Software Design Description for Sierra III - 0062 - Scripting Definition

Revision:	1.7
Status:	Approved

Approvals:	Signature	Date
Project Team Leader (optional)		
Senior Software Developer (mandatory)		
Senior Software Developer (mandatory)		
Senior QA Specialist (mandatory)		

Revision History

Date	Author	Revision	Details
Jan 25, 2007	C. White	1.0	Initial release – based on ATS document
March 13, 2007	C. White	1.1	Comments from group
Jan 8, 2008	C. White	1.2	Added new features
Nov. 7, 2008	K Ryan B Yip	1.3	Added parameters for Receive ATS calculations
Nov. 10, 2008	K Ryan B Yip	1.4	Updated with reviewer comments - Removed descriptions of not implemented features
June 22, 2009	K. Ryan	1.5	Updated with new ATS commands. Added Sec. 6.7 User Interface. Changed Sierra references to Sierra II
June 30, 2009	K. Ryan	1.6	Updated with reviewer comments - Added timing parameters to Sec. 6.6
May 14, 2010	S. Poon	1.7	Added new script statements

Table of Contents

1 Overview

This document provides the software design and usage description for the VSI Scripting Language.

2 Definitions

Term	Definition
RF	Radio Frequency
ATS	Automated Test System
Transmit Signals	Signals transmitted by the Sierra II system and received by the ATS
Receive Signals	Signals transmitted by the ATS and received by the Sierra II system
AWG	Arbitrary Waveform Generator
Acqiris	A 8-bit Digitizer for RF data acquisition
Galil	A motion controller
XML	Extensible Markup Language

3 References

- Product Specification for Project Sierra III
- ATS Specification for Project Sierra III
- Software Specifications for the ATS for Project Sierra III

4 Assumptions

None.

5 General Scripting

5.1 Script Block

All scripts are encapsulated in an XML file with the following script block format.

Name	Type	Description
script	Element	script element
logFile	Attribute	Output file name for log messages
logClear	Attribute	“true" – clear log messages

```
<?xml version="1.0"?>
<script logFile="C:\out.txt" logClear="true">
</script>
```

General XML commands are added using the following statement.

```
<!– Non printable comment -->
```

5.2 Variables and Arrays

Script operations can use variables to store and manipulate values. Variables are defined using “$”. For example:

```
$count
$a
```

Global variables are defined using "$_", for example:

```
$_count
$_a
```

Variables are represented internally by variant types which have a specific format (for example, short, long, double). If one variable is copied from another, it will also copy its type. If unexpected results occur it is possible that a variable is using an incorrect type. For example if a variable is typed as a *short* and a value larger than 32767 is written, it will wrap. To determine the type of a variable, use the debug statement *gettype*.

```
<action code="assign($type, gettype($variable))"/>
<log>script:$type</log>
```

Variables can also be arrays. Arrays are defined by the use of square brackets with an index value "[4]". For example:

```
$count[4]
```

The following statement will assign a value to array position 4. Arrays are zero based.

```
<action code="assign($count[4], 75.3)"/>
```

Some operations operate on *pointers to arrays*. A pointer to the entire array can be specified using square brackets with no value "[]". For example, the following statement assigns a single value to every member of the array.

```
<action code="assign($count[], 75.3)"/>
```

A variable is always treated internally as an array with length 1. The following two statements are equivalent.

```
<action code="assign($a, 6)"/>
<action code="assign($a[0], 6)"/>
```

Internally, arrays are made larger than required such that they do not need to be constantly resized as more data is put into the array. Thus when the following statements is executed:

```
<action code="assign($a[0], 6)"/>
<action code="assign($a[55], 6)"/>
```

We can be assured that array position 0 and 55 will contain the value 6. Array positions between these indexes are defined as *empty*. However there will also exist some undefined number of array positions beyond 55 (typically 32) which are also defined as *empty*. This can cause confusion if one expects the array size to be simply 56 units long. For example, if the following assign statement is run, more than 56 elements will be set.

```
<action code="assign($a[], 6)"/>
```

To define an array to be a specific size, either by lengthening or cropping, use the *setlength* command.

```
<action code="setlength($a[], 256)"/>
```

An array will not automatically be extended if an index position is used that already exists. Thus, if you define an array to be 256 elements long using the set length command and reference position 255, the array will not be padded with additional values. If position 256 is referenced, the array will need to be expanded.

5.3 Log statements

Log statements output to the common output can be added within the script.

Name	Type	Description
log	Element	log element
level	Attribute	Type of log messages: • level • error • warning • info (default)
action	Attribute	Only action available is “clear” to clear the message log

```
<log>Message</log>
<log level="warning">Message</log>
<log action="clear"/>
```

Detailed messages can be constructed using script commands.

```
<action code="assign($logmsg, 'Custom message =  ')"/>
<action code="assign($logmsg, add($logmsg, $variable))"/>
<log>script:$logmsg</log>
```

Variables in log statements require the preceeding ‘script:’ to be interpreted as variables. Otherwise they will be displayed as a string.

5.4 Action Statements

Action statements control many of the operations managing variables and expressions.

Name	Type	Description
action	Element	action element
code	Attribute - required	Code bits that needs to be evaluated

The basic operation is as follows:

```
<action code=""/>
```

The following list shows the complete set of mathematical statements that can be used. Descriptions and examples follow.

Table 1 - Mathematical Expressions

Name	Description
assign	Assign the first argument to be the second
=	Assign the first argument to be the second
add	Add two arguments
+	Add two arguments
+=	Add two arguments, assign result to the first argument
subtract	Subtract second argument from the first argument
-	Subtract second argument from the first argument
-=	Subtract second argument from the first argument, assign result to the first argument
multiply	Multiply two arguments
*	Multiply two arguments
*=	Multiply two arguments, assign result to the first argument
divide	Divide two arguments
/	Divide two arguments
/=	Divide two arguments, assign result to the first argument
abs	Absolute value of the argument
pow	First argument to the power of second argument
log	Logarithm of argument one
log10	Base 10 logarithm of argument one
sqrt	Square root of argument one
sin	Sine of argument one
sinh	Hyperbolic sine of argument one
cos	Cosine of argument one
cosh	Hyperbolic cosine of argument one
tan	Tangent of argument one
tanh	Hyperbolic tangent of argument one
ceiling	Smallest integer no less the argument one
floor	Largest integer not greater than argument one
max	Maximum value of all the arguments
min	Minimal value of all the arguments
average	Calculate the average of all the arguments
changetype	Change the type of the arguments
setlength	Resize array
gettype	Gets the type of a variable
mod	Gets the modulus of a value
bitwiseOr	Performs a bitwise OR operation on the values of the arguments
|	Performs a bitwise OR operation on the values of the arguments
bitwiseAnd	Performs a bitwise AND operation on the values of the arguments
&	Performs a bitwise AND operation on the values of the arguments

5.4.1 Action: assign

Assigns a value to a variable.

```
<action code="assign($a, 6)"/>
<action code="=($a[55], 6)"/>
```

The assign statement can apply to all values in an array.

```
<action code="assign($a[], 6)"/>
```

Assign can also use the result of other action codes.

```
<action code="assign($value, subtract(4, 5))"/>
```

5.4.2 Action: add, subtract, multiply, divide

These expressions can apply to 2 arguments.

```
<action code="assign($value, add(4, 5))"/>
<action code="assign($value, subtract(4, 5))"/>
<action code="assign($value, multiply(4, 5))"/>
<action code="assign($value, divide(4, 5))"/>
```

If a divide by zero is evaluated an error will be thrown.

Add and *multiply* can apply to multiple arguments.

```
<action code="assign($value, add($a, $b, $c))"/>
<action code="assign($value, multiply($a, $b, $c))"/>
```

These expressions can also be applied to two arrays if they are of the same size and type. An error will occur if they are not. This operation can be applied to each array pair. For example, in the case of *add* the following occurs:

$value[0] = $a[0] + $b[0]
$value[1] = $a[1] + $b[1]

```
<action code="assign($value[], add($a[], $b[]))"/>
<action code="assign($value[], subtract(($a[], $b[]))"/>
<action code="assign($value[], multiply(($a[], $b[]))"/>
<action code="assign($value[], divide(($a[], $b[]))"/>
```

In the cases where arrays are evaluated, the return type must be an array and expressed as a pointer-to-array as shown in the above examples. For the array operations, errors will occur if:

1. the arrays are not the same size.
2. the types of the arrays are not the same.
3. there are empty values in the array.
4. a divide by zero occurs.

To prevent these problems, initialize your arrays using the following commands

```
<action code="setlength($a[], 256)"/>
<action code="assign($a[], 0)"/>
```

and possibly a change type operation if you think that some of the value types may have changed.

```
<action code="changetype($a[], ‘float’)"/>
```

5.4.3 Action: abs

Abs can apply to a single argument or an array.

```
<action code="assign($value, abs(-4))"/>
<action code="assign($value[], abs($a[]))"/>
```

For the array operation, an error will occur if any of the values are defined as empty.

5.4.4 Action: gettype

Gets the string type of the variable.

```
<action code="assign($type, gettype($variable))"/>
<log>script:$logmsg</log>
```

5.4.5 Action: changetype

Changes all values in an array to be a specific type.

```
<action code="changetype($a[], 'float')"/>
```

The following types are defined:

```
'char'
'byte'
'short'
'ushort'
'long'
'ulong'
'float'
'double'
'string'
```

5.4.6 Action: setlength

Forces an array to be a specific length either by lengthening or cropping the array. If an array is lengthened the original values will be kept and the array will be padded with empty values. If the array is cropped, some original values may be lost.

```
<action code="setlength($a[], 256)"/>
```

5.5 Sleep Statements

A sleep statement delays the script by a specified number of milliseconds.

Name	Type	Description
sleep	Element	sleep element
time	Attribute - required	Specify number of milliseconds

```
<sleep time="10"/>
<sleep time="script:$variable"/>
```

Variables in sleep statements require the preceeding 'script:' to be interpreted as variables.

5.6 Loop Statements

5.6.1 Simple Loop

Name	Type	Description
loop	Element	loop element
count	Attribute -	Specify the number of times loop repeats

	required	

```
<loop count="256">
     <!-- work -->
</loop>

<loop count="script:$count">
     <!-- work -->
</loop>
```

Loops can be broken. This loop will break out when $count hits 54.

Name	Type	Description
break	Element	break element, break out of the loop

```
<action code="assign($count, 0)"/>
<loop count="256">

     <if test="equal($count, 54)">
          <break/>
     </if>

     <action code="assign($count, add($count, 1))"/>
</loop>
```

5.7 Conditional Statements

Two types of conditions can be tested, if statements, and choose-when statements.

Name	Type	Description
if	Element	if element, for conditional branching
test	Attribute - required	A criteria for the conditional statement

```
<if test="equal($count, 54)">
     <action code="assign($count, 0)"/>
</if>
```

Name	Type	Description
choose	Element	choose element, used with when and otherwise elements for multiple conditional branching
when	Element	when element, see above
test	Attribute - required	An criteria for the conditional statement
Otherwise	Element	otherwise element, see above

```
<choose>
     <when test="equal($a,10)">
          <!-- work -->
     </when>

     <when test="equal($a,11)">
```

```
            <!-- work -->
        </when>

        <otherwise>
            <!-- work -->
        </otherwise>
    </choose>
```

5.7.1 Conditional Expressions

A number of conditional expression statements have been defined, each returning `$true` or `$false.` Statements can be combined using 'and', or 'or' commands.

The Boolean values, `$true` and `$false` are predefined as part of the script command langugae.

Table 2 - Boolean expressions

Name	Description
equal	true if two arguments are equal
notequal	true if two arguments are different
and	true if both arguments are true
or	true if one of or the two or both arguments is true
greater	true if first argument is greater than the second
less	true if first argument is less than the second

```
<action code="assign($ret, equal($a, $b))"/>
<action code="assign($ret, notequal($a, $b))"/>
<action code="assign($ret, and(equal($a, $b), equal($c, $d)))"/>
<action code="assign($ret, or(equal($a, $b), equal($c, $d)))"/>
<action code="assign($ret, greater($a, $b))"/>
<action code="assign($ret, less($a, $b))"/>
```

Some of the expressions can contain multiple arguments

```
<action code="assign($ret, equal($a, $b, $c, $d))"/>
<action code="assign($ret, notequal($a, $b, $c, $d))"/>
<action code="assign($ret, and($true, $true, $false)"/>
<action code="assign($ret, or($true, $true, $false)"/>
```

The statements 'greater' and 'less' must contain 2 arguments.

5.8 Link Statements

Additional functionality can be added by linking another file using the link command. This is similar to an #include statement.

Name	Type	Description
link	Element	link element
file	Attribute - required	Name of the script file

```
<link file="VsiFuncScan.vtf"/>
```

5.9 Function Statements

Functions can be used to encapsulate commonly used operations.

Name	Type	Description
function	Element	function element
name	Attribute - required	Name of the function.
return	Attribute	Return of the function
inline	Attribute	To indicate whether the function is inline (default = false)
param	Element	Parameter for the function
name	Attribute	Name of the parameter for the function

An example of a function returning a value:

```
<function name="TestFuncRet" inline="false" return="$ret">
     <param name="$a"/>
     <param name="$b"/>

     <action code="assign($ret, add($a, $b))"/>

</function>

<action code="assign($ret, TestFuncRet(1, 2))"/>
```

An example of a function not returning a value.

```
<function name="TestFunc" inline="false">
     <param name="$a"/>
     <param name="$b"/>

     <!– work -->

</function>

<action code="TestFunc(1, 2))"/>
```

Function arguments are optional.

```
<function name="TestFunc" inline="false">
     <!– work -->
</function>

<action code="TestFunc())"/>
```

Functions are `inline="false"` by default. This means that the function runs in its own context and thus cannot see variables defined outside the function and its own internal variables will not conflict with outside function variables. This is the safest as the function will not contaminate other variables in the script. A function can also be defined as `inline="true"`. In this case it is run in the same context as its parent and can see, and modify the parents variables.

All functions can see all other functions regardless of what context they are located in.

A variable that is made global by the insertion of an underscore can be seen by all functions regardless of what context they are located in

```
<action code="assign($_globalValue, 75.3)"/>
```

5.10 File Statements

Name	Type	Description
file	Element	file element
name	Attribute - required	Name of the file
action	Attribute - required	Action kind: • append – append to the end of the file • delete – remove the file • createNew – erase the file and create a new one • createIndexed – create a file with indexed name
separator	Attribute	Separator used in the file to separate each value
terminator	Attribute	Terminator used to end each line
maxElements	Attribute	Limit the number of output elements
return	Attribute	return expression

```
<file name="Data.csv" action="append">0000330</file>
```
- Appends "0000330" to the file "Data.csv". If the file is not available, it is created.

```
<file name="Data.csv" action="delete"/>
```
- Erases the file "Data.csv". If the file is not available, noting happens.

```
<file name="Data.csv" action="createNew">0000330</file>
```
- Creates a new file "Data.csv" and appends the value "0000330" to the beginning. If the file currently exists, it is overwritten.

```
<file name="Data.csv" action="append">0000330\n</file>
```
- Appends "0000330\n" to current file.

```
<file name="Data.csv" action="append">script:$a</file>
```
- Appends the single value $a to file.

```
<file name="Data.csv" action="append">script:$a[6]</file>
```
- Appends the single value $a[6] to file.

```
<file name="Data.csv" action="append" separator="," 
terminator="\n">script:$a[]</file>
```
- Appends the array $a[] to file. Each value is separated by "," and the entire set of values is terminated by "\n".

```
<file name="Data.csv" action="append" 
maxElements="64">script:$a[]</file>
```
- Limits the number of output elements to 64.

```
<file name="Data***.csv" action="createIndexed"
return="script:$a"/>
```
- Creates a new indexed file name and inserts it into the target variable $a.

```
<file name="script:$a" action="createNew">script:$a</file>
```
- Erases the file referenced in variable $a and adds the text $a to the file.

6 Component specific script elements

Additional modules can be imported using a module inclusion statement.

Name	Type	Description
using	Element	using element
name	Attribute - required	Name of the script component that will be required
progId	Attribute - required	COM identification for the script component

The following modules are currently available for use.

```
<using name="awg" progID="VsiVevo.TestScriptAwg"/>
<using name="ats" progID="VsiVevo.TestScriptAts"/>
<using name="galil" progID="VsiVevo.TestScriptGalil"/>
<using name="pdm" progID="VsiVevo.TestScriptPdm"/>
<using name="dcm" progID="VsiVevo.TestScriptDcm"/>
<using name="bf" progID="VsiVevo.TestScriptBeamFormer"/>
<using name="ui" progID="VsiVevo.ScriptUI"/>
```

Variables referenced in component specific script elements require the preceeding ‘script:’ to be interpreted as a variables. Otherwise, it will be interpreted as a string which, in most cases, is an error.

6.1 Arbitrary Waveform Generator - awg

The arbitrary waveform generator module controls the Tektronics AFG3101 waveform generator.

Name	Type	Description
awg	Element	For Arbitrary Waveform Generator
action	Attribute - required	**set** – issue commands to AWG that change the device setting or perform an action. **query** – issue command to AWG to query a device setting
command	Attribute - required	Low level option to send commands direct to awg

Table 3 - List of AWG commands supported for set action

Name	Description
shape	Shape of the waveform: **sine**, **square**, **ramp**, or **pulse**

Name	Description
mode	Mode of the waveform: **continuous**, **modulation**, **sweep**, or **burst**
cycle	Number of cycles of a burst signal
frequency	Frequency of the waveform
amplitude	Output amplitude of the waveform
offset	Offset level of the waveform
phase	Phase of the waveform
outputMode	Set the mode of the Trigger Output signal **SYNC**, or **TRIG**
outputState	Enable or disable output: **on**, or **off**
burstDelay	Delay time between trigger and output signal of a burst signal

```
<!-- Generate a burst of sine wave of 9 cycles  -->
<awg action="set" shape="sine"/>
<awg action="set" mode="burst"/>
<awg action="set" frequency="30MHz"/>
<awg action="set" cycle="9"/>
<awg action="set" amplitude="5"/>
<awg action="set" offset="0.8"/>
<awg action="set" phase="30 DEG"/>
<awg action="set" outputMode="TRIG"/>
<awg action="set" outputState="ON"/>
<awg action="set" burstDelay="20ns"/>
```

Commands can be sent directly to the AWG. Consult the Tektronics manual for a list of specific commands if they are required.

```
<!-- Generate a burst of square wave of 8 cycles -->
<awg command="FUNC SQU"/>
<awg command="BURS:MODE TRIG"/>
<awg command="BURS:NCYC 8"/>
<awg command="BURS:STAT ON"/>
<awg command="FREQ 10"/>
<awg command="VOLT:AMPL 0.1"/>
<awg command="VOLT:OFFS 0.0"/>
<awg command="PHAS:ADJS 0"/>
<awg command="TRIG:SOUR EXT"/>
<awg command="SAV 1"/>
<awg command="RCL 1"/>
```

6.2 Acqiris acquisition and beamformer receiving - ats

Name	Type	Description
ats	Element	General ats commands
action	Attribute - required	Perform specific actions as the following: • set – set the parameters • acquire – acquire data from acqiris • receive – receive data from beamformer • centerRxGraphs – center the receive graphs on the

		receive pulse
mode	Attribute - required	Mode of acquisition single – acquire and wait for single data (default) arm – arm an acquisition wait – wait for end of data acquisition
bin	Attribute	The element number (0 to 1023)
timeSize	Attribute	The size of the graph window to use when centering the receive graph

Table 4 - List of ATS commands supported for set action

Name	Description
samplesPerSec	Number of samples per second, for both acquire and receive
delayTime	Trigger delay time in seconds, for Acqiris acquisition
numOfSamples	Number of samples to capture, for Acqiris acquisition
fullScale	Full scale range in volts, for Acqiris acquisition
threshold	Minimum signal threshold for detecting transmit zero crossing (If maximum value of signal does not exceed this value, zero crossing will not be found)
transmitFreqMhz	Transmit frequency in Mhz, used in measuring receive delay profile
rxWindowSize	Size of receive window in samples, in which to search for received signal when finding receive group delay
rxAwgDelayNs	The delay applied to the incoming signal in nanoseconds. This will be used by the receive group delay calculation and for determining the depth to calculate the ideal delay profile
numCycles	Number of cycles in the received waveform. This will be used by the receive group delay calculation and for determining the depth to calculate the ideal delay profile

Before acquiring any data, run the DC offset correction and the TX clock synchronization using the following functions (defined in VsiFuncScan.vts):

```
<action code="DcOffsetCorrection"/>
<action code="SyncTxClock"/>
```

This will perform the same hardware calibration that is performed by the main software application when a transducer is selected and initialized.

6.2.1 Acquisition using Acqiris

To setup acquisition using the Acqiris card the following commands can be used. See the section on Galil commands for details on the galil set action. When using the ATS, this command is required to switch the ATS control circuit to allow transmit signals through to the Acqiris.

```
<ats action="set" delayTime="0.0"/>
<ats action="set" numOfSamples="8192"/>
<ats action="set" fullScale="1.0"/>
<ats action="set" threshold="40"/>
<ats action="acquire"/>
<galil action="set" output="transmit"/>
```

The following statement sets the samples per second parameter which defines the time scale for acquisition. The Acqiris card should always use a value of 2000.0 MHz.

```
<pdm action="write"
     name="Test/Ats/AtsSoft/SamplesPerSec"
     base="10">2000.0</pdm>
```

To acquire a single trigger from the Acqiris acquisition card when the card is being triggered from an external source (i.e. PRF triggers):

```
<ats action="acquire" bin="script:$a"/>
```

In this case, the results are placed into the bin position specified by the variable $a. If computer triggers are used the following commands can be utilized:

```
<ats action="acquire" mode="arm"/>

<!-- Send a SOL computer trigger -->
<pdm action="write"
     name="Hardware/Control/HwTimingGen/Cmp-Trig">00000001</pdm>
<dcm action="write"
     name="Hardware/Control/HwTimingGen/Cmp-Trig"/>

<!-- Receive data -->
<ats action="acquire" mode="wait" bin="script:$a"/>
```

In the previous example, we arm the acquisition module, fire the trigger and then wait for the results. If data does not arrive after approximately 1 second, an error will occur.

6.2.2 Acquisition using beamformer

To setup acquisition using the beamformer, the following commands can be used. The number of samples must match the number of acquisition samples set on the beamformer.

```
<ats action="set" delayTime="0.0"/>
<ats action="set" numOfSamples="256"/>
<ats action="set" fullScale="1.0"/>
<ats action="set" threshold="256"/>
<ats action="set" rxAwgDelayNs="13500"/>

<ats action="set" rxWindowSize="2000"/>
<ats action="set" quadX2Enable="false"/>
<ats action="receive"/>
<galil action="set" output="receive"/>
<galil action="set" output="pass through"/>
```

Set the SamplesPerSec parameter to the receive frequency used during this acquisition.

```
<pdm action="write"
     name="Test/Ats/AtsSoft/SamplesPerSec" base="10">20.0</pdm>
```

To acquire data from the beamformer using computer triggers:

```
<ats action="receive" mode="arm"/>

<!-- Send a SOL computer trigger -->
<pdm action="write"
     name="Hardware/Control/HwTimingGen/Cmp-Trig">00000001</pdm>
```

```
<dcm action="write"
     name="Hardware/Control/HwTimingGen/Cmp-Trig"/>

<ats action="receive" mode="wait" bin="script:$a"/>
```

6.2.3 Managing ATS data

When data is acquired from either the Acqiris or the beamformer a number of values are derived from the profiles and are available for use.

The zero crossing delay values:

```
<pdm action="read"
     name="Test/Ats/AtsSoft/Delay-Values-ZeroCrossing"
     type="float" return="$delayValues[]"/>
```

The group delay values:

```
<pdm action="read"
     name="Test/Ats/AtsSoft/Delay-Values-GroupDelay"
     type="float" return="$groupDelayValues[]"/>
```

6.3 Galil Motion Controller - galil

The Galil commands allow the movement of both the stepper motor and certain electronic switches.

Name	Type	Description
Galil	Element	For Galil motion controller
action	Attribute - required	• move – move to position • load – load and execute a galil script file • set – configure mode
element	Attribute - required	The element position the motor is to move to. Used with move action.

There are 258 possible Galil positions (256 elements, 1 home position, and 1 test pin). To home the motor system, send the motor position to 0.

```
<galil action="move" element="0"/>
```

Otherwise, to move the motor to a specific location on the breakout board, specify a number 1 to 257. Pin 257 is the test location.

```
<galil action="move" element="64"/>
```

Elements on the breakout board are not listed in order. Thus, if you start at element 1 and move sequentially through the elements, the path taken by the motor would be extremely long and inefficient. Helper functions are available to simplify this process.

The Galil interface can also be used to switch RX/TX signal paths. This is done using the Galil *output* attribute.

Table 5 - List of Galil commands supported for *output* attribute

Output Attribute value	Description
transmit	Sets the electronics board to the transmit path.
receive	Sets the electronics board to the receive path.

pass through	Sets the receive path to bypass the narrow band filter.
NB filter	Sets the receive path to use the narrow band filter.

```
<galil action="set" output="receive"/>
<galil action="set" output="pass through"/>
<galil action="set" output="transmit"/>
<galil action="set" output="NB filter"/>
```

6.3.1 Efficient motion functions

Link the Galil helper functions file:

```
<link file="VsiFuncPogo.vtf"/>
```

This file contains functions *GetNextPogoPin*, *ResetNextPogoPin*, *HomeGalil*, *AcquireRange, MapElementToCb, MapElementToChannel,* and *MapElementToCbAndChannel*. These enable efficient motion of the motor over the breakout board. The following example demonstrates how this is used. These scripts require the "Test/Ats/AtsSoft/Pogo-Map" parameter which has been preset with the breakout board mapping order.

```
<loop count="256">
     <action code="assign($ret, GetNextPogoPin(1, 64))"/>

     <if test="equal($ret, $false)">
          <break/>
     </if>
     <if test="notequal($ret, $false)">
          <galil action="move" element="script:$ret"/>
          <!-- work -->
     </if>
</loop>
```

The *MapElementToCb, MapElementToChannel* and *MapElementToCbAndChannel* functions can be used to find the channel board and channel associated with a given array element, as follows:

```
<action code="assign($channelBoard, MapElementToCb($element))"/>
<action code="assign($channel, MapElementToChannel($element))"/>
<action code="assign($cbAndChannel[],
     MapElementCbAndChannel($element))"/>
```

6.3.2 Beamformer data acquisition data

Sometimes the acquisition data needs to be modified after an acquisition. Consider the following acquisition:

```
<ats action="receive" mode="wait" bin="1"/>
```

This might be set to acquire 256 data points and put the data into internal bin 1. This data is also written to the parameter:

```
"Test/Ats/AtsSoft/QI-Data"
```

We can read and write to this parameter to modify the data and then set it back into the required bin.

```
<action code="assign($data, 0)"/>
<action code="setlength($data[], 1024)"/>
<action code="changetype($data[], 'long')"/>

<action code="setlength($dataTemp[], 1024)"/>
<action code="assign($dataTemp[], 0)"/>
<action code="changetype($dataTemp[], 'long')"/>

<action code="setlength($dataInter[], 1024)"/>
<action code="assign($dataInter[], 0)"/>
<action code="changetype($dataInter[], 'long')"/>
```

Now we read the data from the QI-Data parameter and do some math.

```
<pdm action="read" name="Test/Ats/AtsSoft/QI-Data" type="long"
     return="$dataTemp[]"/>
<action code="changetype($dataTemp[], 'long')"/>
<action code="setlength($dataTemp[], 1024)"/>
<action code="assign($dataInter[], add($dataTemp[], $data[]))"/>
<action code="assign($data[], $dataInter[])"/>
```

Now we do some more math and write it back to the QI-Data parameter.

```
<action code="setlength($count[], 1024)"/>
<action code="assign($count[], 32)"/>
<action code="assign($dataInter[], divide($data[], $count[]))"/>
<pdm action="write" name="Test/Ats/AtsSoft/QI-Data"
     type="long">script:$dataInter[]</pdm>
```

Now we “set” this parameter back into bin 1.

```
<ats action="set" bin="1"/>
```

6.4 Parameter Data Manager - pdm

Values can be set and read to the pdm using this interface.

Name	Type	Description
pdm	Element	For setting to and getting parameters from pdm
action	Attribute - required	read – read parameter value from pdm write – write parameter value to pdm set – configure mode
name	Attribute - required	Name of the parameter
bigEndian	Attribute	Order of the bytes
base	Attribute	Base of the number, 16 for hex
link	Attribute	Link to another script file
return	Attribute	Return parameter value to a variable. Used with read action

Name	Type	Description
type	Attribute	Indicate the type of binary data, it can be one of the following: • char • byte • short • ushort • long • ulong • float • float • double
link	Element	Links to another file
file	Attribute - required	The name of the file

A value can be read into a script parameter:

```
<pdm action="read" name="App/Settings/Sys/Unfreeze"
     return="$a"/>
```

A value can be written to a script parameter, by default the value is base 16:

```
<pdm action="write"
     name="Hardware/Tx/HwTx/Enable">0000FFFF</pdm>
```

For memory parameters large strings of numbers can be read in from a file using the link statement:

```
<pdm action="write" name="Focus"><link file="delay.txt"/></pdm>
```

The pdm can also handle arrays using a memory parameter.

```
<pdm action="read" name="Ats/AtsSoft/Data" type="long"
     return="$a[]"/>
```

This will read the data in memory type “At/St/Data” into the array parameter a[] treated as type longs. We can also write it back.

```
<pdm action="write" name="Ats/AtsSoft/Data"
     type="long">script:$a[]</pdm>
```

6.5 Device Control Manager – dcm

Hardware parameters can be written to hardware via TCPIP using this interface.

Name	Type	Description
dcm	Element	For writing values to hardware
action	Attribute - required	• read – read a value from hardware • write – write a value to hardware
name	Attribute - required	The full name of the hardware parameter to process

For example to write a specific value to hardware the value must first be written to the hardware parameter:

```
<pdm action="write"
     name="Hardware/Tx/HwTx/Enable">0000FFFF</pdm>
```

The value is then written to hardware using the dcm interface:

```
<dcm action="write" name="Hardware/Tx/HwTx/Enable"/>
```

6.6 Beamformer Control – bf

The beamformer can be programmed using this interface. The beamformer is programmed by specifying a number of beamformer specification tables which define the configuration of different aspects of beamformer control. Often the same information is put into different specifications. The outputs of the processed specifications are written to hardware parameters which can subsequently be written to hardware using the dcm interface.

The following specifications can be defined for beamformer control.

Name	Description
array	configuration of array
txcommon	common TX configuration
rxcommon	common RX configuration
tgc	time gain curve configuration
transmit0	transmit profile mode 0 configuration
txtest	debug and test parameters to affect all transmit profiles
receive0	receive profile mode 0 configuration
rxtest	debug and test parameters to affect all receive profiles
lineselect	line sequence parameters
timing	frequency and mode timing parameters

Not all profiles are required to be set. The following specifications should always be defined:

- array
- txcommon
- transmit0
- lineselect

If receive profiles are to be generated include the following:

- rxcommon
- receive0

Optional specifications are:

- tgc
- txtest
- rxtest

The following is an example showing a complete specification showing the required specifications (*array*, *txcommon*, *transmit0* and *lineselect*).

```
<bf>
     <!-- Global array specifications -->
     <array name="Default">
          <parameter name="Pitch" value="0.100"/>
```

```
        <parameter name="Lens-Thickness" value="0.4"/>
        <parameter name="Sound-Speed-Lens" value="1015000"/>
        <parameter name="Sound-Speed-Tissue" value="1500000"/>
        <parameter name="Array-Length" value="25.6"/>
        <parameter name="Num-Elements" value="256"/>
    </array>

    <!-- Transmit mode 0 specifications -->
    <transmit0>
        <parameter name="Focal-Depth" value="10.0"/>
        <parameter name="Frequency" value="20000000.0"/>
        <parameter name="Cycles" value="1"/>
        <parameter name="Trg-Setup" value="16"/>
        <parameter name="Pipeline-Delay" value="8"/>
        <parameter name="Steering-Angle" value="0.0"/>
        <parameter name="High-Density" value="false"/>
        <parameter name="TXAH" value="0x0000003F"/>
        <parameter name="TXAM" value="0x0000003F"/>
        <parameter name="TXAE" value="0x00000000"/>
        <parameter name="TXB" value="0x0000003F"/>
        <parameter name="f-Number" value="1.5"/>
    </transmit0>

    <!-- TX Common specifications -->
    <txcommon>
        <parameter name="Enable-CB0" value="0xFFFF"/>
        <parameter name="Enable-CB1" value="0xFFFF"/>
        <parameter name="Enable-CB2" value="0xFFFF"/>
        <parameter name="Enable-CB3" value="0xFFFF"/>
        <parameter name="VP-Mode1" value="25.0"/>
        <parameter name="VP-Mode2" value="0.0"/>
        <parameter name="Enable-VP-Mode1" value="true"/>
        <parameter name="Enable-VP-Mode2" value="false"/>
    </txcommon>

    <!-- Line sequence specification -->
    <lineselect>
        <parameter name="Mode-Id" value="1001"/>
        <parameter name="B-Mode-Width" value="25.6"/>
        <parameter name="High-Density" value="false"/>
        <parameter name="Num-Focal-Depths" value="1"/>
    </lineselect>

</bf>
```

As with other custom components, script values can be used, but only for the *value* attribute:

```
    <parameter name="Focal-Depth" value="script:$a"/>
```

The use of script values is not available for parameters that are assigned hex values (like the *enable* parameters).

Receive profiles can be defined by adding the receive specifications (*rxcommon* and *receive0*):

```
<!-- RX Common specifications -->
<rxcommon>
     <parameter name="Enable-CB0" value="0xFFFF"/>
     <parameter name="Enable-CB1" value="0xFFFF"/>
     <parameter name="Enable-CB2" value="0xFFFF"/>
     <parameter name="Enable-CB3" value="0xFFFF"/>
</rxcommon>

<!-- RX specifications -->
<receive0>
     <parameter name="Frequency" value="20000000.0"/>
     <parameter name="f-Number" value="1.5"/>
     <parameter name="Max-Channels" value="64"/>
     <parameter name="Gate-Depth" value="1.75"/>
     <parameter name="Depth-Cycles" value="256"/>
     <parameter name="Steering-Angle" value="0.0"/>
     <parameter name="High-Density" value="false"/>
     <parameter name="Gap-Delay" value="200"/>
     <parameter name="Mux-Delay" value="600" />
</receive0>
```

When a specification is processed a number of hardware parameters are updated and may be written to hardware. The following table describes the modifications performed:

Specification	Hardware parameters updated
array	VSI_PARAMETER_HW_TXCBCONT_AP_SELECT_CB1 VSI_PARAMETER_HW_TXCBCONT_AP_SELECT_CB2 VSI_PARAMETER_HW_TXCBCONT_AP_SELECT_CB3 VSI_PARAMETER_HW_TXCBCONT_AP_SELECT_CB4 VSI_PARAMETER_HW_RXCBCONT_AP_SELECT_CB1 VSI_PARAMETER_HW_RXCBCONT_AP_SELECT_CB2 VSI_PARAMETER_HW_RXCBCONT_AP_SELECT_CB3 VSI_PARAMETER_HW_RXCBCONT_AP_SELECT_CB4 VSI_PARAMETER_HW_TXCBCONT_AP_ENABLE_CB1 VSI_PARAMETER_HW_TXCBCONT_AP_ENABLE_CB2 VSI_PARAMETER_HW_TXCBCONT_AP_ENABLE_CB3 VSI_PARAMETER_HW_TXCBCONT_AP_ENABLE_CB4 VSI_PARAMETER_HW_TXCBCONT_ELM_SELECT_CB1 VSI_PARAMETER_HW_TXCBCONT_ELM_SELECT_CB VSI_PARAMETER_HW_TXCBCONT_ELM_SELECT_CB3, VSI_PARAMETER_HW_TXCBCONT_ELM_SELECT_CB4
txcommon	VSI_PARAMETER_HW_TXCBCOMM_VP_MODE1 VSI_PARAMETER_HW_TXCBCOMM_VP_MODE2 VSI_PARAMETER_HW_TXCBCOMM_CYCLES_MODE1 VSI_PARAMETER_HW_TXCBCOMM_CYCLES_MODE2 VSI_PARAMETER_HW_TXCBCOMM_CYCLES_MODE3 VSI_PARAMETER_HW_TXCBCOMM_CYCLES_MODE4 VSI_PARAMETER_HW_TXCBCOMM_ENABLE_CB1

Specification	Hardware parameters updated
	VSI_PARAMETER_HW_TXCBCOMM_ENABLE_CB2 VSI_PARAMETER_HW_TXCBCOMM_ENABLE_CB3 VSI_PARAMETER_HW_TXCBCOMM_ENABLE_CB4
rxcommon	VSI_PARAMETER_HW_RXCBCOMM_ENABLE_CB1 VSI_PARAMETER_HW_RXCBCOMM_ENABLE_CB2 VSI_PARAMETER_HW_RXCBCOMM_ENABLE_CB3 VSI_PARAMETER_HW_RXCBCOMM_ENABLE_CB4
tgc	VSI_PARAMETER_HW_TGCCB_ENABLE VSI_PARAMETER_HW_TGCCB_FGAIN VSI_PARAMETER_HW_TGCCB_CURVE_MODE1 VSI_PARAMETER_HW_TGCCB_CURVE_MODE2 VSI_PARAMETER_HW_TGCCB_CURVE_MODE3 VSI_PARAMETER_HW_TGCCB_CURVE_MODE4
transmit0	VSI_PARAMETER_HW_TXCBCONT_DELAY_PROF_MODE1 VSI_PARAMETER_HW_TXCBCOMM_GATE_LENGTH
receive0	VSI_PARAMETER_HW_RXCBCONT_INIT_DELAY_MODE1 VSI_PARAMETER_HW_RXCBCOMM_DYN_FOC_MODE1 VSI_PARAMETER_HW_RXCBCOMM_LENGTH_MODE1 VSI_PARAMETER_HW_RXCBCOMM_MAXDEALY_MODE1
lineselect	VSI_PARAMETER_HW_TXCBCOMM_SEQUENCE_MODE_SELECT VSI_PARAMETER_HW_TXCBCOMM_SEQUENCE_LENGTH VSI_PARAMETER_HW_TXCBCONT_LINE_SEQUENCE VSI_PARAMETER_HW_TXCBCOMM_BMODE_SETTING1 VSI_PARAMETER_HW_RXCBCOMM_SEQUENCE_MODE_SELECT VSI_PARAMETER_HW_RXCBCOMM_SEQUENCE_LENGTH VSI_PARAMETER_HW_RXCBCONT_LINE_SEQUENCE
timing	VSI_PARAMETER_HW_CLOCKCONT_TX_PLL_CONT VSI_PARAMETER_HW_CLOCKCONT_TX_PLL_REGISTERS VSI_PARAMETER_HW_CLOCKCONT_RX_PLL_CONT VSI_PARAMETER_HW_CLOCKCONT_RX_PLL_REGISTERS VSI_PARAMETER_HW_CLOCKCONT_DOP_PLL_CONT VSI_PARAMETER_HW_CLOCKCONT_DOP_PLL_REGISTERS VSI_PARAMETER_HW_CLOCKCONT_MUX_CONT VSI_PARAMETER_HW_CLOCKCONT_TX2RX_DELAY VSI_PARAMETER_HW_CLOCKCONT_RX_MUX_DELAY VSI_PARAMETER_HW_TIMEGEN_B_LINES_FRAME_INTER VSI_PARAMETER_HW_TIMEGEN_BRX_GATE VSI_PARAMETER_HW_TIMEGEN_B_TOF VSI_PARAMETER_HW_TIMEGEN_DOP_BLOCK_SIZE VSI_PARAMETER_HW_TIMEGEN_DOP_ENSEM_SIZE VSI_PARAMETER_HW_TIMEGEN_DOP_LINES_FRAME VSI_PARAMETER_HW_TIMEGEN_DRX_GATE VSI_PARAMETER_HW_TIMEGEN_D_TOF VSI_PARAMETER_HW_TIMEGEN_DTR_PRF VSI_PARAMETER_HW_TIMEGEN_MRX_GATE VSI_PARAMETER_HW_TIMEGEN_MTR_PRF VSI_PARAMETER_HW_TIMEGEN_TX_RX_SYNC VSI_PARAMETER_HW_TIMEGEN_TX_TRIG_SETUP VSI_PARAMETER_HW_TIMEGEN_CF_BLOCK_REPEAT_NUM VSI_PARAMETER_HW_TIMEGEN_CF_BLOCK_SIZE VSI_PARAMETER_HW_TIMEGEN_DOP_LINES_INTER VSI_PARAMETER_HW_TIMEGEN_DOP_PULSES_PRF

Write these values using the dcm interface. For example:

```
<dcm action="write" name="Hardware/Rx/HwRxCbCommon/Enable-CB1"/>
<dcm action="write" name="Hardware/Rx/HwRxCbCommon/Enable-CB2"/>
<dcm action="write" name="Hardware/Rx/HwRxCbCommon/Enable-CB3"/>
<dcm action="write" name="Hardware/Rx/HwRxCbCommon/Enable-CB4"/>
```

6.7 User Interface

A capture statement can be used to save a PNG image of the screen or one of the engineering mode windows.

Name	Type	Description
capture	Element	Capture element
type	Attribute - required	Specify whether to capture the entire screen or a specific window.
windowTitle	Attribute	The title of the window to capture.
windowClass	Attribute	The title of the window to capture.
file	Attribute - required	Name of the image file to be saved.

```
<ui>
 <capture type="window" windowTitle="ATS RX" file="Test-
capture.png"/>
</ui>
<ui>
 <capture type="screen” file="script:$variable"/>
</ui>
```

Variables in capture statements require the preceeding ‘script:’ to be interpreted as variables.