TransIT Meta Model Script Specification

1.) Script Constructs

The following list includes a full definition of the TransIT scripting language syntax, together with a case example for each construct. These constructs define the TransIT workflow modeling language:

A.) Script Constructs: The Model Tag

Syntax	**<model>…</model>**
Definition	This section introduces the basic script template upon which every model should be built. As previously stated, a script should contain a model XML tag, into which a name tag, a global declaration tag, a workflow tag, and a main tag are embedded. These tags are tackled and explained individually in the following text.
Rules	• The script must always be embedded in a <model> tag. • One <name> tag containing the script's name should always be present as a first child node to the model tag. • One global declaration tag should always be included, and it should always contain one <activityList> tag which has a size XML attribute, amongst other tags of other types, if necessary. • One <workflow> tag should always be present in the script, positioned after the <name> and <declaration> tags, containing the actual script workflow. • One <main> tag should always be present as the last child node of the <model> tag.

	• Uppercase or lowercase format may be used for tag definitions, there is no difference in operation since the engine will convert all the tags to lowercase during the parsing process. However parameters, and variable declarations are case sensitive, thus attention should be paid when assignining them. • Standard XML format rules apply, that is, every tag must be opened and close, attribute values should be embedded in double inverted commas, etc… for details about xml format standards please refer to the following web link: http://www.w3schools.com/xml/

Sample Script

```
<?xml version="1.0" encoding="utf-8" ?>

<model>
        <name><!-- Put Model Name Here --></name>


        <decl>
                <activityList size = "*n*">
                        <!--arrayOfActivities -->
                </activityList>
        </decl>

        <workflow>
                …
                …
                <!--Actual Workflow, segments &
                imperative language constructs. -->
                …
                …
        </workflow>

        <main>
                <!--A goto statement indicating which
                segment to execute first.-->
        </main>

</model>
```

B.) Script Constructs: Name Tag

Syntax	**<name>Alpha Numeric Value</name>**
Definition	

	The name tag simply serves as a data holder for the current name of the Transaction Model which is being defined in the script in question.
Rules	• A script should have one instance of the <name> tag. It should be included as the first child node inside a <model> tag. • The name tag has no XML attributes, and takes an alphanumeric inner text value, which represents the Model's Name.

Sample Script	`<?xml version="1.0" encoding="utf-8" ?>`
	
	`<model>`
	`        <name><!--Put Model Name Here --></name>`
	`        …`

C.) Script Constructs: Global/Local Declaration Tag

Syntax	**<decl>…</decl>**
Definition	The <decl> tag's main purpose is that of providing an indicator for a global or local variable declaration present in the script.
Rules	• A script should always have one instance of the <decl> tag included as the second child node inside a <model> tag, right after the <name> tag. This should include an <activityList> tag, amongst other global declarations of type <counter>. • Local declarations can also be present in script segments. These are also constituted by a <decl> tag, present in a segment tag, before the actual workflow code. A local <decl> tag possesses no attributes, and can have one or more children of type <counter>.

Sample Script	`<model>`
	`        <name><!--Put Model Name Here --></name>`

	`<decl>` `<activityList size = "*n*">` `arrayOfActivities` `</activityList>` … `<!--Variables of type <counter> -->` … `</decl>` …

D.) Script Constructs: ActivityList Tag

Syntax	**<activityList>Name of LLT</activityList>**
Definition	The activityList tag has the sole purpose of defining an abstract list of activities, which will be used in order to create the model. The activityList tag contains an XML attribute, named "size" which defines the size of the list. In essence the activityList has properties of an ArrayList where each position in the list signifies an activity.
Rules	• An <activityList> tag, should always be declared globally in a script definition. Only one instance if this tag is allowed per script. • The size attribute of this tag may be alphanumeric, since the value can either be definite, as in integer values, or indefinite, as in *n*, where *n* refers to the size of the list.
Sample Script	`<model>` `<name><!--Put Model Name Here --></name>` `<decl>` `<activityList size = "*n*">` `arrayOfActivities` `</activityList>` … `<!--Variables of type <counter> -->` … `</decl>` …

E.) Script Constructs: Counter Tag

Syntax	**<counter value = “V”>Name of Variable</activityList>** **where V is a Natural Number**
Definition	The counter tag is used in the <decl> tag in order to declare a local or global variable of type Integer. The Inner Text of this tag is considered to represent the variable name, while the value is stored inside a value attribute.
Rules	• While any amount of declarations is allowed, a <counter> tag may be used only inside a <decl> tag. • The value attribute of this tag must always be of type natural number, since the value can only be of definite type. • Counter tags can be assigned a value externally by <goto> statements. This is done if a <goto> statement has a parameter attribute which has the same name as a local variable in the segment it is calling. If this is the case, the local variable, takes the parameter’s value.

Sample Script	`<?xml version="1.0" encoding="utf-8" ?>`
	
	`<model>`
	`        <name>Put Model Name Here</name>`
	
	`        <decl>`
	`                <activityList size = "*n*">`
	`                        arrayOfActivities`
	`                </activityList>`
	
	`                <counter value = "0">globalk</counter>`
	`        </decl>`
	
	`        <workflow>`
	`                <segment id = “A Segment”>`
	`                        <decl>`
	`                                <counter value = "0">k</counter>`
	`                        </decl>`
	`                        …`

F.) Script Constructs: WorkFlow Tag

Syntax	<workflow>…</workflow>
Definition	The scope of the workflow tag is that of containing the actual workflow definition of the model, described using classic imperative language constructs.
Rules	• Every script should contain one workflow tag, placed after the global declarations. Workflow tags do not possess XML attributes. • The workflow tag must contain one or more child notes of type <segment>.

Sample Script	<workflow>
	    <segment id = "A Segment">
	        <decl>
	            <counter value = "0">k</counter>
	        </decl>
	        …

G.) Script Constructs: Segment Tag

Syntax	<segment id = "X">…</segment> Where id is Alphanumeric
Definition	The segment tag is responsible for containing the core part of the Transit Script, where the actual workflow resides. The segment tag has an XML attribute named "id" whose value represents the name of the segment. This name is used by <goto> statements in order to call the segment.
Rules	• Segment tags should always be contained in a workflow tag. Multiple segment tags are allowed, however each one must have a unique value in the "id" attribute. • Parameters may be passed to segments from <goto>

	statements. This is done be declaring a local variable inside the segment tag, which has the same name as a parameter which is being passed. The TManager engine will then automatically cater for value mapping. Please note that recursion is not permitted in the Transit Script. • The child structure of a segment should include, primarily any variable declarations, and then a <begin> tag.

Sample Script	`<workflow>`
	    `<segment id = "A Segment">`
	        `<decl>`
	            `<counter value = "0">k</counter>`
	        `</decl>`
	        `<begin>`
	        …
	    `</segment>`

H.) Script Constructs: Begin Tag

Syntax	**<begin>…</begin>**
Definition	The begin tag is the first tag which servers as an indicator point for the parser that the actual workflow definition has begun. From this point onwards, the script takes a more "Procedural 3rd Generation Language" look.
Rules	• There are no strict rules for the content of the begin tag, as long as it contains one of the following tags: <fordo>, <ifthen>, <elseif>, <execute>, <goto>, or <cmd>. • A begin tag should always be used inside a <segment> tag, and should follow and <decl> tags which define local variables. Only one begin tag is allowed per segment.

Sample Script	`<workflow>`
	    `<segment id = "A Segment">`
	        `<decl>`
	            `<counter value = "0">k</counter>`
	        `</decl>`
	        `<begin>`
	            …
	            …

	`</begin>` ``

I.) Script Constructs: For Do Tag

Syntax	**<fordo begin = "A" end = "B" counter = "C" step = "D">…</fordo>** **Where: A,B and C are *n* based expressions** **Where D is either ++ or --**
Definition	The <fordo> tag is similar to the for loop in the C# and Java languages. It contains four attributes in all; the "begin" and "end" attributes indicating the starting and ending value through which to loop, the "counter" indicating the variable used to keep the current value, and the "step" attribute indicating whether the loop is ascending or descending step values.
Rules	• The fordo must always be contained inside a begin statement. • Nesting is allowed, thus a <fordo> can contain another <fordo> • The begin and end attributes may contain variable names which have been locally or globally declared instead of literal values. These are then converted into a natural number the parent segment is called through a <goto> statement, which passes variable values. • The counter attribute's value must be alphanumeric, and must match the name of a locally or globally declared variable. This variable will hold the value of the current loop count. • The step attribute must always contain either ++ for step up, or – for step down loops. • A for do statement can contain the same tags as a <begin> statement.

Sample Script	`<workflow>`
	 `        <segment id = "Start">`

	`<decl>`
	`        <counter value = "0">k</counter>`
	`</decl>`
	`<begin>`
	`        <fordo begin = "paramone"`
	`                end = "paramtwo"`
	`                counter = "k"`
	`                step = "++">`
	`                        …`
	`                        …`
	`                </fordo>`
	`                …`

J.) Script Constructs: If Then and Else If Tags

Syntax	**<ifthen type = "normal" index = "A" result = "B" >…</ifthen>** **Where: A is an *n* based expression** **Where B is "completed/committed/rolledback/compensated"** **OR** **<ifthen type = "expression" expression1 = "A" operator = "B" expression2 = "C">** **Where A and C are *n* based expressions including + or –** **Where B is one of the operators (<, >, <=, >=, ==)**
Definition	The <ifthen> tag is also similar to the if then else loop in the C# and Java languages. However the use of if then statements in the transit model is restricted to two types; those which check the outcome of the execution of an activity, and those which evaluate expressions, as seen in the syntax formats above. The "type" attribute present in the tag has two values, "normal", which indicates that the statement is an expression outcome evaluator, or "expression" which indicates that the statement is an expression evaluator. In the "normal" statement, the "index attribute indicates the position of the Activity in the "activityList", which is under question, while the result indicates the expected outcome. In the "expression" statement, the attributes "expression1" and "expression2" may contain alphanumeric expressions with operators + or -, while the operator attribute may contain a selection of Boolean operators.

Rules	• The <ifthen> must always be contained inside a begin statement. • Nesting is allowed, thus an <ifthen> can contain another <ifthen> • The <ifthen> tag can contain any structure which the <begin> tag or the <fordo> tags contain. (<fordo>, <ifthen>, <elseif>, <cmd>, etc…) • The index attribute in the normal <ifthen>, and the expression attributes in the expression valuator <ifthen> may contain *n* based expressions, or natural numbers. • When an <ifthen> tag closes, it may be immediately followed by an <elseif> tag, which possesses the same attribute properties of the <ifthen> tag, or an <else> tag with no statements, which simply executes the child notes inside if if the <ifthen> or <elseif> statements preceding it fail.

Sample Script	`<workflow>`
	    `<segment id = "Start">`
	        `<decl>`
	            `<counter value = "0">k</counter>`
	        `</decl>`
	        `<begin>`
	            `<ifthen index = "k"`
	                `result = "rolledback"`
	                `type ="normal">`
	                `…`
	                `…`
	            `</ifthen>`
	            `<elseif type = “expression”`
	                `expression1 = “k”`
	                `operator = “<”`
	                `expression2 = “*n*>`
	                `…`
	                `…`
	            `</elseif>`
	            `<else>`
	                `…`
	                `…`
	            `</else>`
	            `…`
	        `</begin>`
	        `…`

K.) Script Constructs: Execute Tag

Syntax	**<execute position = “A” type = “B”>LLT Name</execute>** **Where A is an *n* based expression** **Where B is “complete/commit/rollback/compensate”**
Definition	This construct is the most important construct in the script, since it maps an activity from the LLT provided by the developer, and executes it according to the parameters defined in this statement. The execute statement has two attributes, the position, which indicates the actual position of the activity to process in the activityList, and the type, which defines till what level should the execution proceed.
Rules	• Execute statements can only be used inside a begin tag, inside a segment. • An Activity may be executed several times, progressively, starting from type complete, and moving on to type commit, to type compensate. The same state cannot be executed twice, as this would cause not make sense in a transactional context. The previously explained rules apply, where if an activity commits, it cannot be rolled back, but has to be compensated. • The position of the activity to execute may be expressed either by a natural number, or by an *n* based expression. • An execute statement does not contain child notes, but its inner Text represents the name of the activityList from which Activities are being processed.

Sample Script	`<begin>`
	    `<execute position = "k" type = "commit">`
	        `arrayOfActivities`
	    `</execute>`
	    …
	    …

L.) Script Constructs: Goto Tag

Syntax	**<goto paramone = "A" paramtwo = "B">Segment Name</goto>** **Where A and B are *n* type expressions**
Definition	The <goto> statement has the main task of issuing calls to segments, either from the main program, or from within a segment itself. Unlike the classic "goto" statement in assembly language, this goto does not promote spaghetti code, since it can only issue segment calls, similar to a method call in C# or Java. The <goto> statement can have an indefinite number of elements, which act as parameters in order to pass values to global or local variables. The engine matches the name of the attribute (for example: paramone), to a the name of a variable inside a segment, or a global variable, and propagates the parameter value to it. Thus parmeters may be passed between segments through the <goto> statement.
Rules	• The <goto> statement can only be used inside a <begin> tag, where multiple instances are allowed, or inside the <main> tag, where only one instance is allowed. • The parameter names should match already existent variables which have been globally or locally declared. • The inner text of the command should match a segment which is listed inside a <workflow> tag inside the same script file.
Sample Script	… … <ifthen index = "k" result = "rolledback" type ="normal"> <goto paramone = "k-1" paramtwo = "0">CompensateAll</goto> <cmd>exitscript</cmd> </ifthen> … …

M.) Script Constructs: CMD Tag

Syntax	<cmd>exitscript</cmd>
Definition	This is a simple command which is part of the workflow, and at present contains only one command, which is the "exitscript" command. As soon as this tag is found, its inner text is analysed, and the corresponding command is executed. Plans to extend this tag are classified as future work.
Rules	• <cmd> statements can only be used inside a begin tag, inside a segment. • Since at present, <cmd> has only the "exitscript" command, it can be stated that <cmd> is solely used to exit the script in case a transaction fails, however this may be extended in future versions.
Sample Script	… … <ifthen index = "k" result = "rolledback" type ="normal"> <goto paramone = "k-1" paramtwo = "0">CompensateAll</goto> <cmd>exitscript</cmd> </ifthen> … …

N.) Script Constructs: Main Tag

Syntax	<main>…</main>
Definition	The main tag has the simple scope of containing one <goto> statement, which indicates the first segment which must be called upon initial execution.
Rules	• The <main> can only be used once in a script, and it should be placed as the final child of the model tag, after the <workflow> tag. • The <main> tag is only allowed to have one child of type <goto> which indicates the starting segment, and passes

	initialization parameters.
Sample Script	`<main>` `<goto paramone = "0" paramtwo = "*n*">Start</goto>` `</main>`

2.) Examples

Provided in this section are three transaction models which have been defined using the TransIT Meta Model Script's constructs. These include the Nested Transaction Model, a custom SAGA based model which dynamically implements a try…catch statement, and the Long Lived Transactions Model, originally conceptualized by Ixaris (Malta) Ltd.

A.) The Nested Model

```
<?xml version="1.0" encoding="utf-8" ?>

<model>
        <name>Nested Model</name>

        <decl>
                <activityList size = "*n*">
                        arrayOfActivities
                </activityList>
        </decl>

        <workflow>
                <segment id = "Start">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "paramone"
                                        end = "paramtwo"
                                        counter = "k"
                                        step = "++">

                                        <execute position = "k" type = "complete">
                                                arrayOfActivities
                                        </execute>

                                        <ifthen index = "k" result = "rolledback" type ="normal">
                                                <goto paramone = "k-1"
                                                        paramtwo = "0">

                                                            RollbackAll
                                                </goto>
                        <cmd>exitscript</cmd>
                    </ifthen>
                </fordo>
                <ifthen type = "expression"
                    expression1 = "k"
                    operator = "=="
                    expression2 = "paramtwo">

                        <goto paramone = "paramone"
                            paramtwo = "paramtwo">

                            CommitAll
                        </goto>
                </ifthen>
            </begin>
        </segment>

        <segment id = "RollbackAll">
            <decl>
                <counter value = "0">k</counter>
            </decl>
            <begin>
                <fordo begin = "paramone"
                    end = "paramtwo"
                    counter = "k"
                    step = "--">

                    <execute position = "k" type = "rollback">
                        arrayOfActivities
                    </execute>
                </fordo>
            </begin>
        </segment>

        <segment id = "CommitAll">
            <decl>
                <counter value = "0">k</counter>
            </decl>
            <begin>
                <fordo begin = "paramone"
                    end = "paramtwo"
                    counter = "k"
                    step = "++">
                    <execute position = "k" type = "commit">
                        arrayOfActivities
                    </execute>
                </fordo>
            </begin>
        </segment>
    </workflow>
    <main>
        <goto paramone = "0" paramtwo = "*n*">Start</goto>
    </main>
</model>
```

B.) The JSR 95 Model (Ixaris Implementation)

```
<?xml version="1.0" encoding="utf-8" ?>

<model>
        <name>LLT Model</name>


        <decl>
                <activityList size = "*n*">
                        arrayOfActivities
                </activityList>
        </decl>

        <workflow>
                <segment id = "Start">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "paramone"
                                        end = "paramtwo"
                                        counter = "k"
                                        step = "++">

<execute position = "k" type = "commit">
                                                arrayOfActivities
                                        </execute>

                                        <ifthen index = "k" result = "rolledback" type ="normal">
                                                <goto paramone = "k-1" paramtwo = "0">
                                                        CompensateAll
                                                </goto>
                                                <cmd>exitscript</cmd>
                                        </ifthen>
                                </fordo>
                        </begin>
                </segment>

                <segment id = "CompensateAll">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "paramone"
                                        end = "paramtwo"
                                        counter = "k"
                                        step = "--">

                                        <execute position = "k" type = "compensate">
                                                arrayOfActivities
                                        </execute>
                                </fordo>
                        </begin>
                </segment>
        </workflow>



        <main>
                <goto paramone = "0" paramtwo = "*n*">Start</goto>
        </main>
</model>
```

C.) The Custom SAGA based Model

```
<?xml version="1.0" encoding="utf-8" ?>

<model>
        <name>TryCatch Saga</name>


        <decl>
                <activityList size = "*n*">
                        arrayOfActivities
                </activityList>
        </decl>

        <workflow>
                <segment id = "Try">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "*n*-*n*"
                                        end = "*n*-(*n*-1)"
                                        counter = "k"
                                        step = "++">

                                        <execute position = "k" type = "complete">
                                                arrayOfActivities
                                        </execute>

                                        <ifthen index = "k" result = "rolledback" type ="normal">
                                                <goto param1 = "k-1" param2 = "*n*-*n*">
                                                        Catch
                                                </goto>
                                                <cmd>exitscript</cmd>
                                        </ifthen>
                                </fordo>
                                <ifthen type = "expression"
                                        expression1 = "k"
                                        operator = "=="
                                        expression2 = "*n*-(*n*-1)">
                                        <fordo begin = "*n*-*n*"
                                                end = "*n*"
                                                counter = "k"
                                                step = "++">

                                                <execute position = "k" type = "commit">
                                                        arrayOfActivities
                                                </execute>
                                                <ifthen index = "k"
                                                        result = "rolledback"
                                                        type ="normal">

                                                        <goto param1 = "k-1"
                                                                param2 = "*n*-*n*">

                                                                Finally
                                                        </goto>

                                                        <cmd>exitscript</cmd>
                                                </ifthen>
                                        </fordo>
                                </ifthen>
                        </begin>
                </segment>



                <segment id = "Catch">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "param1"
                                        end = "param2"
                                        counter = "k"
                                        step = "--">

                                        <execute position = "k" type = "rollback">
                                                arrayOfActivities
                                        </execute>
                                </fordo>
                        </begin>
                </segment>

                <segment id = "Finally">
                        <decl>
                                <counter value = "0">k</counter>
                        </decl>
                        <begin>
                                <fordo begin = "param1"
                                        end = "param2"
                                        counter = "k"
                                        step = "--">

                                        <execute position = "k" type = "compensate">
                                                arrayOfActivities
```

```
                                                </execute>
                                        </fordo>
                                </begin>
                        </segment>
                </workflow>



                <main>
                        <goto>Try</goto>
                </main>
</model>
```

The script syntax is so simple that it can be considered as self explanatory. Virtually any form of transaction model can be modeled using this Meta Model, thus making it an ideal candidate for becoming a possible standard.