

# Site Controller

## Style Sheets Reference Manual

Date: December 2, 2020



**Verifone®**

Verifone Confidential

Verifone, Inc.  
88 West Plumeria Drive  
San Jose, CA 95134  
Telephone: 408-232-7800  
<http://www.verifone.com>

© 2020 Verifone, Inc. All rights reserved.

No part of this publication covered by the copyrights hereon may be reproduced or copied in any form or by any means - graphic, electronic, or mechanical, including photocopying, taping, or information storage and retrieval systems - without written permission of the publisher.

The content of this document is subject to change without notice. The information contained herein does not represent a commitment on the part of Verifone. All features and specifications are subject to change without notice.

Verifone, Ruby SuperSystem, and Secure PumpPAY are registered trademarks of Verifone, Inc. Ruby Card, iOrder, and Commander Site Controller are trademarks of Verifone. All other brand names and trademarks mentioned in this document are the properties of their respective holders.

---

## Revision History

Date	Description
09/06/2017	Converted document from Sapphire to Site Controller to include Commander.
12/02/2020	Changed references from transactionSet to transSet. Updated Top-Level Transforms.



# Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Creating and Using Style Sheets .....</b>	<b>3</b>
Basics .....	3
Push Processing .....	5
Pull Processing .....	6
Modularity .....	7
Modularity Example: The Included Module .....	8
Modularity Example: The Including Module .....	10
<b>3. Top-Level Transforms .....</b>	<b>11</b>
Directory Structure .....	11
<b>4. Lists and Scripts.....</b>	<b>15</b>
Lists .....	15
ns:getSaleSet .....	15
ns:getRefundTLogEntrySet .....	15
ns:getNonRefundTLogEntrySet .....	16
ns:getTLogEntriesForCashier .....	16
ns:getTLogEntriesForRegister .....	16
ns:getTLogEntriesForDept .....	17
ns:getTLogEntriesForCategory .....	17
ns:getTLogEntriesForNetw .....	17
ns:getTLogEntriesForPlu .....	18
ns:getTLogEntriesForFeeDept .....	18
ns:getDrawerEntriesForCashier .....	18
ns:getTLogEntriesForPeriod .....	19
ns:getRawFidList .....	19
ns:getDeptList .....	19
ns:getCategoryList .....	20
ns:getNetwList .....	20
ns:getRawIidList .....	20
ns:getPluList .....	21
ns:getRawTaxList .....	21
ns:getRawTaxList .....	21
ns:getCashierList .....	22
ns:getRegisterList .....	22
ns:getMopList .....	22

ns:getMopSafeList . . . . .	23
ns:getFeeDeptList . . . . .	23
ns:getPeriodList . . . . .	23
ns:getLineSet . . . . .	24
ns:getRawFidListFromLines . . . . .	24
ns:getDeptListFromLines . . . . .	24
ns:getCategoryListFromLines . . . . .	25
ns:getRawIdListFromLines . . . . .	25
ns:getPluListFromLines . . . . .	25
ns:getLinesForDept . . . . .	26
ns:getLinesForCategory . . . . .	26
ns:getLinesForNetw. . . . .	26
ns:getLinesForPlu . . . . .	27
ns:getLinesWithPlu . . . . .	27
ns:getLinesWithoutPlu . . . . .	27
ns:getLinesForFeeDept . . . . .	28
ns:getLinesForInsideSales . . . . .	28
ns:getLinesForFuelSales . . . . .	28
ns:getDiscountSetFromLines . . . . .	29
ns:getPaylineSet . . . . .	29
ns:getPaylinesForMop . . . . .	29
ns:getTillItems . . . . .	30
ns:getTillItemsForMop . . . . .	30
Scripts . . . . .	31
function maskAcctNum(acctNumStr,unMaskedNum) . . . . .	31
function decodeAcctNum(acctNumStr) . . . . .	31
function getErrorCorrectItem(txt) . . . . .	31
function getErrorCorrectPrice(txt) . . . . .	31
function getErrorCorrectQty(txt) . . . . .	32
<b>5. Report Preferences and T-Log Reports . . . . .</b>	<b>33</b>
Report Preferences . . . . .	33
T-Log Report Examples . . . . .	35

# 1

## Introduction

The *Site Controller Style Sheets Reference Manual* provides an introduction to the XML style sheets (XSLT) used with the Verifone Site Controller.

XML makes it easy to encode data in a logical and understandable way. Because the format is a “tree-ordered grammar,” you can insert new information into the document without disrupting the rest of the tree. It is this feature that puts the “X” in XML, the “eXtensible Markup Language.”

This tree structure in the document allows an easy mapping to a tree structure in program space. For example, the DOM (Document Object Model) is an industry-standard, tree-oriented API for working with XML data. But in practice, writing Java or ‘C’ code to access data in a tree structure is more laborious and error prone than it might seem at first. To address this problem, the W3C introduced the “eXtensible Stylesheet Language Transformation,” or XSLT. XSLT is used to translate a document in one XML language into another document in a different XML language. (An XML language is a set of tag names that can appear in a predefined arrangement or order, the content of those tags also being specified (or restricted) to certain arrangements of characters (lexical forms)).

Since HTML is considered an XML language, XSLT is commonly used to change data available in XML format into “pretty print” format using HTML.

This manual describes Site Controller XSLT processing under XSLT 1.0.





# 2

## Creating and Using Style Sheets

XSLT is a text formatting language. It uses pattern matching and rule binding to decide how to proceed. As such it is considered a “declarative” rather than a “procedural” language. Declarative languages are rules based and are written using constructs that are generally more “static” and resistant to unexpected changes in the input data.

As a text formatting language, XSLT has much in common with other such languages (such as *sed*, *awk*, *nroff*, and *troff*) but not much in common with procedural languages such as ‘C’, COBOL, or even Java; it’s a different kind of programming, but the rewards are worth the required learning.

### Basics

The following sections are not intended as a general-purpose XSLT primer. (Refer to *XSLT Programmer’s Reference, 2<sup>nd</sup> Edition* by Michael Kay (Wrox Press, 2001) for an excellent introduction to XSLT.) Rather, the following sections are designed to introduce a few basic concepts used in Site Controller style sheet programming.

Here is one of the simplest possible XSLT programs:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
</xsl:stylesheet>
```

Applied to an XML document, this style sheet will strip out all the tags, concatenate all element content into one long string, and prefix the whole thing with the XML processing instruction. For example, the XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <b>hello</b>
  <c>world.</c>
</a>
```

gets transformed into:

```
<?xml version="1.0" encoding="UTF-8"?>helloworld.
```

Not well-formed XML, nor very useful. Clearly we need to do more. The first thing is to understand that the style sheet is organized into “templates.” Each template is called either when there is a “match” on the current node or when the template is called directly by “name.”

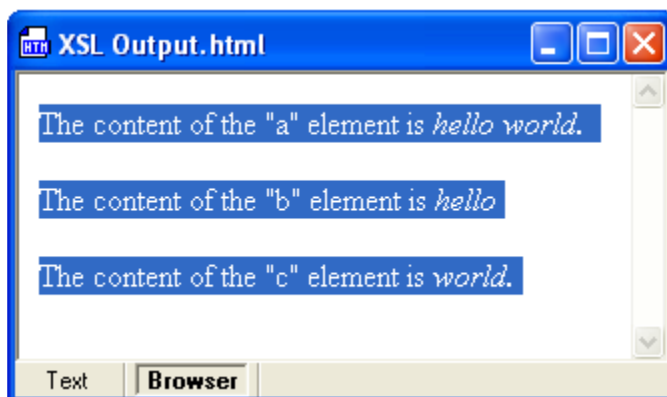
Here’s a slightly more exciting example that makes use of a template called when the processor matches the root-node of the document:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
<html>
<head>
<title>Simple HTML Document</title>
</head>
<body>
<p>The content of the &quot;a&quot; element is <i><xsl:value-of select="a"/></i></p>
<p>The content of the &quot;b&quot; element is <i><xsl:value-of select="a/b"/></i></p>
<p>The content of the &quot;c&quot; element is <i><xsl:value-of select="a/c"/></i></p>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

The output is a usable HTML document, showing in a browser as:



The rule

```
<xsl:template match="/">...
```

is sometimes called the “root rule” or “root template” because it is always the first to fire as the processor begins reading the input XML document. The root rule will usually set the stage for the processing model, which falls into one of two basic styles—*push processing* or *pull processing*.

## Push Processing

---

Push processing implies that the processor is “pushed along” by the structure of the input XML document. Style sheets using this model typically fire their templates (rules) using a “match” attribute, with the processing started with a call to “apply-templates.”

Here is a simple push style sheet:

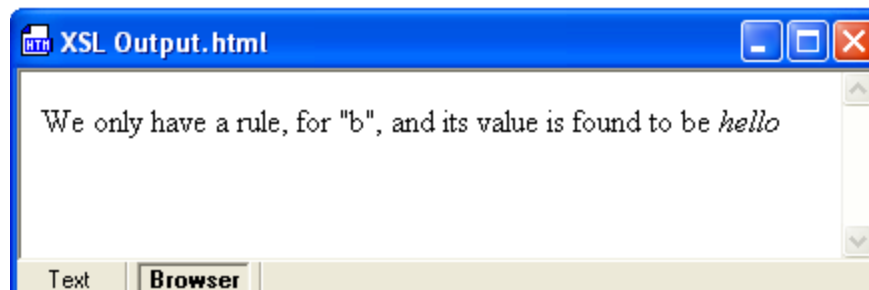
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <html>
    <head>
    <title>Simple HTML Document</title>
    </head>
    <body>
    <xsl:apply-templates/>
    </body>
    </html>
  </xsl:template>

  <xsl:template match="b">
    <p>We only have a rule, for &quot;b&quot;;, and its value is found to be <i><xsl:value-of
      select="."/></i></p>
  </xsl:template>

  <xsl:template match="@*|text()"/>
  </xsl:stylesheet>
```

This style sheet results in the following HTML output:



The last rule in the style sheet above says, “if the text in hand doesn’t match a rule, don’t do anything.” This prevents extra text in child elements from being displayed.

## Pull Processing

While push processing is extremely powerful for applications such as printing books, pull processing is more commonly used to process XML data for reporting purposes. Essentially the style sheet “pulls” values from the XML document, formats them, and outputs them according to how the style sheet is written. Careful readers will notice that this leads to a slightly less “declarative” programming style.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <html>
    <head>
    <title>Simple HTML Document</title>
    </head>
    <body>
    <xsl:call-template name="printC"/>
    <xsl:call-template name="printB"/>
    </body>
    </html>
  </xsl:template>

  <xsl:template name="printC">
    <p>The value of &quot;c&quot; is <i><xsl:value-of select="/a/c"/></i></p>
  </xsl:template>

  <xsl:template name="printB">
    <p>The value of &quot;b&quot; is <i><xsl:value-of select="/a/b"/></i></p>
  </xsl:template>

</xsl:stylesheet>
```

This style sheet renders the following output:



In practice, Site Controller XSLT contains a mixture of *pull* and *push*, though the basic structure of the report programs is always *pull*.

## Modularity

As with any large-size program, large XSLT programs can benefit from being split into modules. Among the benefits of such splitting are:

- Understandability
- Maintainability
- Code reuse

Although XSLT has facilities to support composing modules into whole programs, these facilities are not commonly in use, nor is their use in supporting “style sheet composition” mentioned in the literature. The rules for supporting composition are fairly simple:

- Put all style sheets in your library of modules in a well-known place relative to one another. Site Controller practice is to put a style sheet in a directory with the same name as the root element that the style sheet expects under a general purpose directory “vfit” (for Verifone, Inc. Transforms). This results in all transforms that work on, say “tlog” to fall in the same directory.
- Give each style sheet its own namespace and a local prefix, based on its file name and position in the library.
- Name all global parameters and global variables with QNames starting with the style sheet namespace prefix.
- Name all templates in the style sheet (if they are named) with a QName starting with the style sheet namespace prefix.
- For templates that use the “match” instead of the “name” attribute, set a mode for the template that is a QName starting with the style sheet namespace prefix. Obviously, calls to `apply-templates` targeting those templates will have to be qualified with a mode of the same QName.
- Treat the root template (the attribute `match="/"` appears) as a special case—have it call a template with local name “main” in the style sheet namespace, passing the current **nodeset** as a parameter. Use the “main” template to actually start the processing. Pass the current **nodeset** to other *pull* templates as needed as a parameter. Do not rely on the current node unless inside the control of a `for-each` or `apply-templates` directive.

The net effect of these rules is that there is less reliance on global data (the current node), more use of parameters, assurance that rules won’t fire by mistake when matched from some distant template, that similarly named templates across documents won’t conflict, and that any given completed style sheet can either be called on its own (through the root template) or called from another style sheet using its “main” template.

A simple example showing how this scheme works along with other common Site Controller XSLT practices follows.

## Modularity Example: The Included Module

---

The following style sheet is located on the Site Controller at the following location:

vfit/tlog/tlogSet.xslt

Note the use of standard prefixes for common namespaces.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ext="urn:schemas-microsoft-com:xslt"
xmlns:ns="http://verifone.com/isd/vfit/transSet/tlogSet"
xmlns:t="urn:vfi-sapphire:tlog.2003-06-27"
exclude-result-prefixes="xsl ext ns">

<xsl:import href="scripts.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
<xsl:call-template name="ns:main">
<xsl:with-param name="inputDoc" select="."/>
</xsl:call-template>
</xsl:template>

<!-- main -->
<xsl:template name="ns:main">
<xsl:param name="inputDoc"/>

<!-- processing -->
</xsl:template>
</xsl:stylesheet>
```

### XML Processing Namespaces

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ext="urn:schemas-microsoft-com:xslt"
```

The “xsl” and “xsi” prefixes are fairly standard and should be used verbatim. The “ext” prefix is for processor extensions, required for the use of the “node-set()” function. The “ext” declaration here works for Microsoft processors. Other processors will need a different namespace, though the continued use of the prefix “ext” is recommended.

### Local Style Sheet Namespace

```
xmlns:ns="http://verifone.com/isd/vfit/transSet/tlogSet"
```

Normally the prefix “ns” is used for the local style sheet. Note how the URL is “manufactured” from the style sheet name and location (vfit/transSet/tlogSet.xslt) by removing the “.xslt” and prepending the Verifone URL. Note that you can compose namespaces in different ways and with different URLs. The point is to be as consistent as possible. Too much flexibility in naming may defeat the goals of the scheme.

## Input and Output Namespaces

```
xmlns:t="urn:vfi-sapphire:tlog.2003-06-27"
```

These prefixes must be defined if they are expected on input or written on output.

## Namespaces Excluded on Output

```
exclude-result-prefixes="xsl ext ns"
```

Putting a prefix in this list will prevent its being defined in the output document. Since the namespaces listed here are used only in the processing style sheet, it is appropriate to exclude them.

## Imports

```
<xsl:import href="scripts.xslt"/>
```

Because the locations of all style sheets are relative, this import composes a transform including a file called “scripts.xslt” in the same directory as the importing style sheet. Note that, even though we are using this style sheet as an “included” module, it can include other modules as in this case.

## Default Template and Standard “Main” Template

```
<xsl:template match="/">
<xsl:call-template name="ns:main">
<xsl:with-param name="inputDoc" select="."/>
</xsl:call-template>
</xsl:template>
```

```
<xsl:template name="ns:main">
<xsl:param name="inputDoc"/>
```

```
<!-- processing -->
</xsl:template>
```

This first template intercepts the root element match and instigates pull processing using the template called “main” in the style sheet namespace, passing the current nodeset as a parameter called (by convention) **inputDoc**.

## Modularity Example: The Including Module

---

The following style sheet includes the one defined above. Both are in the same directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ext="urn:schemas-microsoft-com:xslt"
xmlns:ns="http://verifone.com/isd/vfit/transSet/tlogDoc"
xmlns:ns1="http://verifone.com/isd/vfit/transSet/tlogSet"
xmlns:t="urn:vfi-sapphire:tlog.2003-06-27"
exclude-result-prefixes="xsl ext ns ns1"
>
<xsl:import href="tlogSet.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
<xsl:call-template name="ns:main">
<xsl:with-param name="inputDoc" select="."/>
</xsl:call-template>
</xsl:template>

<xsl:template name="ns:main">
<xsl:param name="inputDoc"/>

<xsl:variable name="output1">
<xsl:call-template name="ns1:main">
<xsl:with-param name="inputDoc" select="$inputDoc"/>
</xsl:call-template>
</xsl:variable>

<t:tlog>
<xsl:copy-of select="$output1"/>
</t:tlog>
</xsl:template>
</xsl:stylesheet>
```

Note the following points:

- This style sheet is in a file called `vfit/transSet/tlogDoc.xslt`.
- The namespace for the style sheet to be included (defined above) has the prefix “ns1.”
- There is a standard root template (calling a local “main”).
- The local main template calls the main template of the included style sheet (using the ns1 prefix).
- The results of this first transform are stored in a variable “output1.”
- The transform creates a start tag, copies the contents of the “output1” variable, and then puts the closing tag.
- This transform could in turn be included in another transform, and so on.



# 3

## Top-Level Transforms

### Directory Structure

This section provides the directory structure for the top-level transforms available for Site Controller. For example, one way to access the first transform in the following table is as follows:

[https://site-controller\\_IPAddr/vfit/periodTargs/summaryPd/tr\\_SummaryAllR.xml](https://site-controller_IPAddr/vfit/periodTargs/summaryPd/tr_SummaryAllR.xml)

Transform	Directory
Mobile Payments: Above Site	vfit/periodTargs/aboveSitePd/tr_aboveSite.xml
Age Validation	vfit/periodTargs/ageVPd/tr_VerifyAge.xml
Fuel Attendant	vfit/periodTargs/attendantPd/tr_attendantPd.xml
Fuel Autocollect	vfit/periodTargs/autoCollectPd/tr_autoCollectPd.xml
Batch Details	vfit/periodTargs/batchDetailsPd/tr_batdetail.xml
Batch Summary	vfit/periodTargs/batchSummaryPd/tr_batsum.xml
Batch Totals	vfit/periodTargs/batchTotalsPd/tr_battot.xml
PBT Summary	vfit/periodTargs/biometricPd/tr_biometricPd.xml
Blend Product	vfit/periodTargs/blendPd/tr_blend.xml
Card Type Product	vfit/periodTargs/cardTypeProdPd/tr_cardtypeprod.xml
Car Wash	vfit/periodTargs/carWashPd/tr_carWashPd.xml
Fuel Cash Acceptor	vfit/periodTargs/cashAccPd/tr_cashAccPd.xml
Cash Activity Audit	vfit/periodTargs/cashActivity/tr_CashActivity.xml
Cashier	vfit/periodTargs/cashierPd/tr_cashierPd.xml
Cashier Tracking	vfit/periodTargs/cashierTrackingPd/tr_cashierTrackingPd.xml
Cash Reconciliation	vfit/periodTargs/cashReconcilePd/tr_cashrecon.xml
Category	vfit/periodTargs/categoryPd/tr_CategoryAllC.xml

Transform	Directory
Category by Cashier	vfit/periodTargs/categoryPd/tr_CategoryC.xml
Category by Register	vfit/periodTargs/categoryPd/tr_Category.xml
Credit Card Exception	vfit/periodTargs/ccExceptionPd/tr_ccexcept.xml
Carwash Pay Point	vfit/periodTargs/cwPaypointPd/tr_cwpaypoint.xml
Fuel DCR Statistics	vfit/periodTargs/dcrstPd/tr_dcrstat.xml
Deal	vfit/periodTargs/dealPd/tr_deal.xml
Department	vfit/periodTargs/departmentPd/tr_DeptAllC.xml
Department by Cashier	vfit/periodTargs/departmentPd/tr_DeptC.xml
Department by Register	vfit/periodTargs/departmentPd/tr_Dept.xml
ECheck	vfit/periodTargs/eCheckPd/tr_echeck.xml
Electronic Safe Cashier	vfit/periodTargs/eSafeCashierPd/tr_esafecashier.xml
Electronic Safe Content	vfit/periodTargs/eSafeContentPd/tr_esafecontent.xml
Electronic Safe EOD	vfit/periodTargs/eSafeEodPd/tr_esafeeod.xml
Fuel Dispenser	vfit/periodTargs/fpDispensePd/tr_fpDispenserPd.xml
Fueling Position/ Product (Hose)	vfit/periodTargs/fpHosePd/tr_fpHosePd.xml
Fueling Position/ Product (Hose) Running	vfit/periodTargs/fpHoseRunningPd/tr_fpHoseRunningPd.xml
Fueling Position/ Product (Hose) Test	vfit/periodTargs/fptHosePd/tr_fphosetest.xml
Fuel Price Change	vfit/periodTargs/fuelPriceChange/tr_fuelPriceChange.xml
Fuel Street Price Change	vfit/periodTargs/fuelPriceChange/tr_fuelStreetPriceChange.xml
Fuel Tax	vfit/periodTargs/fuelTaxPd/tr_fueltax.xml
Cashier Fuel Indicators	vfit/periodTargs/fuelTransIndicatorsPd/tr_CashierFuelIndicators.xml
Hourly	vfit/periodTargs/hourlyPd/tr_Hourly.xml
House Account Activity	vfit/periodTargs/houseAccountPd/tr_houseaccount.xml
Kiosk	vfit/periodTargs/kioskReportPd/tr_kioskReport.xml
Loyalty	vfit/periodTargs/loyaltyPd/tr_loyaltyPd.xml
Loyalty Price Per Gallon	vfit/periodTargs/loyaltyPd/tr_loyaltyPPGPd.xml
Loyalty Transaction Detail	vfit/periodTargs/loyaltyPd/tr_loyaltyTransactionDetailPd.xml
Merchandise Sales	vfit/periodTargs/merchandiseSales/tr_merchsales.xml
Money Order Device	vfit/periodTargs/moDevicePd/tr_moDevicePd.xml

Transform	Directory
Network Journal	vfit/periodTargs/netJrnlPd/tr_netJrnl.xml
Network Product	vfit/periodTargs/netProdPd/tr_netProd.xml
Network Totals	vfit/periodTargs/netTotPd/tr_networkTotals.xml
Paid In / Paid Out	vfit/periodTargs/paidInPaidOutPd/tr_paidinout.xml
Payroll	/vfit/periodTargs/payrollPd/tr_payroll.xml
PLU	vfit/periodTargs/pluPd/tr_pluAllC.xml
PLU by Cashier	vfit/periodTargs/pluPd/tr_pluC.xml
PLU Promo	vfit/periodTargs/pluPromoPd/tr_pluPromoC.xml
PLU Top/Bottom Sellers	vfit/periodTargs/pluPd/tr_PLUTopSellersTarg.xml
POP Definition Discount	vfit/periodTargs/popDefPd/tr_popDef.xml
POP Fuel Discount	vfit/periodTargs/popDiscPd/tr_popDisc.xml
POP Program Discount	vfit/periodTargs/popDiscPrgmPd/tr_popDiscPrgm.xml
Prepaid Cards	vfit/periodTargs/prepaidPd/tr_prepaid.xml
Proprietary Network Product	vfit/periodTargs/propProdPd/tr_propProd.xml
Proprietary Network Card	vfit/periodTargs/propCardPd/tr_propCardPd.xml
Fuel Product/ Price Level	vfit/periodTargs/prPriceLvlPd/tr_prPriceLvl.xml
Quick Count	vfit/periodTargs/quickCount/tr_QuickCountTarg.xml
Safe Drop	vfit/periodTargs/safedropPd/tr_safedrop.xml
Fuel Service Level/ Price Level	vfit/periodTargs/slPriceLvlPd/tr_SLPriceLvl.xml
Summary	vfit/periodTargs/summaryPd/tr_SummaryAllR.xml
Summary By Register	vfit/periodTargs/summaryPd/tr_Summary.xml
Fuel Tank Monitor	vfit/periodTargs/tankMonitorPd/tankmonitor.xml
Fuel Tank	vfit/periodTargs/tankPd/tr_TankPd.xml
Fuel Tank Reconciliation	vfit/periodTargs/tankRecPd/tr_tankRecPd.xml
Taxable Rebates	vfit/periodTargs/taxableRebatesPd/tr_taxablerebates.xml
Tax	vfit/periodTargs/taxPd/tr_Tax.xml
Fuel Tier/ Product	vfit/periodTargs/tierProductPd/tr_tierProduct.xml
Transaction Indicators	vfit/periodTargs/transIndicatorsPd/tr_CashierTxnIndicators.xml
Unpaid Transaction	vfit/periodTargs/unpaidTransRptPd/UnpaidTransRpt.xml
Fuel Dispenser	vfit/fuelTotals/tr_dispenser.xml

<b>Transform</b>	<b>Directory</b>
Safe Drop	vfit/transSet/safedropPd.xsl
PLU Exception	vfit/transSet/pluExceptionPd.xsl
MoneyOrder	vfit/transSet/moneyOrderPd.xsl
Ending Cash Reconciliation	vfit/transSet/buildCashReconcile.xslt
Credit Card Exceptions	vfit/transSet/buildCCEExceptions.xslt
Paid Ins - Paid Outs	vfit/transSet/buildPaidInOut.xslt
Merchandise Sales	vfit/transSet/buildMerchandiseSales.xslt
Customer Counts	vfit/transSet/buildCustomerCount.xslt
Cashier Trans Indicators	vfit/transSet/buildTransactionIndicators.xslt
Cashier Fuel Indicators	vfit/transSet/buildFuelTransactionIndicators.xslt
End of Day	vfit/transSet/buildSummary.xslt
Card Type Product Report	vfit/transSet/buildCardtypeProd.xslt
Network Journal	vfit/transSet/buildNetworkJrnl.xslt
Money Order Transactions	vfit/transSet/buildMoneyOrder.xslt
Cashier Tracking	vfit/transSet/buildCashierTracking.xslt
transSet to tlog	vfit/transSet/tlogDoc.xslt

# 4

## Lists and Scripts

### Lists

This section provides a library of generic templates that work against the t:tlog.

#### **ns:getSaleSet**

---

**Purpose**

Routine to return financially meaningful T-Log entries.

**Description**

Creates a list of tlogEntries with a financial effect on totals (a kind of **tlogEntrySet**).

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:sale and t:adjustments.

#### **ns:getRefundTLogEntrySet**

---

**Purpose**

Routine to select T-Log entries based on refund criteria.

**Description**

Creates a list of tlogEntries with a financial effect on totals that are refunds.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:tlog entries where @refund='true'.

## **ns:getNonRefundTLogEntrySet**

---

### **Purpose**

Routine to select T-Log entries that are not refunds.

### **Description**

Creates a list of tlogEntries with a financial effect on totals that are not refunds.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

### **Returns**

Node set of t:tlog entries where not(@refund='true'.

## **ns:getTLogEntriesForCashier**

---

### **Purpose**

Routine to select T-Log entries for a specific cashier.

### **Description**

Creates a list of tlogEntries for a specific cashier and cashier period.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**cashier**—A t:cashier node.

### **Returns**

Node set of t:tlog entries where t:cashier/@sysid and t:cashier/period equals the **cashier** parameter.

## **ns:getTLogEntriesForRegister**

---

### **Purpose**

Routine to select T-Log entries for a specific register.

### **Description**

Creates a list of tlogEntries for a specific register.

### **Parameters**

**tlogEntrySet**—argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**register**—A t:register node.

### **Returns**

Node set of t:tlog entries where t:register/@sysid equals the **register** parameter.

## **ns:getTLogEntriesForDept**

---

### **Purpose**

Routine to select T-Log entries for a specific department.

### **Description**

Creates a set of tlogEntries containing at least one line item for specific department.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**dept**—A t:dept node.

### **Returns**

Node set of t:tlog entries where t:dept/@sysid equals the **dept** parameter.

## **ns:getTLogEntriesForCategory**

---

### **Purpose**

Routine to select T-Log entries for a specific category.

### **Description**

Creates a set of tlogEntries containing at least one line item for a specific category.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**cat**—A t:cat node.

### **Returns**

Node set of t:tlog entries where t:cat/@sysid equals the **cat** parameter.

## **ns:getTLogEntriesForNetw**

---

### **Purpose**

Routine to select T-Log entries for a specific network product code.

### **Description**

Creates a set of tlogEntries containing at least one line item for a specific network product code

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, entriessselect="\$inputDoc/t:tlog"

**netw**—A t:netw node

### **Returns**

Node set of t:tlog entries where t:netw/@sysid equals the netw parameter

## **ns:getTLogEntriesForPlu**

---

### **Purpose**

Routine to select T-Log entries for a specific PLU.

### **Description**

Creates a set of tlogEntries containing at least one line item for a specific PLU.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**plu**—A t:plu node.

### **Returns**

Node set of t:tlog entries where t:plu/upc and t:plu/modifier equals the **plu** parameter.

## **ns:getTLogEntriesForFeeDept**

---

### **Purpose**

Routine to select T-Log entries for a specific fee department.

### **Description**

Creates a set of tlogEntries containing at least one line item with a fee against a specific department.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**dept**—A t:dept node.

### **Returns**

Node set of t:tlog entries where t:fee/t:dept/@sysid equals the **dept** parameter.

## **ns:getDrawerEntriesForCashier**

---

### **Purpose**

Routine to select cashier drawer entries for a specific cashier.

### **Description**

Creates a set of cashier drawer entries (t:closeCashier, t:openCashier, and t:countCashier) for a given cashier.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**cashier**—A t:cashier node.

### **Returns**

Node set of t:closeCashier, t:openCashier and t:countCashier entries where t:cashier/@sysid and t:cashier/period equals the **cashier** parameter.



## ns:getTLogEntriesForPeriod

---

**Purpose**

Routine to select entries for a specific periodtype (HOUR, SHIFT, or DAY) and a specific period sequence

**Description**

Creates a set of tlogEntries for a specific periodtype (HOUR, SHIFT or DAY) and a specific period sequence.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**period**—A vs:period node.

**Returns**

Node set of tlog entries where vs:period/@sysid and vs:period/@periodSeqNum equals the **period** parameter.

## ns:getRawFidList

---

**Purpose**

Routine to create a raw list of fid entries.

**Description**

Creates a raw list of financialIds (dept, subDept, cat, and netw) to work with. Includes only those that have effect on the drawer totals.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of fid entries.

## ns:getDeptList

---

**Purpose**

Routine to create a sorted list of unique departments from the T-Log.

**Description**

Creates a sorted list of unique departments from the T-log. Calls ns:getRawFidList.

**Parameters**

**tlogEntrySet** - argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:dept entries.

## ns:getCategoryList

---

**Purpose**

Routine to create a sorted list of unique categories from the T-Log.

**Description**

Creates a sorted list of unique categories from the T-Log. Calls ns:getRawFidList.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:cat entries.

## ns:getNetwList

---

**Purpose**

Routine to create a sorted list of unique network product codes from the T-Log.

**Description:**

Creates a sorted list of unique network product codes from the T-Log. Calls ns:getRawFidList.

**Parameters:**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns:**

Node set of t:netw entries.

## ns:getRawlidList

---

**Purpose**

Routine to create a raw list of item IDs (PLUs).

**Description**

Creates a raw list of itemIds (PLUs) to work with. Includes only those that have effect on the drawer totals.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**Returns**

Node set of iid entries.

## **ns:getPluList**

---

**Purpose**

Routine to create a sorted list of unique item IDs (PLUs).

**Description**

Creates a sorted list of unique PLUs from the tlog, plus are unique if they have different modifiers.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of iid entries.

## **ns:getRawTaxList**

---

**Purpose**

Routine to create raw list of tax rates.

**Description:**

Creates a raw list of taxes to work with. The taxes are from the t:sumTax element and not based on just the line items.

**Parameters:**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns:**

Node set of vs:taxrateBase entries.

## **ns:getRawTaxList**

---

**Purpose**

Routine to create list of tax rates.

**Description**

Creates a list of all vs:taxrateBase that appear in the t:tlog t:sumTax element, sorted first by tax rate then by name. Calls getRawTaxList.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**Returns**

Node set of vs:taxrateBase entries.

## **ns:getCashierList**

---

**Purpose**

Routine to create a raw list of cashiers.

**Description**

Creates a sorted list of all unique cashiers and periods.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:cashier entries.

## **ns:getRegisterList**

---

**Purpose**

Routine to create a list of registers.

**Description**

Creates a sorted list of all unique registers and tills.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**Returns**

Node set of t:register entries

## **ns:getMopList**

---

**Purpose**

Routine to create a list of methods of payment (MOPs).

**Description**

Creates a sorted list of all unique MOPs @sysid and @cat from paylines.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:mop entries.

## ns:getMopSafeList

---

**Purpose**

Routine to create a list of MOPs from Safe drops and loans.

**Description**

Creates a sorted list of all unique MOPs @sysid and @cat from safe drops and loans.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**Returns**

Node set of t:mop entries.

## ns:getFeeDeptList

---

**Purpose**

Routine to create a list of fee departments.

**Description**

Creates a sorted list of all unique departments assigned to a fee.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

**Returns**

Node set of t:dept entries.

## ns:getPeriodList

---

**Purpose**

Routine to create a list of period sequence numbers.

**Description:**

Creates a list of period sequences for a specific period type (sysid 0 =HOUR, sysid 1 =SHIFT, sysid 2 =DAY).

**Parameters:**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**period**—A vs:period node.

**Returns**

Node set of vs:period entries.

## **ns:getLineSet**

---

### **Purpose**

Routine to create a list of lines.

### **Description**

Creates a list of line item entries.

### **Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog"

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getRawFidListFromLines**

---

### **Purpose**

Routine to create a raw list of fids.

### **Description**

Creates a raw list of financialIds (**dept**, **subDept**, **cat**, and **netw**) to work with. Includes only those that have an effect on the drawer totals. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

### **Returns**

Node set of fid entries.

## **ns:getDeptListFromLines**

---

### **Purpose**

Routine to create a sorted list of unique departments from lines.

### **Description**

Creates a sorted list of unique departments from a line set.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet

### **Returns**

Node set of t:dept entries.

## **ns:getCategoryListFromLines**

---

**Purpose**

Routine to create a sorted list of unique categories from lines.

**Description**

Creates a sorted list of unique categories from a line set.

**Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**Returns**

Node set of t:cat entries.

## **ns:getRawIidListFromLines**

---

**Purpose**

Routine to create list of itemIds (PLUs) from lines.

**Description**

Creates a raw list of itemIds (PLUs) to work with. Includes only those that have an effect on the drawer totals. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

**Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**Returns**

Node set of iid entries.

## **ns:getPluListFromLines**

---

**Purpose**

Routine to create sorted list of unique itemIds (PLUs) from lines.

**Description**

Creates a sorted list of unique PLUs from the lineSet. PLUs are unique if they have different modifiers. Call ns:getRawIidListFromLines.

**Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**Returns**

Node set of t:plu entries

## **ns:getLinesForDept**

---

### **Purpose**

Routine to create list of lines for a specific department.

### **Description**

Creates a set of line items for a specific department. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**dept**—A t:dept node.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesForCategory**

---

### **Purpose**

Routine to create list of lines for a specific category.

### **Description**

Creates a set of line items for a specific category. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**cat**—A t:cat node

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesForNetw**

---

### **Purpose**

Routine to create list of lines for a specific network product code.

### **Description**

Creates a set of line items for a specific network product code. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**netw**—A t:netw node.

### **Returns**

Node set of t:line, t:cline, and t:sline entries



## **ns:getLinesForPlu**

---

### **Purpose**

Routine to create list of lines for a specific PLU.

### **Description**

Creates a set of line items for a specific PLU. Tests for the existence of both t:cline and t:sline. If this template is called a second time, t:sline is returned

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**plu**—A t:plu node.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesWithPlu**

---

### **Purpose**

Routine to create list of lines that have a PLU.

### **Description**

Creates a set of line items that contain a PLU. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesWithoutPlu**

---

### **Purpose**

Routine to create list of lines that do not have a PLU.

### **Description**

Creates a set of line items that do not contain a PLU. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline or t:sline as returned by ns:getLineSet.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesForFeeDept**

---

### **Purpose**

Routine to create list of lines that have a specific fee department.

### **Description**

Creates a set of line items that have a specific fee department. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**dept**—A t:dept node.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesForInsideSales**

---

### **Purpose**

Routine to create list of lines that are non-fuel sales.

### **Description**

Creates a set of line items that are non-fuel. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getLinesForFuelSales**

---

### **Purpose**

Routine to create list of lines that are fuel sales.

### **Description**

Creates a set of line items that are fuel sales. Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

### **Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet

### **Returns**

Node set of t:line, t:cline, and t:sline entries.

## **ns:getDiscountSetFromLines**

---

**Purpose**

Routine to create list of discounts.

**Description**

Creates a set of discount items (t:promo, t:disc, t:match, and t:combo). Tests for both the existence of t:cline and t:sline. If this template is called a second time, t:sline is returned.

**Parameters**

**lineSet**—Node set of lines stripped to the t:line, t:cline, or t:sline as returned by ns:getLineSet.

**Returns**

Node set of t:promo, t:disc, t:match, and t:combo entries.

## **ns:getPaylineSet**

---

**Purpose**

Routine to create list of paylines.

**Description**

Creates a set of paylines.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:payline entries.

## **ns:getPaylinesForMop**

---

**Purpose**

Routine to create list of paylines for a specific MOP.

**Description**

Creates a set of paylines for a specific MOP.

**Parameters**

**paylineSet**—Node set of lines stripped to the t:payline, as returned by ns:getPaylineSet.

**mop**—A t:mop node.

**Returns**

Node set of t:payline entries.

## **ns:getTillItems**

---

**Purpose**

Routine to create list of t:tillxxxx items (like drops, loans, payins, and payouts).

**Description**

Creates a set of t:tillxxxx items like drops, loans, payins, and payouts.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example, select="\$inputDoc/t:tlog".

**Returns**

Node set of t:tillxxxx entries.

## **ns:getTillItemsForMop**

---

**Purpose**

Routine to create list of t:tillxxxx items (like drops, loans, payins, and payouts) for a specific MOP.

**Description**

Creates set of t:tillxxxx items like drops, loans, payins, and payouts for a specific MOP.

**Parameters**

**tlogEntrySet**—Argument stripped of the t:tlog element, holding tlog entries; for example select="\$inputDoc/t:tlog".

**mop**—A t:mop node.

**Returns**

Node set of t:tillxxxx entries.

## Scripts

Scripts.xslt contains a set of JavaScript functions. This section describes only the functions that are currently used against, or meaningful to, the tlog.

### **function maskAcctNum(acctNumStr,unMaskedNum)**

---

**Description**

This function returns a string where the account will be \* masked except for the numbers specified by unMaskedNum. If unMaskedNum is not passed, it defaults to 4.

**Parameters**

**acctNumStr**—A string

**unMaskedNum** [optional]—A positive integer

### **function decodeAcctNum(acctNumStr)**

---

**Description**

This function returns a string with the account number decoded.

**Parameter**

**acctNumStr**—A string of an encoded acct number

**Note:** *The following three functions are used to extract information from the t:event element with respect to error corrects. More recently, a copy of trLine is included in the error correct event and these functions are no longer needed.*

### **function getErrorCorrectItem(txt)**

---

**Description**

This function returns the description of the error corrected item.

**Parameter**

**txt**—The contents of the text element of t:event/@type='errorCorrect' excluding the first 40 characters [which are \*].

### **function getErrorCorrectPrice(txt)**

---

**Description**

This function returns the price of the error corrected item.

**Parameter**

**txt**—The contents of the text element of t:event/@type='errorCorrect'.

## **function getErrorCorrectQty(txt)**

---

### **Description**

This function returns the quantity of the error corrected item.

### **Parameter**

**txt**—The contents of the text element of t:event/@type='errorCorrect'.

# 5

## Report Preferences and T-Log Reports

### Report Preferences

ReptPrefs is an XML document that holds site-specific or customer-specific information used to process T-Log based reports. Various style sheets (XSLTs) include the reptPrefs to “customize” the totals that are reported. There are five types of preference:

- **deptPrefs** contains deptGrp elements, which are a collection of department numbers and names that should be included or excluded in the report.
- **categoryPrefs** contains categoryGrp elements, which are a collection of category numbers and names that should be included or excluded in the report.
- **pluPrefs** contains pluGrp elements, which are a collection of plus with modifier and name that should be included or excluded in the report.
- **mopPrefs** contains mopGrp elements, which are a collection of MOPs and description that should be included or excluded in the report.
- **reptDefinitionMap** is used to contain a collection of different maps; that is to say, the reptDefinitionMap can have Dept, Category, MOP and PLU maps. Only one reptDefinitionMap can exist.

A group name must be unique with respect to all groups, and map names must be unique with respect to all maps. Group and map names cannot contain spaces and special characters, such as @, #, \$, %, ^, etc. All names are case sensitive.

Maps contain the same information as groups; the reason for having both maps and groups is so that a name assigned to a group can be reused for a map.

There are some map and group names that are currently used or referenced by existing style sheets (XSLT). Changing these names or deleting the entries will cause the associated report to no longer function correctly. The style sheets have been coded in such a manner that department and category groups or maps are interchangeable. That is, any group or map name currently assigned to a category can be assigned to a department, and vice versa. This is because categories and departments are similar in functionality. For

example, if a name from a category is assigned to a department, the category must be renamed or deleted, since group and map names must be unique.

The following table provides a list of map and group names and what XSLT file references them as well as the “default” display name of the resulting report. Reports that use maps will change only if the associated map is changed, and reports that use groups will change only if the associated group is changed, regardless of whether they share the same name.

Display Name	Group Type & Name	Map Type & Name	Transform Name
Credit Card Exceptions		MOP: imprinterSales	buildCCExceptions.xslt
Merchandise Sales	CAT: merchandiseSales-1		buildMerchandiseSales.xslt
End of Day		CAT: merchandiseSales CAT: otherServices DEPT: carWashSales DEPT: lottoSales DEPT: lotterySales DEPT: moneyOrderSales MOP: lottoWinners MOP: lotteryWinners MOP: driveOffs MOP: imprinterSales MOP: creditSales MOP: debitSales MOP: coupons MOP: overShort	buildSummary.xslt
Card Type Product Report	**cardTypeProductReport  **There is currently no entry in reptPrefs. This is intended to be used to report/track Manual Fuel sales and would be either a DEPT or CAT group containing a list of departments that are used for Manual Fuel		buildCardtypeProd.xslt
Money Order Transactions	DEPT: moneyOrder		buildMoneyOrder.xslt



Cashier Tracking	PLU: pluTracking DEPT: deptTracking CAT: catTracking		buildCashierTracking.xslt
Ending Cash Reconciliation	MOP: cashReconciliation-1		buildCashReconcile.xslt

## T-Log Report Examples

The following style sheets produce cashier PLU reports. In the first example, the xslt in the transSet directory accepts a transSet document; that is, the root element of the document is named <transSet>. In the second example, the xslt in the tlog directory accepts a tlog document; that is, the root element of the document is named <t:tlog>.

### Example 1—file:transSet/cashierPluRept.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ext="urn:schemas-microsoft-com:xslt"
xmlns:ns="http://verifone.com/isd/vfit/transSet/cashierPluRept"
xmlns:ns1="http://verifone.com/isd/vfit/transSet/tlogDoc"
xmlns:ns2="http://verifone.com/isd/vfit/tlog/cashierPluRept"
xmlns:t="urn:vfi-sapphire:tlog.2003-06-27"
exclude-result-prefixes="xsl ext ns ns1 t ns2"
>

<!--create a "ext" prefix to support the MS XSLT processor extensions -->
<!--create a "ns" prefix for the local style sheet namespace-->
<!--create a "ns1" prefix with the local namespace of tlogDoc.xslt which will be imported -->
<!--create a "ns2" prefix with the local namespace of cashierPluRept.xslt in the tlog directory which
will be imported -->
<!-- create a "t" prefix with the namespace for the t:tlog-->

<!-- import the tlogDoc.xslt used to convert transSet to t:tlog-->
<xsl:import href="tlogDoc.xslt"/>
<!-- import the ../tlog/cashierPluRept.xslt used to create the report from a t:tlog-->
<xsl:import href="../tlog/cashierPluRept.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

```
<!--root template -->
<xsl:template match="/">
  <!--call the named template main in the local style sheet namespace passing it the transSet
  document -->
  <xsl:call-template name="ns:main">
    <xsl:with-param name="inputDoc" select="."/>
  </xsl:call-template>
</xsl:template>

<!-- named template main with the local style sheet prefix-->
<xsl:template name="ns:main">
  <!--declare the parameter that can be accepted -->
  <xsl:param name="inputDoc"/>

  <!-- create a variable to hold/catch the results of the call to main in the tlogDoc.xslt-->
  <xsl:variable name="tlog">
    <!--call the named template main in the tlogDoc.xslt (note the prefix of ns1 which
    resolves to the local namespace for tlogDoc.xslt) -->
    <xsl:call-template name="ns1:main">
      <!--set the inputDoc param to ns:main inputDoc, using the MS XSLT processor extention  function
      node-set() to force inputDoc to a tree/node set -->
      <xsl:with-param name="inputDoc" select="ext:node-set($inputDoc)"/>
    </xsl:call-template>
  </xsl:variable>

  <!--create a variable to hold/catch the results of the call to main in ../tlog/cashierPluRept.xslt -->
  <xsl:variable name="report">
    <!--call the named template main in the ../tlog/cashierPluRept.xslt(note the prefix of ns1
    which resolves to the local namespace for ../tlog/cashierPluRept.xslt) -->
    <xsl:call-template name="ns2:main">
      <!--set the inputDoc param to the tlog variable, using the MS XSLT processor
      extention  function node-set() to force tlog to a tree/node set -->
      <xsl:with-param name="inputDoc" select="ext:node-set($tlog)"/>
    </xsl:call-template>
  </xsl:variable>

  <!-- using copy-of copy the contents of the report variable to std out-->
  <xsl:copy-of select="$report"/>
</xsl:template>
```

```
</xsl:stylesheet>
```

## Example 2—file:tlog/cashierPluRept.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ext="urn:schemas-microsoft-com:xslt"
xmlns:ns="http://verifone.com/isd/vfit/tlog/cashierPluRept"
xmlns:ns1="http://verifone.com/isd/vfit/tlog/lists"
xmlns:t="urn:vfi-sapphire:tlog.2003-06-27"
exclude-result-prefixes="xsl ext ns ns1">
  <!--create a "ext" prefix to support the MS XSLT processor extensions -->
  <!--create a "ns" prefix for the local style sheet namespace-->
  <!--create a "ns1" prefix with the local namespace of lists.xslt which will be imported -->
  <!-- create a "t" prefix with the namespace for the t:tlog-->

  <!-- import the lists.xslt used to generate lists/node sets of "things"-->
  <xsl:import href="lists.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <!--root template -->
  <xsl:template match="/">
    <!--call the named template main in the local style sheet namespace passing it the t:tlog document
    -->
    <xsl:call-template name="ns:main">
      <xsl:with-param name="inputDoc" select="."/>
    </xsl:call-template>
  </xsl:template>

  <!-- named template main with the local style sheet prefix-->
  <xsl:template name="ns:main">
    <!--declare the parameter that can be accepted -->
    <xsl:param name="inputDoc"/>
    <!--create a variables to hold the child nodes of the inputDoc using the MS XSLT
```

processor extension function node-set() to force inputDoc to a tree/node set -->

```
<xsl:variable name="rawSet" select="ext:node-set($inputDoc)/t:tlog"/>
```

<!--create a variable to hold the header information of the t:tlog, the t:hdr has information about when the t:tlog was opened, closed, site number and sequence number among other things -->

```
<xsl:variable name="header" select="$rawSet/t:hdr"/>
```

<!-- create a variable to hold/catch the results of the call to getCashierList in the lists.xslt  
This creates a list of cashiers-->

```
<xsl:variable name="cashierList">
```

<!--call the named template getCashierList in lists.xslt(note the prefix of ns1 which resolves to the local namespace for lists.xslt) -->

```
<xsl:call-template name="ns1:getCashierList">
```

<!--set the tlogEntrySet to rawSet. Note that the XSLT processor extension node-set() is not used  
this is because when rawSet was created it was forced to a tree/node set -->

```
<xsl:with-param name="tlogEntrySet" select="$rawSet"/>
```

```
</xsl:call-template>
```

```
</xsl:variable>
```

<!-- create a variable to hold/catch the results of the call to getPluList in the lists.xslt  
This creates a list of PLUs-->

```
<xsl:variable name="pluList">
```

<!--call the named template getPluList in lists.xslt (note the prefix of ns1 which resolves to the local namespace for lists.xslt) -->

```
<xsl:call-template name="ns1:getPluList">
```

<!--set the tlogEntrySet to rawSet. Note that the XSLT processor extension node-set() is not used  
this is because when rawSet was created it was forced to a tree/node set -->

```
<xsl:with-param name="tlogEntrySet" select="$rawSet"/>
```

```
</xsl:call-template>
```

```
</xsl:variable>
```

<!-- create a variable to hold/catch the results of the call to getNonRefundTLogEntrySet in the lists.xslt  
This creates a list of of sales that are not voids, or refunds-->

```
<xsl:variable name="sales">
```

<!--call the named template getNonRefundTLogEntrySet in lists.xslt

(note the prefix of ns1 which resolves to the local namespace for lists.xslt) -->

```
<xsl:call-template name="ns1:getNonRefundTLogEntrySet">
```

<!--set the tlogEntrySet to rawSet. Note that the XSLT processor extension node-set() is not used

this is because when rawSet was created it was forced to a tree/node set -->

```
<xsl:with-param name="tlogEntrySet" select="$rawSet"/>
```

```
</xsl:call-template>
```

```
</xsl:variable>
```

<!--create a variable to hold/catch the results of processing/creating the report-->

```
<xsl:variable name="report">
```

<!--loop through each cashier using the cashierList and a for-each-->

```
<xsl:for-each select="ext:node-set($cashierList)/*">
```

<!-- create a variable to hold the contents of the current node of the cashierList-->

```
<xsl:variable name="curCsh" select="."/>
```

<!--create a element/entry for the current cashier-->

```
<xsl:element name="cashierEntry">
```

<!--create a attribute to hold the name of the current cashier-->

```
<xsl:attribute name="name">
```

```
<xsl:value-of select="."/>
```

```
</xsl:attribute>
```

<!--create a attribute to hold the sysid of the current cashier-->

```
<xsl:attribute name="sysid">
```

```
<xsl:value-of select="./@sysid"/>
```

```
</xsl:attribute>
```

<!-- create a variable to hold/catch the results of the call to

getTLogEntriesForCashier in the lists.xslt

This creates a list of of sales that are not voids, or refunds for a specific cashier-->

```
<xsl:variable name="cshSales">
```

<!--call the named template getTLogEntriesForCashier in lists.xslt

(note the prefix of ns1 which resolves to the local namespace for lists.xslt) -->

```
<xsl:call-template name="ns1:getTLogEntriesForCashier">
```

<!--set the tlogEntrySet param to sales. The sales variable has

been "distilled" to only have sales related transactions.

Note that the XSLT processor extension node-set()

is used. This is because when sales was created it was not forced to a tree/node set -->

```
<xsl:with-param name="tlogEntrySet" select="ext:node-set($sales)"/>
```

<!--set the cashier param to curCsh. Note that the XSLT processor extension node-set() is used this is because when curCsh was created it was not forced to a tree/node set -->

```
<xsl:with-param name="cashier" select="ext:node-set($curCsh)"/>
```

```
</xsl:call-template>
```

```
</xsl:variable>
```

<!--loop through each plu using the pluList and a for-each-->

```
<xsl:for-each select="ext:node-set($pluList)/*">
```

<!-- create a variable to hold the contents of the current node of the pluList-->

```
<xsl:variable name="curPlu" select="."/>
```

<!--create a element/entry for the current PLU-->

```
<xsl:element name="pluEntry">
```

<!-- copy elements from the current node that will be meaningful in the context of the report-->

```
<xsl:copy-of select="./upc"/>
```

```
<xsl:copy-of select="./modifier"/>
```

```
<xsl:copy-of select="./name"/>
```

<!--create a element to hold the total sales -->

```
<xsl:element name="totalSales">
```

<!--sum up the tot of each line where the iid/t:plu/upc and

iid/t:plu/modifier to the current node Note that the XSLT

processor extension node-set() is used this is

because when cshSales was created it was not forced to a tree/node set -->

```
<xsl:value-of select="sum(ext:node-set($cshSales)/*//lines/*[iid/t:plu/upc = $curPlu/upc and iid/t:plu/modifier = $curPlu/modifier]/tot)"/>
```

```
</xsl:element>
```

```
<!-- create a element to hold the total qty -->
<xsl:element name="totalQty">
  <!--sum up the qty of each line where the iid/t:plu/upc and
  iid/t:plu/modifier to the current node Note that the XSLT
  processor extension node-set() is used this is
  because when cshSales was created it was not forced to
  a tree/node set -->
  <xsl:value-of select="sum(ext:node-set($cshSales)/*//lines/
  *[iid/t:plu/upc = $curPlu/upc and iid/t:plu/modifier =
  $curPlu/modifier]/qty)"/>
</xsl:element>
</xsl:element>
</xsl:for-each>
</xsl:element>
</xsl:for-each>
</xsl:variable>

<xsl:element name="cashierPluReport">
  <xsl:copy-of select="$header"/>
  <xsl:copy-of select="$report"/>
</xsl:element>
</xsl:template>
</xsl:stylesheet>
```

