<tr>
<th>flag</th>
<th>output device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The screenshot is exported into the Windows clipboard (in the bitmap-format of the screen)</td>
</tr>
<tr>
<td>1</td>
<td>A bitmap-file will be exported. file$ has to be set accordingly</td>
</tr>
<tr>
<td>2+</td>
<td>Print output to the Windows standard printer.</td>
</tr>
<tr>
<td>2</td>
<td>Print standard portrait</td>
</tr>
<tr>
<td>3</td>
<td>Print standard portrait, invert black &amp; white</td>
</tr>
<tr>
<td>4</td>
<td>Print landscape</td>
</tr>
<tr>
<td>5</td>
<td>Print landscape, invert black &amp; white</td>
</tr>
</tbody>
</table>

**xlo, ylo** X- & Y-coordinate references in pixel units of the lower left corner of the window part to be exported. The origin is the lower left corner of the Thales window.

**xup, yup** X- & Y-coordinate references in pixel units of the upper right corner of the window part to be exported.

**file$** String-expression for target name and path of the bitmap file. An empty string must be present as place holder in the case of a printer hardcopy.

**xpos, ypos** X- & Y-coordinate references for the destination of the printout in the document in units of 1/16 mm. The origin is the upper left corner of the document.

**zoom** Zoom-factor scaling of the printout on the document in mm/pixel*16.

**Example**

```
HARDCOPY(0) 'Copies the pixel contend of the active Thales window to the clipboard '
HARDCOPY(0,40,20,240,120)'Copies the pixel content between XY-position (40,20) and ' 'position (240,120) to the clipboard'
HARDCOPY(1,40,20,240,120,"c:\temp\temp.bmp")'Saves the pixels between position ' ' (40,20) and position (240,120) into the file "temp.bmp" using the path "c:\temp"
HARDCOPY(2,40,20,240,120,"",160,80,3)'sends the window clipping to the printer.'
'The XY-print-position is (10,5) cm. A screen pixel represents 3/16 mm. '
```

SCRIPT1

SCRIPT_END
15. Advanced Programming: Script Functions, Special Labels, Menu Techniques

15.1 Script Functions

SCRIPT provides a set of useful functions, which are not implemented as SCRIPT Keywords but can be called with theandiBasic keyword "$fn" (omitted in the table), followed by the function name. Functions returning strings end with a $-character. Parameters in ()-parenthesis are obligatory, parameters in []-brackets are optional. In the following table 'EX:' denotes an example, ‘APP:’ denotes an application.

<table>
<thead>
<tr>
<th>Function name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lim(x,min,max)</td>
<td>returns x, clamped to min, max’</td>
</tr>
<tr>
<td>lg(x)</td>
<td>returns ld10(abs(x)), except for x=0: returns –200. ld10 is the 10base logarithm</td>
</tr>
<tr>
<td>Str$(x)</td>
<td>returns str$(x) to 5 significant digits</td>
</tr>
</tbody>
</table>
| Date$(d$) | returns MM.DD.JJJJ (English mode) or DD.MM.JJJJ (German mode) from d$=DDMMJJ  
EX: print fnDate$("22.02.46") -> "Feb,02.1946" |
| DT$(t) | returns MM,DD.JJ (English mode) or DD.MM.JJ (German mode) from time/ms since 1.1.1980.  
EX: print fnDT$(0,),fnDT$(1e7) -> "01,01.80 04,25.80" |
| L7$(d$) | returns effective stringlength for 7-segment font (digits count 2, characters ‘!._’ count 1) |
| VAL$(x[,s]) | returns str$(x) in engineers units. s defines significant digits  
s=0->5 digits, s=1->4 digits, s=1->6 digits...: sdigits= 5-s  
EX: print fnVAL$(Pi), fnVAL$(Pi*1e5), fnVAL$(Pi,-2) ->"3.1416 314.16K 3.141593" |
| time1$("hhmmss") | returns "hh:mm:ss" |
| time2$("hh:mm:ss") | returns "hhmmss" |
| TNAME$(res) | returns "JJMMDDhhmmss" as unequivocal namestring from date and time  
EX: on Jun,10.2005 at 10:11:12 printfnTNAME$(0) -> "050610101010"  
EX: on Jun,10.2005 at 10:11:12 printfnTNAME$(2) -> "0506101010"  
res >0 cuts #res trailing characters (e.g. suppressing seconds) |
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time$(s)</strong></td>
<td>returns &quot;hhmmss&quot;, if $s&lt;8640000, else returns &quot;hhmmss&quot;</td>
</tr>
<tr>
<td><strong>Time(t$)</strong></td>
<td>returns $s$ from &quot;hhmmss&quot;. EX: print fnTime(&quot;180000&quot;) -&gt; 64800</td>
</tr>
</tbody>
</table>
| **TI(0,[Time$]Date$)** | TI(0) returns time in milliseconds referred to 1.1.1980  
                    TI(0,Date$) returns Date$ in ms ref. to 1.1.1980 in multiples of 86400  
                    TI(0,Time$Date$) returns Time$Date$ in ms.  
                    EX: on Jan,5.2006 at 21:09:27 print fnTI(0) -> 820962566.86  
                    EX: printfnTI(0,"030180") -> 72800                                                                                          |
| **Dtt$(d$)**      | returns "DDMMYY" from "MM.DD.YY" or "DD.MM.YY"                                                                                                                                                       |
| **Dti$(t$)**      | returns "hhmmss" from "hh:mm:ss"                                                                                                                                                                       |
| **Dtti$(d$t$)**   | returns "DDMMYYhhmmss" from "MM,DD.YY hh:mm:ss", "DD.MM.YY hh:mm:ss"                                                                                                                                  |
| **VAL(num$)**     | returns numeric value from number strings of all formats                                                                                                                                               |
| **LSPC$(a$)**     | returns $a$, leading spaces stripped                                                                                                                                                                   |
| **TSPC$(a$)**     | returns $a$, trailing spaces stripped                                                                                                                                                                  |
| **NSPC$(a$)**     | returns $a$, leading and trailing spaces stripped                                                                                                                                                      |
| **LOG$(mess$)**   | appends the message string mess$ to the logfile "c:\flink\flink.log"                                                                                                                              |
| **WAITDISP(x,y,head$)** | opens a display-bar window with the header head$ at x,y                                                                                                                                             |
| **WAITBAR(Frac)** | displays the bar. 0->empty bar, 1-> full bar                                                                                                                                                           |
| **PLACE$(s$,r$,[Pos[,Len]])** | replaces in the source string s$ at the position Pos a partial string with the length Len by the replacement string r$  
                       Len=0 inserts r$ at Pos without removing characters  
                       EX: print fnPLACE("Source is not true","always",11,3) -> "Source is always true"  
                       print fnPLACE("Source is not true","always",15) -> "Source is not always true"                                                                                     |
| **BLANK$(s$,r$)** | replaces partial strings of the source s$, identical to r$ by blanks. EX: print fnBLANK\("ABXCDXYEF",\"XY\"\) -> "ab cd ef"                                                                                           |
15.2 Complex Menus

General

It is the main goal of SCRIPT to support the Thales user with an easy way to automate frequently recurrent tasks. Many scripts will therefore exhibit only one or a small set of script buttons which in turn may activate sub-functions like menus, toggle-groups and so on.

However, more challenging applications may be managed easily by SCRIPT too. A frequent problem is for instance to display instant status information or to support several sub-functions on the main panel of the script screen from start, without first pressing a script button.

A set of special control variables and sub-routine label names are reserved for such purposes. The application of the reserved label ‘START::’ was already explained in chapter 3.4.1. A second reserved label can be used for controlled restoring of settings, which were intermediately changed by the script application. If, for instance, the script changes the settings of the color look up table, a restore routine can be inserted after the label ‘END::’, which is automatically entered, when SCRIPT is left.