

# **Contents**

## **Preface** 5

## Introduction 1-1

## **Tutorial**

Import the ::T4X::OBJECTS::* Namespace	2-1
Get and Dump the Default Log Channel	2-2
Load the PLMXML Module	2-3
Take a Snapshot of the Object Server's Cache	2-4
Load a PLMXML File	
Test If an Object Is Implemented by a Given Class	
Find the Header of a Bill of Material (BOM)	2-8
Get Information About the PLMXML File	2-9
Get a Module Handle from an Object	
Get Meta Data of an Object	
Dump the Contents of a BOM Header	2-12
Find the Header of a Bill of Processes (BOP)	2-13
Get the Container Element	2-14
Enable ID Tracing	
Get the Lines of a Bill of Material (BOM)	
Get the First ProductView Element	2-17
Retrieve Attributes	
Revert the Object Cache to a Snapshot	2-19
Get the Master Form from a BOM Header or Line	2-20
Disable Dumping for the Entire PLMXML Module	2-21
Fetch Traced IDs	2-22
Define Procedures to Traverse a Process Tree	
Get the Beginnings for Traversing BOP Lines ————————————————————————————————————	
Get User Data	
Resolve Reference Attributes	
Get the Lines of a Bill of Processes (BOP)	
Get Subordinate Occurrences	
Null Object Handles	2-34
Get the Item Revision from a BOM Header or Line	
Get Form Data	
Get a Form from an Occurrence	
Get the Item from an Item Revision	
Get the PLMXML Equivalent of an Item ID	2-39
Get the Type of an Item Revision	2-40
Get the Value of an ICO Property	2-41

Traverse a Process Tree	— 2-42
Get the Successors for Traversing BOP Lines ————————————————————————————————————	<b>—</b> 2-43
Get the Children of a BOP Header or Line	<b>—</b> 2-45
Get the Process Revision from a BOP Header or Line	2-47
Get the Sequence Number of a BOP Line	_ 2-48
Find the Header of a Bill of Material (BOM) Without Using Shortcuts	<b>2-49</b>
Get the Master Form from an Item Without Using Shortcuts	- 2-51
Get the Type of an Item	<b>—</b> 2-52
Get the Description of a ProcessRevision ————————————————————————————————————	<b>—</b> 2-53

4

## **Preface**

This documentation cannot be used as a substitute for consulting advice, because it can never consider the individual business processes and configuration. Despite our best efforts it is probable that some information about functionality and coherence may be incomplete.

Issue: July 2018

### Legal notice:

All rights reserved. No part of this documentation may be copied by any means or made available to entities or persons other than employees of the licensee of the Teamcenter Gateway or those that have a legitimate right to use this documentation as part of their assignment on behalf of the licensee to enable or support usage of the software for use within the boundaries of the license agreement.

© 2011-2018 Siemens Product Lifecycle Management Software Inc.

#### Trademark notice:

Siemens, the Siemens logo and SIMATIC IT are registered trademarks of Siemens AG.

Camstar and Teamcenter are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries.

Oracle is a registered trademark of Oracle Corporation.

SAP, R/3, SAP S/4HANA®, SAP Business Suite® and mySAP are trademarks or registered trademarks of SAP or its affiliates in Germany and other countries.

TESIS is a registered trademark of TESIS GmbH.

All other trademarks, registered trademarks or service marks belong to their respective holders.

## 1. Introduction

The PLMXML Reader adds to the Teamcenter Gateway the ability to read product data from PLMXML files and to use this data in Teamcenter Gateway mapping templates. It is based on the Object Server, a software system allowing TCL programmers to use functionality implemented in the Java programming language.

#### How to use this Manual

The Tutorial part of this manual is generated from the test\_and\_teach\_plmxml2\_reader.tcl test script that you can find in the var/test/Samples folder in Teamcenter Gateway installations that contain the PLMXML Reader. When we build our software this script is run to test if the PLMXML Reader works as expected. The test script contains code samples that are executed in the test. The very same code samples are also extracted from that test script and put into this manual. So all the code examples you find in this manual are actually proved to work as they should, except those in the *More Examples* sections.

You can use the code examples to write your own test scripts or mapping procedures. You can use the test\_and\_teach\_plmxm12\_reader.tcl to generate a sequence of commands to be copied into your code. Set the **Action to perform** field to **CODE Generate code** and enter into the **Filter** field the name of the Tcl namespace that contains the example you are interested in. The script will print out the code of the example at hand but also the code of the prerequisite examples.

The chapters in the **Tutorial** part can contain the following types of sections:

#### **Prerequisites**

contains a list of links to chapters you should have read before and contain the prerequisite code examples.

#### **Example**

contains the example code in typewriter font and an explanation below in regular font. We ran this code to ensure it works as expected.

#### **More Examples**

contains additional example code. This code does not undergo automatic testing. Look at the comments in the code to see what it does.

#### **Bad Example (DO NOT COPY):**

## 1. Introduction

contains code that is supposed to produce invalid results. This is there to make you aware of situations that may cause unexpected or erroneous behavior. The code is automatically tested to verify that it indeed produces invalid results.

# 2. Tutorial

## 2.1 Import the ::T4X::OBJECTS::\* Namespace

## **Example:**

```
namespace import ::T4X::OBJECTS::*
```

We recommend importing the :: T4X::OBJECTS::\* namespace. This allows you to use the tpmodule, tpwith, tplet commands without having to prefix them with the namespace name. Their names all begin with "tp" so name clashes are unlikely and you know they're not standard Tcl.

## 2.2 Get and Dump the Default Log Channel

## **Prerequisites:**

• Import the ::T4X::OBJECTS::\* Namespace

## **Example:**

```
set defaultLogChannel [::T4X::OBJECTS::getDefaultLogChannel]
puts "defaultLogChannel = $defaultLogChannel"
```

If you need to know the log channel the PLMXML module logs to by default run this snippet of code.

## 2.3 Load the PLMXML Module

## **Prerequisites:**

Import the ::T4X::OBJECTS::\* Namespace

#### **Example:**

set Module [tpmodule de.tesis.plmware.objects.module.plmxml]

#### Caution:

Authors of mappings normally do not have to bother with the loading of modules as the T4x framework loads the required modules before the mapping procedure gets called.

Load the PLMXML module using the tpmodule command. The returned string is an internal name you can use to refer to the module when we go further. Such strings we call object handles. Object handles can refer to any kinds of objects, not just modules.

The PLMXML module is based on the PLMXML SDK provided by Siemens PLM Software. It provides means to load, query, manipulate and save PLMXML files. XML Elements are represented as objects and their attributes can be accessed through so-called methods. We will come to them later on.

© 2018 Siemens Product Lifecycle Management Software, Inc.

## 2.4 Take a Snapshot of the Object Server's Cache

## **Prerequisites:**

Load the PLMXML Module

## **Example:**

set Snapshot [tpsnapshot]

#### Caution:

Authors of mappings normally do not have to bother with the handling of snapshots as snapshots are created before the mapping procedure gets called and reverted after the mapping procedure returns.

The object server maintains a table of all the handles delivered to the client and the objects they represent. This table is called *object cache*.

After processing a document the object cache has to be reverted to the state it had before loading the document. Such a state is called a *snapshot* and the tpsnapshot command can be used to save a snapshot. See Revert the Object Cache to a Snapshot.

There are two reasons why the object cache needs to be reverted:

- 1. In order to ease debugging the handles of objects which represent PLMXML elements and have an id attribute are the values of the id attributes prefixed with "~", e.g. "~id123". Such IDs are unique throughout a document but may have duplicates in other documents. Reverting the object cache removes all such handles so they won't resolve to objects if passed to a method accidentally.
- 2. To free memory resources between transactions.

## 2.5 Load a PLMXML File

### **Prerequisites:**

• Take a Snapshot of the Object Server's Cache

#### **Example:**

```
tplet BomDocument $Module load $::BomTestInputFileFQName
tplet RoutingDocument $Module load $::RoutingTestInputFileFQName
```

#### Caution:

Authors of mappings normally do not have to bother with the loading of PLMXML files as the T4x framework loads the file before the mapping procedure gets called. Only if they need to fetch data from additional files they need to follow the steps described in this section.

As soon as we have the handle of the PLMXML module we can use it to load a PLMXML file.

This is the first time we come across the tplet command. It is used to invoke a so-called *method*. A method provides a piece of functionality e.g. loading a PLMXML file. It has a name e.g. load and can receive arguments (also called parameters), e.g. the name of the BOM file. You see it is quite the same as a standard Tcl command or a Tcl procedure.

The PLMXML Reader is based on the PLMXML SDK which is implemented in an object oriented programming language. So now it is time to learn some terms from object oriented programming.

With Tcl there can be multiple procedures with the same name but different behavior and different argument lists. They then have to be in different namespaces. This is similar with methods. However when talking about methods the term *class* is used instead of *namespace*. So methods can have the same name and arguments but still behave differently because they are defined in different classes. In the object oriented terminology *classes* implement *methods* and *objects* are instances of *classes*.

The class to be used to invoke a method is determined by the object identified by the object handle passed to the tplet command.

The tplet command serves as a bridge between Tcl and the object oriented PLMXML SDK. Its arguments are as follows:

Pos.	Description
1	The name of a variable to receive the method's result
2	The object handle

Pos.	Description
3	The name of the method to invoke
4n	optional arguments to be passed to the method

In most cases methods return a result. Some methods don't. Their return type is called void and the method a void method. In such cases the variable that would receive the method's result is not set.

Some methods return arrays. In such cases the variable becomes a Tcl array.

Here we call the load method twice to parse two PLMXML files and load them into memory. The method receives the file system path of the PLMXML file as an argument and returns a handle for a document object.

If you are writing a procedure in a mapping the document object is passed into the procedure as a parameter. Only if the mapping required you to read a second PLMXML file you would need to call the load method.

If you are writing a test script you need to call the load method.

## 2.6 Test If an Object Is Implemented by a Given Class

## **Prerequisites:**

Load a PLMXML File

### **Example:**

```
tplet isDocument $BomDocument isA Document
tplet isDocumentWithFQClass $BomDocument isA
org.plmxml.sdk.plmxml.plmxml60.Document
tplet isDocumentBase $BomDocument isA DocumentBase
tplet isOccurrence $BomDocument isA Occurrence
```

As pointed out in Load a PLMXML File methods are implemented by classes and objects are instances of classes. A class defines the sets of methods one can invoke on an object that is an instance of that class. One class does not have to implement all these methods on its own. Instead it can inherit method implementations from another class which then acts as a so called super-class. Such classes in turn can inherit methods from other super classes building up hierarchies of classes. In object oriented terminology a class that inherits methods from a super class "extends" the super class.

Let's assume there is an object which is an instance of the ProcessOccurrence class. Because the ProcessOccurrence class extends the Occurrence class you can also look at it as an Occurrence. We say the object "is a" Occurrence or "is a" ProcessOccurrence.

You could also say the object is of Occurrence type as well as of ProcessOccurrence type. Often the terms *class* and *type* can be used interchangeably. You do not have to bother with the differences between these terms.

In order to test whether a class or its super classes contributes methods to an object, we can use the isA method.

It is called with an object handle and the name of a class and returns true if the object is implemented by the given class or false otherwise.

Note that you can only call the isA method on PLMXML documents and objects that may be contained in them.

## 2.7 Find the Header of a Bill of Material (BOM)

## **Prerequisites:**

Load a PLMXML File

### **Example:**

```
tplet BomHeader $BomDocument findHeader "MEProductContext"
if {$BomHeader == "null"} {
  error "Could not find the BOM Header"
}
```

We already mentioned extensions. Some of these implement methods we call *shortcuts*. They are just there to provide a shorter and more convenient way to achieve a task than would be possible with the SDK alone.

The findHeader method is a shortcut to navigate to the root of a structure made up of occurrence elements. There are several flavors of occurrences for Bill Of Material structures, Plant structures, Bill of Process structures etc. The contextType argument of the findHeader method is matched against the subType attribute of a StructureContext element in the course of searching the occurrence that represents the header of the structure:

contextType	type of stucture
MEProductContext	Bill of Material
MEPlantContext	Plant
MEProcessContext	Bill of Processes

If the findHeader shortcut is not able to find the header of the BOM structure it returns the special null object handle. All methods that return object handles will return the null object handle to indicate the fact that there is no such object.

## 2.8 Get Information About the PLMXML File

## **Prerequisites:**

Load a PLMXML File

## **Example:**

```
tplet BomFile $BomDocument getFile
tplet FileName $BomFile getName
```

As mentioned already the PLMXML module is based on the PLMXML SDK. Unfortunately the classes that make up the PLMXML SDK do not always provide enough functionality or provide it in an inconvenient way.

The PLMXML Reader therefore added methods to the SDK through so called extensions.

The method getFile has been added to the SDK's Document class through an extension. It returns a File object which can then be further queried.

## 2.9 Get a Module Handle from an Object

## **Prerequisites:**

Load a PLMXML File

## **Example:**

tplet HomeModule \$BomDocument getHomeModule

Every object provides the method named <code>getHomeModule</code>. You can call this method through the <code>tplet</code> command to obtain a handle to the module object, the same object returned from the <code>tpmodule</code> command (see Load the PLMXML Module). This allows you to call methods of the module even if the handle returned from <code>tpmodule</code> is not in the current scope.

## 2.10 Get Meta Data of an Object

## **Prerequisites:**

Load a PLMXML File

## **Example:**

```
tpdescribe $BomDocument Description
```

If you want to know which methods an object provides and by which class it is implemented you can call the tpdescribe command with the handle of the object in question and the name of an array variable that will receive the result.

The keys of the array map to the following pieces of information:

Key	Maps to
type.name	Name of the implementing java class
method <n> java</n>	Prototype of the method for Java programmers
method <n> example</n>	Example usage of the method for mapping authors

In the above list <N> stands for a non-negative integer number enumerating the methods, starting with so-called *extensions*. You will learn more on these later on.

## **More Examples:**

```
# You can iterate through the tcl examples using
set keys [lsort -integer -index 1 [array names Description "method *
example"]]
foreach key $keys {
  puts $Description($key)
}
```

## 2.11 Dump the Contents of a BOM Header

## **Prerequisites:**

• Find the Header of a Bill of Material (BOM)

### **Example:**

```
tpwith $BomHeader dump
tplet dumpOutput $BomHeader dumpToString
tplet dumpOutputHeaderOnly $BomHeader dumpToString -maxLevel 0
```

This is the first time we use the tpwith command. It is similar to the tplet command in that it invokes methods on the object server. It does however not set a variable. Instead it returns the result of the method call. For methods that don't return a value (void methods) an empty string is returned. For methods that return an array a list of key value pairs is returned to be used as input to the array set command.

Here we call the dump method using the tpwith command.

Every object supports this method. It can be used to send the object's state to the log server. It does not return anything.

There is also the dumpToString method that returns the characters that dump would send to the log server as a string.

Both methods support the following options:

```
-maxLevel <N>
```

When recursively collecting information from an object the depth of the recursion can be limited by setting this option to a positive integer value. If this option is not set the depth is unlimited. The arguments <code>-maxLevel 0</code> mean to dump just this object, the BOM header in this case.

-traceIds

Normally the dump methods do not trace IDs even if ID tracing is enabled. If this flag is set then they do trace IDs. See **Enable ID Tracing** for more on ID tracing.

## 2.12 Find the Header of a Bill of Processes (BOP)

## **Prerequisites:**

- Load a PLMXML File
- Find the Header of a Bill of Material (BOM)

## **Example:**

tplet BopHeader \$RoutingDocument findHeader "MEProcessContext"

We already navigated to the BOMHeader using the findHeader method (see Find the Header of a Bill of Material (BOM)). Here we use the same method to navigate to the root of a structure made up of ProcessOccurrence elements.

## 2.13 Get the Container Element

## **Prerequisites:**

• Find the Header of a Bill of Material (BOM)

## **Example:**

tplet AProductView \$BomHeader getOwner

Some methods provided by the PLMXML SDK are defined for all types of objects. They are called *generic methods* and normally used to deal with aspects of XML be it the PLMXML dialect or something else.

One such method is the getOwner method which returns an object handle representing the element which contains the element on which the method was called. We avoid the term parent as it is used in the context of Occurrence elements and has a different meaning there.

## 2.14 Enable ID Tracing

## **Prerequisites:**

- Get a Module Handle from an Object
- Dump the Contents of a BOM Header

### **Example:**

tpwith \$HomeModule enableIdTracing

There is another feature to help you debugging your mapping code.

Let's say you are looking at the input PLMXML file with some other tool e.g. an XML editor or a graphical visualization tool. In order to reduce the number of displayed elements you need a list of all the PLMXML elements that are affected by your mapping code.

This is what ID tracing does. Once enabled it records the IDs of the PLMXML elements returned or processed by the methods you call.

#### Caution:

Note that ID tracing is a debugging tool and you must never use the trace information in your mapping code!

This warning is issued because it's up to the methods to decide which elements should be traced and which not and this may change between versions of the software.

## 2.15 Get the Lines of a Bill of Material (BOM)

## **Prerequisites:**

• Find the Header of a Bill of Material (BOM)

### **Example:**

```
tplet BomLines $BomHeader getBomLines
tplet AllBomLines $BomHeader getBomLines -maxLevel infinite
```

In order to get the children of a BOM header or BOM line use the getBomLines method.

By default this method returns a list of handles representing the direct children of the BOM header or BOM line.

With the option

```
-maxLevel <N>
```

you can control how often this method is recursively applied collecting also the children of the children and so on.

Valid values for N are:

```
< N > = 1
```

collect the direct children of the BOM header or BOM line. They are returned in the order of their IDs as listed in the occurrences attribute.

```
< N > > 1
```

collect the direct children and their children etc. down to a depth of N

```
<N> = "infinite"
```

collect the direct children and their children etc. with unlimited depth of recursion

Note that for  $<\mathbb{N}>>1$  you should not rely on the order of the elements.

## 2.16 Get the First ProductView Element

## **Prerequisites:**

- Load a PLMXML File
- Get the Container Element

#### **Example:**

tplet ProductViews \$BomDocument getElementsByClass ProductView
set FirstProductView [lindex \$ProductViews 0]

Other generic methods the PLMXML SDK provides are <code>getElements</code> and <code>getElementsByClass</code>. The first one returns a list of object handles representing all the elements contained in the element on which the method was called. You may consider it the reverse operation of the <code>getOwner</code> method (see <code>Get the Container Element</code>).

The getElementsByClass method differs from getElements in that it filters the contained elements by their class which is given as the first and only argument.

## 2.17 Retrieve Attributes

## **Prerequisites:**

• Find the Header of a Bill of Material (BOM)

## **Example:**

tplet id \$BomHeader getId

PLMXML files carry most of the product data as XML attributes. Seldom product data appears as the contents of XML elements. In both cases the way to read out this kind of information you use methods the names of which start with get followed by the name of the attribute with the first character turned int upper case. E.g. to get the id of an object you would call its getId method.

## 2.18 Revert the Object Cache to a Snapshot

### **Prerequisites:**

- Take a Snapshot of the Object Server's Cache
- Dump the Contents of a BOM Header

## **Example:**

```
# We can still call a method on the snapshot object
tpwith $Snapshot toString
# We revert the object cache to a point in time before the
# snapshot was created
tprevertto $Snapshot
# We can no longer call a method on the snapshot object
# because it's gone
set caughtError [catch {tpwith $Snapshot toString}]
# We can still call a method on the module object
tpwith $Module toString
```

#### Caution:

Authors of mappings normally do not have to bother with the handling of snapshots as snapshots are created before the mapping procedures get called and reverted after the mapping procedures return.

If a snapshot of the object cache has been created using the tpsnapshot command the tprevertto command can be used to restore the object cache's state to a point in time just before the snapshot was created. This also means that the snapshot handle becomes invalid by calling the tprevertto command.

For more on snapshots see Take a Snapshot of the Object Server's Cache.

## 2.19 Get the Master Form from a BOM Header or Line

## **Prerequisites:**

• Find the Header of a Bill of Material (BOM)

## **Example:**

tplet MasterForm \$BomHeader getMasterForm

In order to navigate to the form associated with an item the PLMXML Reader provides the getMasterForm shortcut. It expects the handle of the BOM header or BOM line as its first and only argument and returns the handle of a Form element. Note that in PLMXML you don't start at an item but at a BOM header or BOM line.

## 2.20 Disable Dumping for the Entire PLMXML Module

## **Prerequisites:**

- Get a Module Handle from an Object
- Dump the Contents of a BOM Header

#### **Example:**

```
# Disable dumping
tpwith $HomeModule disableDumping
# These do not dump
tpwith $HomeModule dump
tplet dumpOutput1 $HomeModule dumpToString
# Enable dumping again
tpwith $HomeModule enableDumping
# These do dump
tpwith $HomeModule dump
tpwith $HomeModule dump
```

The methods <code>dumpToString</code> are there to debug mappings in the course of their development. The calls to these methods should be removed or commented out as soon as the mapping is production ready or if you want to execute performance tests.

Alternatively these methods can be put out of operation by disabling dumping for the entire PLMXML module.

## 2.21 Fetch Traced IDs

## **Prerequisites:**

• Enable ID Tracing

## **Example:**

tplet TracedIds \$HomeModule fetchTracedIds

You have already learned that with ID tracing you can collect the IDs of the element affected by your mapping code. If ID tracing has been enabled using the PLMXML module's <code>enableIdTracing</code> method (see <code>Enable ID Tracing</code>) you can retrieve the IDs of the affected objects using the module's <code>fetchTracedIds</code> method. It returns a list of the affected IDs. Note that the returned list is in no particular order.

## 2.22 Define Procedures to Traverse a Process Tree

## **Prerequisites:**

Find the Header of a Bill of Processes (BOP)

## **Example:**

```
# Recursivly traverse the whole process tree
# @param Start element to start from
# @param Order order by which to traverse the tree
# @ResultVar Name of an array to collect the results
proc TraverseProcessTree {Start Order ResultVar} {
  upvar $ResultVar Result
  # Number of successors already visited
 set Result(joins) 0
  # Number of all visited elements
 set Result(all) 0
  # Number of elements at which a left to right traversal begins
 set Result(beginnings) 0
  # Number of elements not ordered by the given order
 set Result (unordered) 0
 RecursivelyTraverseTopToBottom $Start $Order 0 Result
proc RecursivelyTraverseTopToBottom {BopElement Order Depth \
 ResultVar} {
 upvar $ResultVar Result
  # Do something with the BOP element. Here we just count it.
 incr Result(all)
 set NewDepth [expr {$Depth + 1}]
  # Get leftmost elements e.g. those with no
  # predecessors or those with the lowest sequence
  # number. May be empty if children are not sorted by
  # the given order.
  tplet Beginnings $BopElement getBeginnings -by $Order
  # Foreach leftmost element start a new horizontal
  # traversal. Such traversals may cover the same
  # elements if they have elements that share the same
  # successors
  foreach Beginning $Beginnings {
    RecursivelyTraverseLeftToRight $Beginning $Order $NewDepth \
     Result
  # Get children not covered by 'getBeginnings'
  tplet UnorderedChildren $BopElement getChildren \
    -excludeOrderedBy [lindex $Order 0]
```

```
foreach Child $UnorderedChildren {
    RecursivelyTraverseTopToBottom $Child $Order $NewDepth Result
}
proc RecursivelyTraverseLeftToRight {BopElement Order Depth \
 ResultVar} {
 upvar $ResultVar Result
  # Look at the successor relation depicted below (you
  # get the predecessors by inverting the direction of
  # the arrows):
  #
                               | id13 |
                           --> |
                                        --> |
    | id11 | --> | id12 |
                                           | id15 | --> | id16 |
                               | id14 |
  #
  # You can easily imagine that in the course of
  # traversing left to right it might happen that a
  # 'ProcessOccurrence' element gets visited more than
  # once. In the picture above these would be the one
  # with id="id15" and all that follow. We call this a
  # "join".
  # Here we count such joins and immediately return so
  # to not process 'ProcessOccurrence' elements more
  # than once.
  if {[info exists Result([list visitedOrdered $BopElement])]} {
   incr Result(joins)
    ######
   return
    ######
  set Result([list visitedOrdered $BopElement]) {}
 RecursivelyTraverseTopToBottom $BopElement $Order $Depth Result
 tplet Successors $BopElement getSuccessors -by $Order
  foreach Successor $Successors {
    RecursivelyTraverseLeftToRight $Successor $Order $Depth Result
  }
}
```

Traversing a tree of ProcessOccurrence elements can be complicated if you want to take sequence numbers or predecessor relations into account to determine the order of traversal.

This code example provides you with definitions of procedures that traverses the whole tree either in sequencenumbers or references order falling back to container order if the other two are not available.

See Traverse a Process Tree for how to apply them.

## 2.23 Get the Beginnings for Traversing BOP Lines

## **Prerequisites:**

• Find the Header of a Bill of Processes (BOP)

### **Example:**

```
# BOP header has no child with a 'predecessorRefs' attribute so we get
# an empty list.
tplet BeginningsByRefs1 $BopHeader getBeginnings -by references
# BOP header has one child with a 'SequenceNumber' user value so we get
# a list with one handle
tplet BeginningsBySeqNo1 $BopHeader getBeginnings -by sequencenumbers
# BOP header has one child so we get a list with one handle
tplet BeginningsByContainer1 $BopHeader getBeginnings -by container
# Drill down into the first child.
# On the next level children have 'predecessorRefs' attributes and one is
# identified as the beginning because it is not referenced by other
children.
tplet BeginningsByRefs2 $BeginningsByContainer1 getBeginnings -by
references
tplet BeginningsBySeqNo2 $BeginningsByContainer1 getBeginnings -by
sequencenumbers
tplet BeginningsByContainer2 $BeginningsByContainer1 getBeginnings -by
container
```

BOP headers as well as BOP lines are represented by ProcessOccurrence elements. These can be ordered by one or more of the following systems:

#### container

The BOP lines are ordered according to their appearance in the parent's occurrence attribute

### sequencenumbers

The BOP lines are ordered according to their sequence numbers. These are stored as <code>UserValue</code> elements titled <code>SequenceNumber</code>

#### references

The BOP lines are ordered according to their predecessorRefs attributes

In order to traverse a tree of ProcessOccurrence elements you ask it for the first child in the given order and then ask that child for its successor. **THATS NOT ENTIRELY TRUE, PLEASE READ ON**.

At least for the container system this is the truth. For sequencenumbers and references things can be a bit more complicated. There can be multiple children with the same sequence number and along the same lines a ProcessOccurrence can have multiple predecessors.

So the whole truth reads as follows:

In order to traverse a tree of ProcessOccurrence elements you ask it for the leftmost children in the given order and then ask those children for their successors.

The method to get the leftmost children is called <code>getBeginnings</code>. Notice the plural form of the name. In general it returns a list of object handles which can be different according to the options passed to the method. These are the options it accepts:

-by <LIST OF ORDER SYSTEMS>

The given list must not be empty. Its first element determines according to which system, i.e. container, sequencenumbers or references, the leftmost children are retrieved from the PLMXML document. The next elements are then used to sort the resulting list.

For container order the returned list normally contains one element. Only in the case that the ProcessOccurrence does not have any children the returned list is empty. For sequencenumbers order or references order the list can contain more than one element.

-by references is the default for this option

-includeOrderedBy <LIST OF ORDER SYSTEMS>

Filter the resulting list of object handles by the given order systems. Only return children ordered by the given order system. You probably will never use this option with this method but with the getChildren method.

-excludeOrderedBy <LIST OF ORDER SYSTEMS>

Filter the resulting list of object handles by the given order systems. Only return children **not** ordered by the given order system. You probably will never use this option with this method but with the getChildren method.

## 2.24 Get User Data

## **Prerequisites:**

- Get the Lines of a Bill of Material (BOM)
- Find the Header of a Bill of Material (BOM)

#### **Example:**

```
foreach bomLine $BomLines {
    set bomLinesBySeqNo([tpwith $bomLine getUserDataValue SequenceNumber])
$bomLine
}
unset -nocomplain dummy
tplet dummy $BomHeader getUserDataValue FooBar
tplet myDefault $BomHeader getUserDataValue -default "Hello World" FooBar
```

In order to associate arbitrary data with elements applications can add UserData elements to them. "Arbitrary data" means attributes whose names and types are not specified in the PLMXML schemata, e.g. customer specific data.

You can imagine UserData elements as tables with three columns:

- 1. title the names of the attributes
- 2. value their values
- 3. type optional types which can be used to classify attributes

The rows of the table are represented as <code>UserValue</code> elements. They carry the three attributes <code>title</code>, <code>value</code> and <code>type</code>.

Also the UserData elements have an optional type attribute which can be used to classify the whole sets of attributes they represent.

#### Caution:

Note that UserData is the name of the PLMXML element and should not be confused with the user\_data\_1, user\_data\_2, etc. attributes defined in the Teamcenter data model.

In order to retrieve a value from a form or any other element containing UserData elements use the getUserDataValue shortcut giving the name of the attribute.

#### It accepts the following options:

```
-default <DEFAULT>
```

If the attribute could not be found return <DEFAULT>. If no default is given and this shortcut is called with the tplet command the target variable remains unset.

```
-dataType <DATA TYPE>
```

When selecting attributes only consider UserData elements whose type attribute is <DATA TYPE>.

```
-valueType <VALUE_TYPE>
```

When selecting attributes only consider UserValue elements whose type attribute is <VALUE TYPE>.

#### Caution:

NOTE that if an attribute could not be found and no default was given with -default getUserDataValue behaves as follows:

- If called with tplet <TARGET> <HANDLE> getUserDataValue ... and TARGET did not exist before TARGET remains unset.
- If called with tplet <TARGET> getUserDataValue ... and <TARGET> did exist before an error is raised.
- If called with tpwith <HANDLE> getUserDataValue ... the empty string is returned.

You can check whether an attribute exists or not using the hasUserDataValue method. It accepts the same arguments and options as getUserDataValue.

# 2.25 Resolve Reference Attributes

#### **Prerequisites:**

- Find the Header of a Bill of Material (BOM)
- Get the First ProductView Element
- Retrieve Attributes

#### **Example:**

```
# Single URI reference
tplet ItemRevisionURI $BomHeader getInstancedURI
tplet ItemRevision $BomHeader resolveInstancedURI
# Multiple URI reference
tplet AssociatedAttachments $BomHeader resolveAssociatedAttachmentURIs
# Single NCName reference
tplet primaryOccurrence $FirstProductView resolvePrimaryOccurrenceRef
tplet primaryOccurrenceId $FirstProductView getPrimaryOccurrenceId
# Multiple NCName reference
tplet BomChildIds $BomHeader getOccurrenceIds
tplet BomChildren $BomHeader resolveOccurrenceRefs
```

The PLMXML dialect makes heavy use of references. Elements are identified by their id attributes and can themselves refer to other elements through reference attributes. They contain the IDs of the other elements. All id and reference attributes are just character strings of a specific format. For example they must not contain space characters.

In order to get to a referenced element the reference has to be resolved. The PLMXML SDK provides a set of methods to do that. Their names start with resolve.

There are four different kinds of references and the methods provided by the PLMXML SDK are named accordingly:

	Kind	Method Name
1	Single URI references	resolve <attr>URI</attr>
2	Multiple URI references	resolve <attr>URIs</attr>
3	Single NCName references	resolve <attr>Ref</attr>
4	Multiple NCName references	resolve <attr>Refs</attr>

In the above table <ATTR> stands for the name of the attribute containing the reference.

The plural forms, those whose names end with the letter s, return space separated lists of references to be iterated over using Tcl's foreach command. The PLMXML SDK also provides methods to get the number of elements in such a list and to address their members by a zero-based integer index so you could iterate over them using a for loop with an integer index. However with respect to the readability and performance of the code we strongly recommend that you use the resolve methods listed above.

Actually the resolve<ATTR>URIs kind of methods are not part of the PLMXML SDK but the PLMXML Reader added those methods through extensions.

You can also retrieve the reference attributes without resolving them, but you will probably never need this. Just replace resolve with get and Ref with Id in the names of the methods, e.g. resolveOccurrenceRefs becomes getOccurrenceIds and resolveInstanceURI becomes getInstanceURI.

"URI" stands for uniform resource identifier, "NCName" for non-colonized name. In the PLMXML context they are just strings that allow one element in an PLMXML file to address another element in an PLMXML file. The URI and NCName kind of references differ in their scope. The URI references may reference elements in other documents. Most references are of the URI kind but only seldom they cross file boundaries. Their format is <DOC>#<ID> where <DOC> identifies a document and <ID> the element within the document. The <DOC> part may be missing meaning that the reference resolves to an element within the same document.

NCName references are more limited in their scope. They can only reference elements within the same document. Their format is <ID> without the # symbol. IDs consist of a number of characters out of a set of allowed characters. Most often they appear as the string id followed by a positive integer number.

# 2.26 Get the Lines of a Bill of Processes (BOP)

#### **Prerequisites:**

- Find the Header of a Bill of Processes (BOP)
- Get the Lines of a Bill of Material (BOM)

#### **Example:**

```
tplet BopLines1 $BopHeader getBopLines
tplet BopLines2 $BopHeader getBopLines -maxLevel 1
tplet BopLines3 $BopHeader getBopLines -maxLevel 2
tplet BopLines $BopHeader getBopLines -maxLevel infinite
```

Like getBomLines returns the children of a BOM header or BOM line the getBopLines method returns the children of a BOP header or BOP line. It accepts the same arguments and options.

### 2.27 Get Subordinate Occurrences

#### **Prerequisites:**

- Find the Header of a Bill of Processes (BOP)
- Get the Lines of a Bill of Processes (BOP)

#### **Example:**

```
tplet SubOccurrences1 $BopHeader getSubOccurrences
tplet SubOccurrences2 $BopHeader getSubOccurrences -maxLevel 1
tplet SubOccurrences3 $BopHeader getSubOccurrences -maxLevel 2
tplet SubOccurrences $BopHeader getSubOccurrences -maxLevel infinite
```

You already know the getBomLines and getBopLines shortcuts to get the children of a BOM header or a BOM line. The getSubOccurrences does the same but under a more general name. It can be used for any kinds of Occurrence elements be them ProductOccurrence elements, ProcessOccurence elements, etc.

# 2.28 Null Object Handles

# **Prerequisites:**

- Load a PLMXML File
- Find the Header of a Bill of Material (BOM)
- Retrieve Attributes

# **Bad Example (DO NOT COPY):**

```
tplet FooHeader $BomDocument findHeader "MEFooContext"
tplet fooId $FooHeader getId
```

The code above calls the findHeader method with a context type that does not exist. The method will indicate this fact by returning the null handle. Its Tcl representation is the string null.

# 2.29 Get the Item Revision from a BOM Header or Line

# **Prerequisites:**

- Find the Header of a Bill of Material (BOM)
- Resolve Reference Attributes

#### **Example:**

tplet ItemRevision \$BomHeader resolveInstancedURI

From a BOM header or BOM line you can navigate to the item revision by resolving the <code>instancedRef</code> reference. Note that <code>instancedRef</code> is the name of the attribute in the PLMXML file but <code>resolveInstancedURI</code> is the name of the method.

# 2.30 Get Form Data

# **Prerequisites:**

- Get the Master Form from a BOM Header or Line
- Get User Data

# **Example:**

```
tplet LastReleaseStatus $MasterForm getUserDataValue \
   LastReleaseStatus -dataType FormAttributes -default ""
```

As soon as you have the handle of a Form element you can retrieve its attributes using the getUserDataValue shortcut. See Get User Data for more.

# 2.31 Get a Form from an Occurrence

#### **Prerequisites:**

Get the Master Form from a BOM Header or Line

#### **Example:**

```
tplet Form $PlantTree getForm -subType "MEPlantRevision Master"
IMAN master form
```

You already know the getMasterForm shortcut to navigate to the form associated with an item. There is a more general shortcut which allows you to search for other kinds of forms starting from an Occurrence element.

It is called getForm and expects one argument:

#### relationNames

list of relations ('IMAN\_master\_form' in the example) to follow when searching for the form. Relation names are matched against the role attribute of the AssociatedAttachment element which leads to the form.

With the <code>-subType</code> <code><SUBTYPE></code> option you can restrict the search to forms for which the <code>subType</code> attribute matches the string given in <code><SUBTYPE></code>. On the same lines you can use the <code>-subClass</code> <code><SUBCLASS></code> option. It limits the search to forms for which the <code>subClass</code> attribute matches the string given in <code><SUBCLASS></code>.

#### 2.32 Get the Item from an Item Revision

#### **Prerequisites:**

• Get the Item Revision from a BOM Header or Line

# **Example:**

tplet Item \$ItemRevision resolveMasterURI

From an item revision you can navigate to the item by resolving the masterRef reference. Note that masterRef is the name of the attribute in the PLMXML file but resolveMasterURI is the name of the method.

You probably noticed that the names used in PLMXML are different from those you know from Teamcenter data models, e.g. an Item is a Product, an ItemRevision a ProductRevision and the attribute leading to the item is masterRef as opposed to itemRef.

# 2.33 Get the PLMXML Equivalent of an Item ID

#### **Prerequisites:**

• Get the Item from an Item Revision

# **Example:**

tplet ItemId \$Item getProductId

In order to get the item's id often mapped to the material number on the SAP side use the getProductId method.

#### Caution:

Do not confuse the item's id with the id attribute of a Product element. The id attribute is only valid within the PLMXML document and you normally do not have to deal with it. You find the item's id in the productId attribute.

# 2.34 Get the Type of an Item Revision

# **Prerequisites:**

- Get the Item Revision from a BOM Header or Line
- Retrieve Attributes

#### **Example:**

```
tplet ItemRevisionType $ItemRevision getSubType
```

In order to get the type of an item revision use the getSubType method.

Note that the method is called getSubType as opposed to getType. This is because

- 1. this is the name of the PLMXML attribute
- 2. in order to not confuse it with the type of the PLMXML element which would be ProductRevision

# 2.35 Get the Value of an ICO Property

#### **Prerequisites:**

- Get the Item from an Item Revision
- Resolve Reference Attributes

#### **Example:**

```
set result {}
tplet Classifications $Item getElementsByClass plmxml cl:Classification
foreach Classification $Classifications {
  tplet Icos $Classification resolveIcoURIs
  foreach Ico $Icos {
    tplet Properties $Ico getProperties
    foreach Property $Properties {
      tplet AttributeId $Property getAttributeId
      puts "AttributeId = $AttributeId"
      tplet Values $Property getValues
      foreach Value $Values {
        tplet Content $Value getElementContent
        puts "Content = $Content"
        lappend result $AttributeId $Content
    }
  }
}
```

How to obtain the text value of a plmxml\_cl:Value element (where plmxml\_cl is the prefix of the http://www.plmxml.org/Schemas/PLMXMLClassificationSchema namespace) is not immediately obvious. You need to iterate over the ICOProperty objects returned by the getProperties method. Ask the ICOProperty elements for their attribute id and their values. The actual value, i.e. the text contained in the text node, is returned from the getElementContent method.

#### 2.36 Traverse a Process Tree

#### **Prerequisites:**

• Define Procedures to Traverse a Process Tree

# **Example:**

```
catch {array unset Result1}
catch {array unset Result2}
catch {array unset Result3}
catch {array unset Result4}
TraverseProcessTree $BopHeader {references} Result1
TraverseProcessTree $BopHeader {references sequencenumbers} Result2
TraverseProcessTree $BopHeader {sequencenumbers} Result3
TraverseProcessTree $BopHeader {sequencenumbers references} Result4
```

Here you see how to apply the procedures defined in **Define Procedures to Traverse a Process Tree**.

# 2.37 Get the Successors for Traversing BOP Lines

#### **Prerequisites:**

Get the Beginnings for Traversing BOP Lines

#### **Example:**

```
# Move down one level
tplet successors $BeginningsByContainer2 getBeginnings -by container
set totalCount 0
set lengthsByRefs(0) 0
set lengthsByRefs(1) 0
set lengthsByRefs(2) 0
set lengthsBySeqNos(0) 0
set lengthsBySeqNos(1) 0
set lengthsBySeqNos(2) 0
set unequalOrNullSeqNosCount 0
# Iterate in container order
while {[llength $successors]} {
 incr totalCount
 tplet successorsByRefs $successors getSuccessors -by references
  tplet successorsBySeqNos $successors getSuccessors -by sequencenumbers
  # Some BOM elements have more than one successor in reference
  # or sequence number order. We count them here for verification.
 incr lengthsByRefs([llength $successorsByRefs])
 incr lengthsBySeqNos([llength $successorsBySeqNos])
  # All elements returned by 'getSuccessors -by sequencenumbers'
  # are supposed to have a SequenceNumber and it is the same for
  # all
 if {![allHaveEqualNonNullSequenceNumbers $successorsBySeqNos]} {
    incr unequalOrNullSeqNosCount
  # Prepare next iteration
 tplet successors $successors getSuccessors -by container
}
```

Remember how to traverse a tree of ProcessOccurrence elements:

In order to traverse a tree of ProcessOccurrence elements you ask it for the leftmost children in the given order and then ask those children for their successors.

The method to get a ProcessOccurence's successors is called getSuccessors it accepts exactly the same otions as the getBeginningsMethod. You will normally use these methods with the same options.

#### Caution:

NOTE that you will not find successorRefs attributes in the PLMXML document. The successor relations are computed from the predecessorRefs attributes.

# 2.38 Get the Children of a BOP Header or Line

#### **Prerequisites:**

- Find the Header of a Bill of Processes (BOP)
- Get the Beginnings for Traversing BOP Lines
- Get the Successors for Traversing BOP Lines

#### **Example:**

```
tplet children1 $BopHeader getChildren
set onlyChild1 [lindex $children1 0]
tplet children2 $onlyChild1 getChildren
set firstChild2 [lindex $children2 0]
tplet children3 $firstChild2 getChildren
tplet children3NoRefs $firstChild2 getChildren -excludeOrderedBy
references
tplet children3NoSeqNos $firstChild2 getChildren -excludeOrderedBy
sequencenumbers
```

You have already learned how to traverse a tree of ProcessOccurrence elements using the getBeginnings and getSuccessors methods.

There is another less fine grained way to do so using the <code>getChildren</code> method. This method returns the children of a <code>ProcessOccurrence</code> as a single list of object handles. It accepts the same options as the <code>getBeginnings</code> and <code>getSuccessors</code> methods but their meanings are slightly different for this method:

```
-by <LIST OF ORDER SYSTEMS>
```

The given list must not be empty. Its elements determine according to which systems, i.e. container, sequencenumbers or references, the children are sorted before they are returned. The result list is first sorted by the first order system given in the list. Children which are equal according to this order system, e.g. have the same sequence number for the sequencenumbers order system, are then further sorted by the second element of <LIST OF ORDER SYSTEMS>. -by container is the default for this option

```
-includeOrderedBy <LIST OF ORDER SYSTEMS>
```

Filter the resulting list of object handles by the given order systems. Only return children ordered by the given order system.

-excludeOrderedBy <LIST OF ORDER SYSTEMS>

Filter the resulting list of object handles by the given order systems. Only return children **not** ordered by the given order system.

# 2.39 Get the Process Revision from a BOP Header or Line

# **Prerequisites:**

- Find the Header of a Bill of Processes (BOP)
- Resolve Reference Attributes

#### **Example:**

tplet ProcessRevision \$BopHeader resolveInstancedURI

Like with item revisions you can navigate from a BOP header or BOP line to the process revision by resolving the <code>instancedRef</code> reference. Note that <code>instancedRef</code> is the name of the attribute in the PLMXML file but <code>resolveInstancedURI</code> is the name of the method.

# 2.40 Get the Sequence Number of a BOP Line

#### **Prerequisites:**

Get the Lines of a Bill of Processes (BOP)

#### **Example:**

```
set firstChild [lindex [tpwith $BopHeader getBopLines] 0]
tplet BopHeaderHasSequenceNumber $BopHeader isOrderedBySequenceNumbers
tplet BopHeaderSequenceNumber $BopHeader getSequenceNumber
tplet FirstChildHasSequenceNumber $firstChild isOrderedBySequenceNumbers
tplet FirstChildSequenceNumber $firstChild getSequenceNumber
```

BOP lines are represented by ProcessOccurrence elements. One way to define an order among them are sequence numbers. These are stored as UserValue elements titled SequenceNumber. The getSequenceNumber shortcut returns the value of the SequenceNumber attribute or the string NONE if no sequence number is defined for the BOP line, i.e. the SequenceNumber attribute does not exist or its value is the empty string.

You can check if a BOP line has a sequence number by calling the isOrderedBySequenceNumbers method. It returns true if the BOP line has a sequence number, false otherwise.

# 2.41 Find the Header of a Bill of Material (BOM) Without Using Shortcuts

#### **Prerequisites:**

- Find the Header of a Bill of Material (BOM)
- Test If an Object Is Implemented by a Given Class
- Resolve Reference Attributes

#### **Example:**

```
proc FindHeader {document contextType} {
 variable UseForeachShortCut
 tplet header $document getHeader 0
  tplet objects $header resolveTraverseRootURIs
  foreach object $objects {
    set result [RecursivelyFindHeader $object $contextType]
    if {$result != "null"} {
     return $result
 return null;
proc RecursivelyFindHeader {object contextType} {
 if {![tpwith $object isA AssociatedAttachment]} {
    return null;
  }
  set aa $object
  tplet object $aa resolveAttachmentURI
  if {[tpwith $object isA Managed]} {
    set context $object
    tplet subType $context getSubType
    if {$contextType == $subType} {
      tplet objects $aa resolveChildURIs
      foreach object $objects {
        if {[tpwith $object isA AssociatedAttachment]} {
          set aa2 $object
          tplet object $aa2 resolveAttachmentURI
          if {[tpwith $object isA Occurrence]} {
            return $object
        }
      }
  }
```

```
tplet objects $aa resolveChildURIs
foreach object $objects {
   set result [RecursivelyFindHeader $object $contextType]
   if {$result != "null"} {
      return $result
    }
   }
   return null;
}
```

In the case that a shortcut does not entirely satisfies your needs you may re-implement it in Tcl including your improvements. For some shortcuts we already provide the Tcl equivalents. The FindHeader procedure above is the Tcl equivalent for the findHeader shortcut.

2-50 PLMXML Reader Guide

# 2.42 Get the Master Form from an Item Without Using Shortcuts

#### **Prerequisites:**

- Get the Master Form from a BOM Header or Line
- Find the Header of a Bill of Material (BOM) Without Using Shortcuts

#### **Example:**

```
proc GetMasterForm {Occurrence} {
   set MasterForm null
   # Get the AssociatedAttachements element referred to
   # from the BOM header occurrence for wich
   # role="IMAN_master_form" applies. Note that it is not
   # referred to from the Item
   tplet AssociatedAttachments $Occurrence resolveAssociatedAttachmentURIs
   foreach AssociatedAttachment $AssociatedAttachments {
      tplet Role $AssociatedAttachment getRole
      if {$Role == "IMAN_master_form"} {
         tplet MasterForm $AssociatedAttachment resolveAttachmentURI
      }
   }
   return $MasterForm
}
```

You have already learned that shortcuts can be re-implemented in Tcl if the shortcut does not entirely fit your needs. Here we present the Tcl equivalent for the getMasterForm shortcut.

# 2.43 Get the Type of an Item

# **Prerequisites:**

- Get the Item from an Item Revision
- Get the Type of an Item Revision

# **Example:**

```
tplet ItemType $Item getSubType
```

As with item revisions you get the type of an item using the getSubType method.

# 2.44 Get the Description of a ProcessRevision

#### **Prerequisites:**

• Get the Process Revision from a BOP Header or Line

# **Example:**

tplet ProcessRevisionDescription \$ProcessRevision getDescription

ProcessRevision elements may contain Description elements. Its text content can be retrieved with the getDescription method just as if it were an attribute.

# **Siemens Industry Software**

# Headquarters

Granite Park One 5800 Granite Parkway Suite 600 Plano, TX 75024 USA +1 972 987 3000

#### **Americas**

Granite Park One 5800 Granite Parkway Suite 600 Plano, TX 75024 USA +1 314 264 8499

# **Europe**

Stephenson House Sir William Siemens Square Frimley, Camberley Surrey, GU16 8QD +44 (0) 1276 413200

# **Asia-Pacific**

Suites 4301-4302, 43/F
AIA Kowloon Tower, Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

# **About Siemens PLM Software**

Siemens PLM Software, a business unit of the Siemens Industry Automation Division, is a leading global provider of product lifecycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.

© 2018 Siemens Product Lifecycle Management Software Inc. Siemens, the Siemens logo and SIMATIC IT are registered trademarks of Siemens AG. Camstar, D-Cubed, Femap, Fibersim, Geolus, I-deas, JT, NX, Omneo, Parasolid, Solid Edge, Syncrofit, Teamcenter and Tecnomatix are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. All other trademarks, registered trademarks or service marks belong to their respective holders.