



ProductXpress

Reference Guide

ProductXpress Calculator v3.7

April 2013

Copyright Statement

All right, title and interest in and to the software (the "Software") and the accompanying documentation or materials (the "Documentation"), including all proprietary rights, therein including all patent rights, trade secrets, trademarks and copyrights, shall remain the exclusive property of Hewlett-Packard Development Company, L.P. No interest, license or any right respecting the Software and the Documentation, other than expressly granted in the Software License Agreement, is granted by implication or otherwise.

© 2000 - 2013 Hewlett-Packard Development Company, L.P.

Disclaimer

Hewlett-Packard Development Company, L.P. will not be held responsible for any errors or omissions contained in this guide. However, an effort will be made to correct any such error or omission within a reasonable period of time following its detection.

Table of Contents

About This Guide	1
Calculator Overview	3
Introduction to Calculator.....	3
The Push-model and the Pull-model	3
Service Calculator	4
Embedded Calculator	4
Java Interface.....	4
C Interface	4
.NET Interface	4
Function Calculator	5
The Service Calculator	7
Service Calculator Configuration and Maintenance	7
Operational Settings Files	7
Deployment Package Management	12
Deploying with Deplmp.exe.....	13
Suspend, resume and shutdown with Deplmp.....	17
External Table Management	18
External Table Details	18
Using Tablelmp.exe	19
Loading CSV Files on Deployment of External Tables	22
Managing Secured Sockets.....	24
Test Mode of Calculator.....	25
The CalcServerTest.exe Testing Tool	26
The PxTest.exe Testing Tool	34
PxTest Syntax	34
PxTest Commands	34
PxTest Configuration File	37
SNMP Access to the Calculator	39
Defined MIBs	39
Calculator Web Portal	40
Connect to a Calculator.....	40
Perform a Calculation	40
View the Documentation	41
Download Calculator Examples.....	41
Interfacing the Service Calculator	41
The Raw TCP/IP Calling Protocol	42
The SOAP Calling Protocol	43
Web Services Description Language (WSDL)	62
The CalcClient Library	63
The Java Interface.....	81
The COM Interface.....	95
Perl Interface	97
XSLT Transformation Functions.....	97

Changing Client Priority	99
Runtime Considerations for the Service Calculator	100
Deployment Package Decryption	100
The Embedded Calculator	105
Embedded Calculator Configuration and Maintenance.....	105
Operational Settings Files	105
Interfacing the Embedded Calculator.....	108
Embedded Calculator with a C-interface.....	108
Interface specification for C.....	108
XML Format Definitions	141
Used Namespaces.....	141
Runtime Configuration Information	141
External Table Information	143
time list	143
feature references	144
feature values.....	144
children returned by getChildren callback function.....	145
linked resources returned by getLinkedResources callback function.....	145
resource roots returned by getResourceRoots callback function.....	146
parameter values for getFilteredData callback function	146
result returned by getFilteredData callback function	147
error returned by getLastPxError function	148
Embedded Calculator for Java.....	148
Interface Specification for Java.....	148
Embedded Calculator for .NET.....	187
Runtime Considerations for the Embedded Calculator.....	189
Multithreading	189
Creation of deployment object files with DepComp	189
Common Items for Service and Embedded Calculator	191
Output Aggregation (dealing with huge sets of data).....	191
The Protocol.....	191
Example	192
Calculator Input-Output Specification for the Push Calculator	194
Characters and Floating Point in Calculator.....	196
XML Reference	197
Common XML Elements.....	202
XML Elements for Static and Dynamic Deployments.....	204
Calculator Input Example	208
XML Elements for Function Deployments.....	213
Navigation Names, Short Names and Mangled Names.....	216
Output Attributes.....	216
Error behaviour attributes.....	217
Error Handling.....	219
Calculator Input and Output Syntax Specification	220
Static and Dynamic Deployments Definitions.....	221
Function Deployment Definitions.....	235

Common Definitions	242
XSLT Extension Functions.....	251
Introduction	251
API	251
Table XML Syntax	260
How to read multi-dimensional tables.....	262
TableID as return type	263
External Functions	263
Interface.....	263
External Function Context	264
Function Arguments	264
External Function Properties.....	264
Returning Results from an External Function.....	264
Handling External Function Data.....	265
Error Handling.....	268
Java Interface.....	269
.NET Interface	270
Compatibility Mode.....	274
Configuring External Functions	274
Other Issues	276
Common Configuration and Maintenance	277
Logging	277
Tracing	277
Tuning	278
Redistribution of Calculator Components.....	279
Troubleshooting Guide	280
Troubleshooting Steps	280
Appendix A - Examples for the Service Calculator	283
Appendix B - Specification of Calculator Input-Output XML	285
Appendix C - Examples for the Embedded Calculator	287
Appendix D - Transition Matrix	289
Appendix E - W3C XML Schema Datatypes	291

About This Guide

This is the Reference Guide for ProductXpress Calculator v3.7.

Intention and Usage

The goal of this guide is to serve as a detailed reference describing the features of ProductXpress Calculator.

Audience

This guide is intended for System Administrators and System Integrators.

Terminology

In this document, the following terminology is used:

- ▶ ProductXpress Calculator is referred to as Calculator
- ▶ ProductXpress Designer is referred to as Designer
- ▶ Calculator Console is referred to as Console
- ▶ Calculator Input XML is sometimes referred to as Calculator Request

Related Documents

- ▶ *ProductXpress Calculator Installation Guide*
- ▶ *ProductXpress Calculator User Guide*
- ▶ *ProductXpress Path Language Reference Guide*
- ▶ *ProductXpress Calculator Console Online Help*

Calculator Overview

In this section the Calculator is introduced. It looks at the different Calculator models and offers a general description that is described in detail in the remaining chapters. It looks in general at the difference between the Push and the Pull Calculators and also takes a brief look at the difference between the Embedded Calculator, the Service Calculator and the Function Calculator.

For technical details about these Calculators, please refer to the remainder of this guide.

For a more detailed description of the Calculator architecture and usage, please refer to the *ProductXpress Calculator User Guide*.

Introduction to Calculator

The ProductXpress Calculator is delivered with two different implementation possibilities. Depending on how the client application will communicate with the Calculator, it can be implemented as either:

- ▶ A Service Calculator, running as a standalone service.
- ▶ An Embedded Calculator, where the Calculator is embedded into an existing application.

The Push-model and the Pull-model

The Calculator supports two types of model. The Service Calculator supports the push model, the Embedded Calculator supports both the push and the pull model. Each model is described as follows.

Push-model

When using the *push*-model, a client has to provide all required data beforehand (as is the case for the Service Calculator). The calculation requests are XML formatted requests. These requests have the same format that is used for the Service Calculator; see Common Items for the Service and Embedded Calculator (see "[Common Items for Service and Embedded Calculator](#)" on page 191).

Pull-model

When using the *pull*-model, a client interfacing with the Calculator does not have to provide all instances with their attribute values beforehand (as is the case with the *push*-model). The client just requests what features to calculate and the Calculator will incrementally *pull-in* the required data through a callback interface (to be implemented by the client).

Every instance is identified by an instance ID and a definition ID that uniquely identifies the definition for the instance.

The instance ID is an ID that is generated by the client in such a way that it uniquely identifies an instance of a definition from the product structure.

The definition ID uniquely identifies a definition from the product structure, like a product definition, a coverage definition, an attribute definition, etc. It is used by the Calculator for internal lookup of a definition, and therefore it is an internal calculator id. The definition ID is a character string representation of a guide. The path format is a user friendly way to identify a definition (see the *ProductXpress Path Language Reference Guide*). An interface is provided to enable the client to perform the translation from the path format to the guide format that must be used as the definition id.

Service Calculator

The Service Calculator can be used through three types of interface:

- ▶ TCP/IP
- ▶ SOAP
- ▶ COM

The Service Calculator works according the so called *push*-model, meaning that in the calculation request all necessary input data is contained as well as the attributes that must be calculated and the instance information.

The strength of the Service Calculator lies in high volume calculations.

Embedded Calculator

There are three separate types of embedded implementations available.

Java Interface

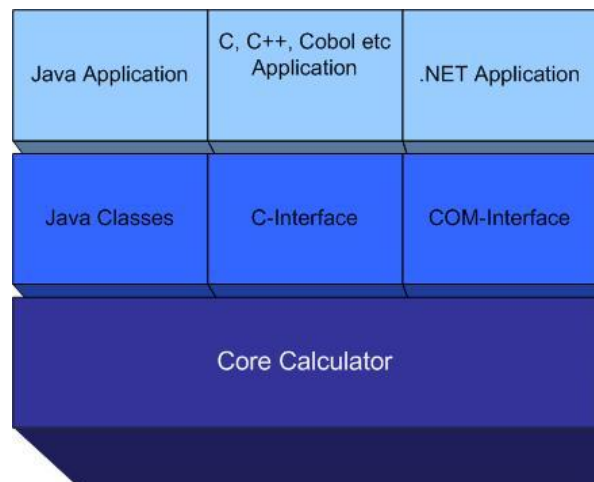
This consists of three jar files (pxjavaruntime.jar, FiaJNI.jar and PxVal.jar) containing Java classes to interface with Calculator, and a dll/shared library (CalculatorJNI.dll, libCalculatorJNI.so on AIX and Solaris) that contains the core Calculator and the JNI interface used by the Java classes.

C Interface

This consists of a single dll/shared library (Calculator.dll, libCalculator.so on AIX and Solaris) that contains the core Calculator and implements a C interface for Calculator.

.NET Interface

This consists of a COM dll (CalculatorCOM.dll, only available on Windows) that contains the core Calculator and a COM interface that can be used from the .NET platform.



The Embedded Calculator can work according the so called *push*- or *pull*-models.

When using the *pull*-model, the calculation request contains only information about what must be calculated. Required input attribute values or product structure instance information is requested by Calculator through a callback mechanism.

When using the push-model, the calculation request contains all information needed to perform the requested calculations. This includes values of entered attributes and product structure instance information.

The Embedded Calculator is tightly integrated with the client application.

Function Calculator

The Function Calculator can be useful in lightweight systems, e.g. a Web quotation system, in which calculations do not require complex data structures or a financial product model. The Function Calculator is available in both the Service and the Embedded Calculator with a push model. It is not itself a separate Calculator, but merely a mode supported by Calculator.

The Service Calculator

This section looks at the Service Calculator from a technical perspective.

The configuration and maintenance of the Service Calculator are described. It gives details of how to configure and tune the Service Calculator through the settings files after installation. It further describes the options for the management of the deployment packages using depimp, the execution of the Calculators using calcservertest and the management of external tables using tableimp.

It also describes how to interface the Calculator with an external client application. It describes the following interface options:

- TCP/IP Protocol
- SOAP protocol
- 4.2.6 Web Services Description Language (WSDL)
- COM
- The transform Interface using either Perl, C, SOAP and WSDL
- The Runtime Configuration Interface using SOAP and WSDL

Also outlined are considerations during runtime such as decryption of deployment packages, multithreading possibilities and the use of multiple instances of Calculator on a single machine.

Service Calculator Configuration and Maintenance

Configuration of the Service Calculator is done through the following files:

- Operational settings files
- Error Conversion configuration settings file
- ExternalFunctions.Settings file

These files are loaded on startup of the service Calculator (or on first usage of the configuration information) and therefore Calculator must be restarted after changing the configuration.

Operational Settings Files

The Service Calculator can be configured through the **.Settings** files, which are located in the **Etc** directory of the installation. The **.Settings** files are saved in XML¹ format.

The main element of a **.Settings** file has the same name as the file name without the extension.

Each "setting" is represented by an XML-element in the **.Settings** file. The value is specified by the "value" attribute .

Each "setting" can have sub-settings. A sub-setting is a child element of the XML-element representing its parent setting. It also has a "value" attribute to specify its value.

Example: **Calculator.Settings** file, containing setting "Trace" with value "Clc", "break-on-error" is a sub-setting of "Options" and has the value "true".

```
<?xml version="1.0"?>
<Calculator>
  <Trace value="Clc"/>
  <Options>
    <break-on-error value="true"/>
  </Options>
</Calculator>
```

```
</Options>
</Calculator>
```

The Calculator Service can be configured through the following operational settings files:

- Calculator.Settings
- Deplmp.Settings
- TableImp.Settings

Calculator.Settings

Setting Name	Description	Default Value
MaxClients	It specifies the maximum number of clients that can be connected to the server at the same time. If more clients try to connect, the server will close the least recently used connection.	32
MaxIdleWorkers	It specifies the number of calculation threads that is kept alive, while there is no work for them to do.	MaxWorkers
MaxWorkers	It specifies the maximum number of concurrent worker threads. Each worker thread handles a Request.	Number of CPUs
ListenPort	It specifies the ID number of the TCP port where the server waits for new clients to connect.	30001
Listen Address	Addresses used to listen for incoming requests. There can be multiple entries of listen addresses defined.	ANY ("0.0.0.0"(ipv4) :: (ipv6))
Hostname	The hostname to be used by the calculator.	Default hostname of machine.
Options overwrite the defaults for Calculation Input options normally specified on the clc:CalculationInput tag in the XML input for the Calculator (refer to Error behaviour attributes (on page 217)).	Options can have the following sub-settings: Error behaviour options: ignore-input-structure-errors ignore-output-structure-errors check-link-multiplicity check-input-values check-output-values ignore-input-mapping-errors ignore-output-mapping-errors break-on-error Calculator Output options: timer	false false false false false false false false

Setting Name	Description	Default Value
	input-in-output	false
RealPrecision	It specifies the number of precision digits to be used in the calculations. Varies from 10-16. It can be used in combination with the type of rounding method, specified under the setting <i>RealRoundMethod</i> . For example: <RealPrecision value="11"/>	14
RealRoundMethod	It specifies the rounding method to be used in the calculations. Possible values: "up", "round", "down", "floor", "ceiling", "banker". It can be used in combination with the precision value, specified under the <i>RealPrecision</i> setting. For example: <RealRoundMethod value="down"/>	round
SocketReadTimeout	It specifies the timeout for reading of calculator input data on the socket in seconds. If no data is received within the configured period an error is logged and the treatment of the current request is aborted. A value of 0 means no timeout is to be used.	30
DefaultURIPrefix	Specifies the URI prefix (URI part without deployment name) used when deploying deployment packages with either Deplmp or Calculator Console.	"http://www.example.org/"
Trace	It logs extra information. For further details, see Tracing (on page 277).	-

Setting Name	Description	Default Value
TraceFile	It specifies the filename, relative to the working directory, where the tracing output is placed. Note that this file is created or overwritten at startup of Calculator, and should not be moved or deleted when Calculator is running.	-
Mode	It specifies the mode in which Calculator runs; the following two modes are available: test: The Test Mode of Calculator allows packages of checked out deployments to be loaded and tested in Calculator. production: The Production Mode of Calculator allows packages of checked in deployments to be loaded and tested in Calculator.	production
LogRequestErrors	If true, errors in calculations are logged when in the calculation request the attribute "break-on-error" is set to true.	true
FinishCalcsOnShutdown	When all worker threads are busy, the shutdown takes place when the shutdown request can be handled by an available worker thread, until then it is queued. When this is set to true, when the Calculator service is stopped then the current calculations will complete before shutdown.	true
ProductCacheUpperBound	It specifies the maximum size of the Global cache. When the Global cache exceeds this size, it is reduced to the <i>ProductCacheLowerBound</i> size. The size of the Global cache is checked at regular intervals.	70000
ProductCacheLowerBound	It specifies the size to which the Global cache is reduced, when it exceeds the <i>ProductCacheUpperBound</i> size.	50000
ProductCacheCheckInterval	It determines the interval, in seconds,	5 seconds

Setting Name	Description	Default Value
	between the checks on the size of the Global cache.	
PolicyCacheUpperBound	It specifies the maximum size of the Policy cache. When the Policy Cache exceeds this size, it is reduced to the <i>PolicyCacheLowerBound</i> size. The size of the Policy Cache is checked each time an element is added. Policy cache: contains items of which the value can change between instanced of a product; this cache is emptied after each request.	1000000
PolicyCacheLowerBound	It specifies the size to which the Policy cache is reduced, when it exceeds the <i>PolicyCacheUpperBound</i> size.	800000
NoProductCache	When this setting has the value "true", no Global cache is used.	false
ProductCacheCleanup	It determines if the ProductCache size is kept within bounds.	true
ProductCacheAgingFactor	It ensures that values that are not reused anymore will be removed eventually. Increase this value if you suspect that values that were cached in the beginning are preventing other, more reuseable values from staying cached. Value may be set between 0 and 100.	10
Cacheexternalfunctions	It enables automatic creation of wrapper functions around function variables. This is relevant for function variables mapped on an external function. The wrapper introduces a cache which can (in some circumstances) avoid calls to the external function. In order to switch off the caching of external functions use: <Optim> <cacheexternalfunctions value="0" /> </Optim>	1
Strict	When Calculator optimizes, it sometimes optimizes away errors. This means that results are returned even if the functions, when evaluated	0

Setting Name	Description	Default Value
	<p>in the way they were defined, would return an error. The strict mode is intended to force the Calculator to throw errors in these circumstances:</p> <pre><Optim> <strict value="1"/> </Optim> </Calculator></pre>	

Deplmp.Settings

Setting Name	Description	Default Value
Trace	It logs extra information. For further details, see Tracing (on page 277).	-
TraceFile	It specifies the filename, relative to working directory, where tracing output is placed.	-

TableImp.Settings

Setting Name	Description	Default Value
Trace	It logs extra information. For further details, see Tracing (on page 277).	-
TraceFile	It specifies the filename, relative to working directory, where tracing output is placed.	-

Deployment Package Management

There are four types of deployments supported by ProductXpress.

Static Deployment

A static deployment package is a representation of the financial product, the target environment (for example an administration system), and the mapping between the product and the environment.

Dynamic Deployment

A dynamic deployment package is a representation of the financial product.

-
- i** Static and dynamic deployments are also referred to as entry point deployments. From these deployments, deployment packages can be created in Designer that then are loaded into Calculator.
-

Type Deployments

A type deployment package is a representation of Subtypes of structure elements.

Function Deployment

A function deployment package is a representation of the set of functions that are deployed.

-
- ❗ The deployments are "packaged" into a file with extension .pxdpz (a package archive file) or .pxdp (a single package file). A package archive file contains (among others) a file with extension .pxdp. A .pxdp file is formatted according to the ProductXpress XML standard for deployment packages. A .pxdpz or .pxdp file has been generated by ProductXpress Designer and it is loaded into the ProductXpress Calculator. The Deployment Package Management works the same for all types of deployments.
-

Deploying with Deplmp.exe

Deplmp.exe is a command line tool used for deployment package management. Deplmp enables:

- ▶ Addition of a deployment package to the list of known deployment packages for Calculator
- ▶ Removal of a deployment package from the list of known deployment packages for Calculator
- ▶ Configuration of the type deployments to be used by an entry point deployment
- ▶ Configuration of selectors for external resource roots
- ▶ Listing of all deployment packages from the list of known deployment packages for Calculator
- ▶ Provision of details about a deployment package file

-
- ❗ The Calculator console tool provides a graphical user interface to some of the operations that are performed by Deplmp. For further information, see the Calculator Console's online help.
 - ❗ Deplmp operates on a local Calculator. Therefore the local Service Calculator must be running.
-

Deplmp Syntax

Deplmp uses the following syntax for managing deployment packages:

```
DepImp
-add=<local deployment file>
[-name=<name>] [-actdate=<date>] [-documentation=<documentation>]
[-branchRename=<origName>:<newName>,<origName>:<newName>,...]
[-uri=<URI for configuration>] [-nopreoptimize]
-list [-details]
-remove -id=<configuration id> | -name=<deployed name>
[-version=<version>]
[-type=entryPoint|type|function]
```

```

-modify -id=<configuration id> [-documentation=<new documentation>]
[-actdate=<new date>] [-name=<new name>]
[-uri=<new URI>] [-preoptimize] [-nopreoptimize]
-repair
-check -id=<configuration id>
-getfile -filename=<target file> -id=<configuration id>
-addtype -id=<configuration id> -typeId=<configuration id>
-removetype -id=<configuration id> -typeId=<configuration id>
-setselector -id=<configuration id> -root=<root id>
-name=<deployed name> [-date=<date> | -version=<version>]
-resetselector -id=<configuration id> -root=<root id>
-describe=<local deployment file>
-keylist
-keyimport=<key export file>
-keyremove=<key label>
-keyrelabel=<old label> <new label>
-suspend
-resume
-shutdown
-server

```

i This syntax is listed when running Deplmp without arguments.

add

```

-add=<local deployment file> [-name=<name> [-server=<instance
name>]] [-actdate=<date>]]
[-documentation=<documentation>]
[-branchRename=<origName>:<newName>,<origName>:<newName>,...]
[-uri=<URI for configuration>]

```

Add deploys a package. In order to use correctly,

- ▶ supply the filename of the package including the path (note that this file must be accessible by the Calculator. As the Calculator is running as a service by a different user account, it may be the case that this user does not have access rights to the same files/directories as the user that executes Deplmp)
- ▶ use double quotes if the filename contains spaces (e.g. -add="xxx\Program Files\yyy")

This option can be used in combination with the arguments **name**, **actdate**, **documentation**, **branchRename**, **uri** and **nopreoptimize**:

- ▶ **name** is the deployment name that is referenced by Requests; by default the name of the deployment as defined in Designer is used.
- ▶ **server** is the instance on which the Calculator is running. It is only required when deplmp is used on any instance other than the default.
- ▶ **actdate** is the deployment active date; by default the creation date of the deployed version of the deployment component is used.
- ▶ **documentation** is the documentation for the deployment package to add.
- ▶ **branchRename** makes it possible to solve conflicts between branch names of already added deployments and the branch name(s) contained in the deployment to add.

- **uri** is the XML namespace URI for the deployment; by default it is the URI as provided at the creation of the deployment package in Designer or "*http://www.example.org/<name of the deployment component>*" when non is provided in designer>, where non NCName³ characters in the deployment component name are replaced by "_".
- **nopreoptimize** is used to optimize the calculations of the deployment package at first use by a calculation request; by default the calculations are optimized at Calculator startup.

Examples:

```
DepImp -add=myDeploymentPackage.pxdp
DepImp -add="D:\Program Files\myDeploymentPackage.pxdp"
DepImp -add=myDeploymentPackage.pxdp -name=newname
        -actdate=1990-01-01 -server=instancename
DepImp -add=myDeploymentPackage.pxdp
branchRename=origName:newName,anotherOrigName:anotherNewName
```

list

```
-list [-details] [-server=<instance name>]
```

This argument lists information about deployment packages that have been added to the Calculator. The argument **details** provides the following additional details about entry point deployments: the used type deployments and the external resource roots.

Examples:

```
DepImp -list
This results in:
Loaded entry point deployments configurations:
Configuration Id      : 305dcd74-8559-4f1f-a6f8-0b7a0a9f368d
Name                  : statisDeployment1
Version               : 0.1
Active Date           : 2004-08-01
Documentation          : Some comment
```

Loaded type deployments configurations:

```
Loaded function deployments configurations:
Configuration Id      : 4effbe65-d8a1-498b-a175-6ed97f4026e7
Name                  : FunctionDeployment
Version               : 0.8
Active Date           : 2006-04-07
Documentation          : Some comment
```

```
DepImp -list -details
This results in:
```

```
Loaded entry point deployments configurations:
Configuration Id      : 305dcd74-8559-4f1f-a6f8-0b7a0a9f368d
Name                  : statisDeployment1
URI                   : statisDeployment1
Version               : 0.1
Active Date           : 2004-08-01
Documentation          : Some comment
External Resource Roots:
Used type deployments:
```

Loaded type deployments configurations:

Loaded function deployments configurations:

```
Configuration Id      : 4effbe65-d8a1-498b-a175-6ed97f4026e7
Name                  : FunctionDeployment
URI                   : FunctionDeployment
Version               : 0.8
Active Date           : 2006-04-07
Documentation          : Some comment
```

remove

```
-remove -id=<configuration id> | -name=<deployed name>
[-version=<version>] [-server=<instance name>]
[-type=entryPoint|type|function]
```

Remove removes a deployed package. The supplied ID is the configuration ID of the deployed package file as listed with the **list** argument. As an alternative, the name of the deployed package and optionally the type (if name alone is not unique) can be provided to remove a deployed package.

modify

```
-modify -id=<configuration id> [-documentation=<new documentation>]
[-actdate=<new date>] [-name=<new name>] [-uri=<new URI>]
[-preoptimize] [-nopreoptimize] [-server=<instance name>]
```

Modify enables the user to modify some attributes of the deployed deployment package. The supplied **ID** is the configuration ID of the deployed package file as listed with the **list** argument. The new values for the provided attributes to change will replace the current values for these attributes.

check

```
-check -id=<configuration id> [-server=<instance name>]
```

Checks the configuration with the provided ID, which is the ID of the deployed deployment package. The result of check can be the following:

- ▶ not valid
In this case the configuration can only be used to calculate when Calculator is in 'production' mode.
- ▶ valid
In this case the configuration can be used, it could contain warnings.

getfile

```
-getfile -filename=<target file> -id=<configuration id>
[-server=<instance name>]
```

Getfile extracts the deployment package from the runtime configuration with the provided ID and stores it in the "target file".

repair

Repair repairs a corrupted runtime configuration of Calculator.

addtype

```
-addtype -id=<configuration id> -typeId=<configuration id>
[-server=<instance name>]
```

Addtype adds a type deployment configuration to an entry point configuration (only allowed for dynamic entry point deployments). As a result subtypes from the added type deployment configuration can be used by the entry point configuration.

removetype

```
-removetype -id=<configuration id> -typeId=<configuration id>
[-server=<instance name>]
```

Removetype removes a type deployment configuration from an entry point configuration. As a result the subtypes from this type deployment can no longer be used by the entry point configuration.

setselector

```
-setselector -id=<configuration id> -root=<root id> -name=<deployed
name>
[-date=<date> | -version=<version>]
[-server=<instance name>]
```

Setselector selects an entry point deployment configuration for an external resource root. As a result the selected entry point deployment configuration is used for the external resource root.

- **ID** is the ID of the entry point deployment configuration that defines the external resource root.
- **root** is the ID of the external resource root.
- **name** is the name of the entry point deployment configuration that has to be selected.
- **date** is the active date of the entry point deployment configuration that has to be selected.
- **version** is the version of the entry point deployment configuration that has to be selected.

resetselector

```
-resetselector -id=<configuration id> -root=<root id>
[-server=<instance name>]
```

Resetselector resets the selection of an entry point deployment configuration for an external resource root. As a result there is no longer an entry point deployment configuration selected for the external resource root.

- **ID** is the ID of the entry point deployment configuration that defines the external resource root.
- **root** is the ID of the external resource root.

describe

```
[-server=<instance name>]
```

Describe gives details about a local (not deployed) deployment package.

Suspend, resume and shutdown with Deplmp

The Calculator can be suspended, resumed and shutdown with the Deplmp tool.

Deplmp Syntax

```
DepImp
```

```
-suspend  
-resume  
-shutdown
```

suspend

```
[-server=<instance name>]
```

Suspend suspends the Calculator. After this, new calculation requests are not processed anymore. The returned result contains a "down for maintenance" error message.

resume

```
[-server=<instance name>]
```

This resumes a suspended Calculator. The Calculator caches are cleared and new calculation requests are processed again.

shutdown

```
[-server=<instance name>]
```

This shuts down the Calculator.

External Table Management

Deployment packages can have references to tables that are external to the deployment package (non external tables are tables that are contained in the deployment package). These external tables are created via ProductXpress Designer.

To enable the ProductXpress Calculator to use an external table, it must first be deployed for the ProductXpress Calculator. Deployment of external tables is done by means of the TableImp command line tool.

External Table Details

Each external table contains the following details:

Table Name

Within the Table Name field, the name of the External Table is displayed. The name of the External Tables as defined in Designer is used.

Documentation

Within the Documentation field, the description of the External Table, as created within Designer is displayed.

Preload

The Preload field indicates whether an External Table is preloaded by the Calculator at start up. An external table is always preloaded automatically when it is deployed to a Database.

Database Location

If the External Table has been deployed to a Database, then the Database Location field displays the path to the database where the External Table is deployed.

Cache Size

The Cache Size field displays the cache in the database, and allows the user to determine how many table rows are cached.

Created By

The Created By field displays the username of the user that has created or modified last the External Table.

Creation Time

The Creation Time field displays the time that the External Table was created.

Deployment time

The Deployment Time field displays the time that the External Table was deployed.

Using TableImp.exe

TableImp.exe is a command line tool used for external table management. TableImp enables:

- ▶ Addition of an external table to the external tables that are deployed for the Calculator. An external table can be deployed to a local file for the Calculator or to a database.
- ▶ Removal an external table from the deployed external tables for the Calculator.
- ▶ Listing of all deployed external tables for the Calculator.
- ▶ Modification/Update of an already deployed external table.
- ▶ Repair of the external table configuration for the Calculator, if the configuration has become inconsistent.

i The Calculator console tool provides graphical user interface to some of the operations that are performed by TableImp. For further information, refer to the Calculator Console's online help.

TableImp operates on a local Calculator. With Calculator Console, external tables of remote Calculators can also be managed.

TableImp Syntax

TableImp uses the following syntax:

```
TableImp
-add=<local external table file> [-documentation=<documentation>]
[-preload=true|false]
[-database=<database spec>[-cachesize=<number of rows to
cache>][-name=<new name>]]
[-csv=<local csv file>]
[-server=<instance name>]

-list [-long] [-order]

-remove=<calculator deployed external table id>

-update=<local external table file> [-csv=<local csv file>]

-modify=<calculator deployed external table id>
```

```
[-cachesize=<new number of rows to cache>][-documentation=<new documentation>]
```

```
-repair
```

add

```
-add=<local external table file> [-documentation=<documentation>]
[-preload=true|false]
[-database=<database spec>]
[-cachesize=<number of rows to cache>]
[-name=<new name>]]
[-csv=<local csv file>]
[-server=<instance name>]
```

Add adds external table data to the external table configuration of Calculator.

External table data can be stored in a local file for Calculator (no **-database** option) or in a database (with **-database** option).

Additional options are:

- ▶ **documentation:** Stores the provided text as documentation for the deployed external table.
- ▶ **preload:** Indicates if the external table data is to be loaded by the Calculator during startup ('true') or when the external table is first used in a calculation ('false'). The default value is 'true'.
- ▶ **database:** Store the external table data in a database that is specified with the provided '<database spec>'.
 - ◆ <driver>:<user>[/<password>]@[<server>]:[<schema>]
 - ◆ <driver> 'Db2' for a DB2 database, 'MsSql' for a MsSQL database, 'MySql' for a MySQL database or 'Oracle' for an Oracle database.
 - ◆ <user> specifies a database authorization-name (user identifier).
 - ◆ <password> specifies the corresponding authentication-string (password) (optional).
 - ◆ <server> specifies the Data Source: the name or alias-name of the database.
 - ◆ <schema> value indicates the particular schema to be used in the database (optional). If omitted the schema of <user> is used.

Example:

```
-database=Db2:db2admin/db2admin@PX2:CalcServer
```

- ▶ **cachesize:** Sets the number of rows that is cached for external table data that is stored in a database. This option can only be used in combination with the **-database** option. The default value is 5.
- ▶ **name:** Use the provided name as the table name in the database in which the external table data must be stored. By default the external table name as defined in the provided external table file is used.
- ▶ **csv:** Use the local csv file instead of the local external table file to read the external table data from; the local external table file and the local csv file must have matching table

signatures (i.e. the number, the order, the names and the data types of dimensions and columns must match); the local external table file must be empty, i.e. it must not contain non-null dimension or column values.

- **server:** is the instance on which the Calculator is running. It is only required when `deplmp` is used on any instance other than the default

list

List lists info on the external tables that are present in the external table configuration of the Calculator. Without the **-long** option only the ID and name of the external tables are listed. With the **-long** option the documentation and preload option are also listed. If the table resides on an instance, then the **-server** should be used in order to specify the instance name.

Examples:

```
TableImp -list
```

Results in:

```
id          : 55369b95-ed7f-ffff-ffff-ffffffffffffff
name        : externalTable test
id          : 330d63cf-49ec-4929-aa69-8a5e3805658c
name        : extable
```

```
TableImp -list -long
```

Results in:

```
id          : 55369b95-ed7f-ffff-ffff-ffffffffffffff
name        : externalTable test
Documentation : user documentation
Pre-load    : true
id          : 330d63cf-49ec-4929-aa69-8a5e3805658c
name        : extable
Documentation : user documentation
Pre-load    : true
```

remove

Remove removes external table data from the external table configuration of the Calculator. Use the ID of the table as listed with the **-list** option. If the table resides on an instance, then the **-server** should be used in order to specify the instance name.

Example:

```
TableImp -remove=330d63cf-49ec-4929-aa69-8a5e3805658c
```

-
- ❗ For external tables that were added to a database, the table in the database that holds the data of the external table is dropped.
-

update

```
-update=<local external table file> [-csv=<local csv file>]
```

Update updates the data of an external table with the table data that is provided in the external table file. Optionally the **-csv** argument can be specified (see the **-add** option for a description of the **-csv** option). If the table resides on an instance, then the **-server** should be used in order to specify the instance name.

Example:

```
TableImp -update=MyExternalTable.pxet
```

modify

```
-modify=<local external table file> -cachesize=<number of rows to cache>
```

Modify modifies the number of cache rows for an external database in memory.

- ▶ **cachesize:** Sets the number of rows that is cached for external table data that is stored in a database.
- ▶ **documentation:** Stores the provided documentation as documentation for the deployed external table (replaces existing documentation).
- ▶ If the table resides on an instance, then the **-server** should be used in order to specify the instance name.

repair

Repair tries to repair inconsistencies in the external table configuration. If the table resides on an instance, then the **-server** should be used in order to specify the instance name.

Loading CSV Files on Deployment of External Tables

By using the Calculator Management Console (the 'Add Table ...' menu entry) or the TableImp command line tool (the **-csv** option), it is possible to deploy an (empty) External Table together with a local CSV file. This will result in a deployed External Table with its table data being loaded from the CSV file.

Note that reading External Table values from a CSV file is subject to a number of limitations, like: newline characters in values are not supported (reading a CSV file is done per line), the value separator (a.k.a. list separator) must consist of exactly one character and sets and ranges in dimension values are not supported.

Also note that it is possible for the user to provide one or more settings which can help in successfully reading a CSV file in which list separators or date/time values are present in a format that does not match the current locale settings. The settings regarding date/time are strings containing zero or more conversion specifiers (see below for a list of supported conversion specifiers). The following settings can be specified in a settings file called 'TableDataCSVLoader.Settings':

name	description	default value (Windows)	default value (Unix)
decimal-point	string consisting of exactly one character	locale setting	locale setting
thousands-separator	string consisting of exactly one character	locale setting	locale setting
list-separator	string consisting of exactly one character	locale setting	","
date-time-format	date and time conversion specification	"%x %X"	"%x %X"
time-format	time conversion specification	deduced from	"%H:%M:%S"

name	description	default value (Windows)	default value (Unix)
	(e.g. "%H:%M:%S")	locale setting	
date-format	date conversion specification (e.g. "%Y/%m/%d")	deduced from locale setting	"%m/%d/%y"
am-designator	am designator string (e.g. "AM")	locale setting	"AM"
pm-designator	pm designator string (e.g. "PM")	locale setting	"PM"
day-name[0-6]	name of day (e.g. day-name0 is "Monday")	locale setting	"Sunday", etc.
short-day-name[0-6]	short name of day (e.g. short-day-name0 is "Mon")	locale setting	"Sun", etc.
month-name[0-11]	name of month (e.g. month-name0 is "January")	locale setting	"January", etc.
short-month-name[0-11]	short name of month (e.g. short-month-name0 is "Jan")	locale setting	"Jan", etc.

On Windows, the default date/time settings are deduced from the locale date/time settings. For example, in case the locale setting for the time format is "h:mm:ss tt", it will result in a time conversion specification "%I:%M:%S %p". See below for a list of supported conversion specifiers. On UNIX, all but *decimal-point* and *thousands-separator* settings have fixed default values.

The following date/time conversion specifiers can be used in the date/time format settings mentioned above. Note that these conversion specifiers are a subset of the ones supported by the 'strftime'/'strptime' Standard C Library functions.

conversion specifier	Description
%A	is replaced by national representation of the full weekday name.
%a	is replaced by national representation of the abbreviated weekday name.
%B	is replaced by national representation of the full month name.
%b	is replaced by national representation of the abbreviated month name.
%c	is replaced by national representation of time and date.
%d	is replaced by the day of the month as a decimal number (01-31).
%H	is replaced by the hour (24-hour clock) as a decimal number (00-23).
%I	is replaced by the hour (12-hour clock) as a decimal number (01-12).
%j	is replaced by the day of the year as a decimal number (001-366).

conversion specifier	Description
%M	is replaced by the minute as a decimal number (00-59).
%m	is replaced by the month as a decimal number (01-12).
%p	is replaced by national representation of either "ante meridiem" or "post meridiem" as appropriate.
%S	is replaced by the second as a decimal number (00-60).
%U	is replaced by the week number of the year (Sunday as the first day of the week) as a decimal number (00-53).
%W	is replaced by the week number of the year (Monday as the first day of the week) as a decimal number (00-53).
%w	is replaced by the weekday (Sunday as the first day of the week) as a decimal number (0-6).
%X	is replaced by national representation of the time.
%x	is replaced by national representation of the date.
%Y	is replaced by the year with century as a decimal number.
%y	is replaced by the year without century as a decimal number (00-99).

Managing Secured Sockets

Secured sockets can be managed with the tool **SSLConf**. This tool allows you to:

- ▶ Activate/deactivate the use of secured sockets for communication with the Calculator server.
- ▶ Configure the use of private keys and certificates (generate or import).

SSLConf Syntax

```
-on [-verify=none|peer] [-newkey=<certificate authority certificate
output filename> [-days=<number of days certificate valid>]
[-bits=<number of bits in key>]]
| [-cert=<certificate input filename> -key=<private key input
filename> -CAcert=<certificate authority certificate input
filename>]
-off
-newkey=<certificate filename> [-days=<number of days certificate
valid>] [-bits=<number of bits in key>]
-cert=<certificate input filename> -key=<private key input filename>
-CAcert=<certificate authority certificate input filename>
```

on

On switches the use of secured sockets on (make sure a key is configured (see **newkey** option)).

The following sub options can be used:

- ▶ **verify**: The verification mode can be **none** (no peer verification) or **peer** (verification of peer). The default value is **none**.
- ▶ For the other options, see option **newkey**.

off

Off switches use of secured sockets off.

newkey

Newkey generates or installs a new key and certificate. Provide the name of the output file to which the certificate authority certificate is written. This file can then be used to import in clients of the Calculator, like the Calculator Console.

The following sub options can be used:

- ▶ **days**: The number of days the created certificate will be valid. The default is 30 days.
- ▶ **bits**: The size of the key in bits. The default is 1024 bits. Keys using less than 1024 bits are considered insecure.
- ▶ **cert**: Instead of generating a new certificate, use the one in the provided file.
- ▶ **key**: Instead of generating a new private key, use the one in the provided file.
- ▶ **CAcert**: Instead of generating a new certificate authority certificate, use the one in the provided file.

❗ Certificates and key provided in files must use the PEM encoding, which is a binary encoding using Distinguished Encoding Rules (DER) translated into an ASCII form by using base64 encoding.

Certificates and keys are generated using the PEM encoding.

Test Mode of Calculator

A production Calculator uses deployment packages that are based on checked in deliverables which have passed the validation. This is done to ensure the consistency of the deliverable and to link the calculation results to a definite set of calculations that cannot be changed. However, testing a deployment package often results in changes to the calculations and retesting. Calculator under test mode accepts deployment packages while ignoring the status of the source deliverable. In order to avoid the need to keep on upgrading and checking in deliverables during the testing phase.

When Calculator is started in Test mode, the deployment packages may belong to checked out deployments or to checked in deployments which have not passed the validation. This eliminates the need to check in the deliverable each time a deployment package based on that deliverable is tested.

The Calculator can be configured to start in test mode through the Mode setting in the Calculator.Settings.

Best Practices for Test Mode of Calculator

Loading multiple test deployment packages can lead to errors in Calculator.

Calculator cannot distinguish between checked out versions of the same deliverable, therefore you cannot load two different test deployment packages of the same product. Furthermore it may lead to errors when test deployment packages of different products use different versions of the same checked out deliverable.

You should only load one test deployment package in the Calculator at a time (i.e. remove all other test deployment packages), otherwise you have to make sure that your test deployment packages use exactly the same versions of checked out deliverables.

The CalcServerTest.exe Testing Tool

It can be invaluable to test the created financial products as soon as possible from the functional or calculations perspective, without waiting for the completion of the technical integration efforts to link the production system(s) with the Calculator.

Irrespective of the status of the actual Calculator integration with the production system, independent tests can be performed for a deployment package utilizing the CalcServerTest tool that is delivered with Calculator.

CalcServerTest is a command-line utility that can be used to send requests to the Calculator. As mentioned before, it simulates the production system by making specific calls to Calculator that resides locally or on a remote server.

CalcServerTest Syntax

CalcServerTest can be used with the following options:

```
-r=<request filename> [-h=<hostname>] [-l=<listenport>]
[-o=<output filename>] [-c=<expected result filename>]
[-i=<iterations>] [-t] [-d] [-priority]
[-threads=<number of threads>]
-s=<script filename> [-h=<hostname>] [-l=<listenport>]
[-o=<output filename>] [-c=<expected result filename>]
[-i=<iterations>] [-t] [-d] [-priority]
[-threads=<number of threads>] [-D<lua global variable
name>=<value>] *

-xslt=<xslt filename> -r=<request filename>
[-params=<xslt parameters filename>] [-o=<output filename>]
-xslt=<xslt filename> -xml=<xml filename>
-name=<configuration name>
[-version=<configuration version> | -date=<configuration date>]
[-params=<xslt parameters filename>] [-o=<output filename>]
-help

-xslt1=<first xslt filename> -xslt2=<second xslt filename>
-xml=<xml filename> -name=<configuration name>
[-version=<configuration version> | -date=<configuration date>]
[-params1=<first xslt parameters filename>]
[-params2=<second xslt parameters filename>]
[-o=<output filename>]
-aggregation=<request filename> [-o=<output directory>]
[-priority=(Low|Medium|High)] [-verbose]

-ping
-version
```


CalcServerTest Options

The following outlines the CalcServerTest options:

-aggregation=<request filename>

When running CalcServerTest with the option `-aggregation`, the calculation request must contain one or more aggregation links. Each link has its own ID, as specified in the calculation request. For each aggregation link, there must be a subdirectory of the directory in which CalcServerTest is run. The name of the subdirectory must be equal to the ID. The subdirectory contains all aggregated resources (in the form of calculation requests) that are aggregated over the corresponding aggregation link.

When no output directory is specified, CalcServerTest will store the calculator output file in a subdirectory "ActualResults". "ActualResults" will have subdirectories for each aggregation link in which the calculator output files for the aggregated resources are stored.

The directory in which CalcserverTest is run can have no other subdirectories other than the ones corresponding to the aggregation links, "ActualResult", and the output directory specified with the `-o` option, if any.

-c=<expected result filename>

Compares the result with the expected result. Use in Combination with `-o`.

-D<global Lua variable name>=<value>

Defines a global variable for a Lua script. Use in combination with `-s` for Lua scripts.

-d

Disconnect after sending a request without waiting for the response.

-date=<configuration date>

The date of the configuration to use(only in combination with `-xslt`).

-h=<hostname>

Name or IP address of the machine where Calculator runs (default localhost)

-help

Display help information of CalcServerTest.

-i=<iterations>

Number of times that the Request or Testscript must be run. Default 1.

-l=<listenport>

TCP/IP port number of Calculator connection (default 30001)

-name=<configuration name>

The name of the configuration to use (only in combination with `-xslt`).

-o=<output filename>

Filename for Request result (default **Request.xml**). Use only in combination with `-r`.

-params=<xslt parameters filename>

The filename of the file that contains parameters for XSLT transformation (only in combination with `-xslt`).

-params1=<first xslt parameters filename>

The filename of the file that contains parameters for the first XSLT transformation (only in combination with `-xslt1`).

-params2=<second xslt parameters filename>

The filename of the file that contains parameters for the second XSLT transformation (only in combination with `-xslt2`).

-ping

Test connection with Calculator by sending the ping command (default option when `CalcServerTest` is used without parameters). If the Calculator is working correctly, this will return 'Ping Result OK'.

-priority

Use the provided priority for the request to the Calculator. Can have the values: Low, Normal, High (Normal priority is the default).

-r=<request filename>

Request to be calculated (default **Request.xml**). Exclusive with `-s`.

-s=<script filename>

Execute script file. The script file can be an XML formatted script file or a Lua script (<http://www.lua.org/> (<http://www.lua.org/>)) file.

-soap

Send request as SOAP message. If the request file does not contain a complete SOAP message but only the Calculator request, a SOAP message is constructed for the request. Otherwise the SOAP message from the request file is sent to the Calculator.

-ssl

Use secure sockets (uses the certificate as configured for the Calculator install to which the `CalcServerTest` belongs).

-t

Measure request time.

-threads=<number of threads>

Send the same request or script concurrently via multiple threads to a single Calculator (useful for stress testing and scaling, including multiple CPU environments).

-version

The ProductXpress version.

-version=<configuration version>

The version of the configuration to use (only in combination with `-xslt`).

-xml=<xml filename>

The filename of input XML to be used for XSLT transformation (only in combination with `-xslt`).

-xslt=<xslt filename>

The filename of the XSLT stylesheet (Name of file as located in the XSL directory of the Calculator installation).

-xslt1=<first xslt filename>

The filename of the first XSLT stylesheet (Name of file as located in the XSL directory of the Calculator installation). It is to be executed before the calculation, output of transformation should be a calculation request.

-xslt2=<second xslt filename>

The filename of the second XSLT stylesheet (Name of file as located in the XSL directory of the Calculator installation). It is to be executed on the result of the calculation.

Testscripts for CalcServerTest.exe

CalcServerTest can execute two types of scripts:

- XML formatted scripts
- Lua scripts

XML formatted scripts

An XML formatted test script for CalcServerTest has the following structure:

XML element	Meaning
CalcServerTest	Main element. Mandatory. Contains zero or more Connector elements.
Connector	Specifies a request. Contains one Request element (mandatory), one Result element (optional) and one Expected element (optional).
Request	The name of the file containing the request.
Result	Filename where result will be written.
Expected	Filename of expected result. Result will be compared with this file.

Example:

```
<CalcServerTest>
  <Connector>
    <Request>TestInput/Request.xml</Request>
    <Result>ActualResults/Result.xml</Result>
    <Expected>ExpectedResults/Result.xml</Expected>
  </Connector>
  <Connector>
    <Request>TestInput/issue5652.in</Request>
    <Result>ActualResults/issue5652.out</Result>
  </Connector>
  <Connector>
    <Request>TestInput/issue5050.in</Request>
  </Connector>
</CalcServerTest>
```

In this example, three Requests are calculated. The request files are located in the TestInput directory. The result of TestInput/Request.xml is written to file ActualResults/Result.xml and compared with ExpectedResults/Result.xml. The result of TestInput/issue5652.in is written to ActualResults/issue5652.out.

Lua scripts

The 5.1 version of the Lua script language (<http://www.lua.org/> (<http://www.lua.org/>)) can be used to create scripts to be executed by CalcServerTest. The language is extended with a PX library that makes specific PX functions available for use within scripts (see "px Lua library").

Other included libraries are:

LuaFileSystem (<http://www.keplerproject.org/luafilesystem/index.html> (<http://www.keplerproject.org/luafilesystem/index.html>))

Simple calculation example:

```
io.input("test.clcin");
request = io.read("*all");
print("Request=");
print(request);
result = px.calculate(request);
print("Result=");
print(result);
```

Asynchronous calculation example:

```
io.input("test.clcin");
request = io.read("*all");
print("Request=");
print(request);
id = px.asyncCalculate(request);

ready = px.ready(id);
while ready == false do
    ready = px.ready(id);
end

result = px.wait(id);

print("Result=");
print(result);

px.clearCalculatorDistrList();
px.addCalculatorToDistrList("MyName",
"http://localhost:30001/ClcSrv", 3);

id1 = px.asyncCalculate(request);
id2 = px.asyncCalculate(request);
id3 = px.asyncCalculate(request);

ready1 = px.ready(id1);
ready2 = px.ready(id2);
ready3 = px.ready(id3);
while ready1 == false and ready2 == false and ready3 == false do
    ready1 = px.ready(id1);
    ready2 = px.ready(id2);
    ready3 = px.ready(id3);
end

result1 = px.wait(id1);
print("Result1=");
print(result1);
```

```
result2 = px.wait(id2);
print("Result2=");
print(result2);
```

```
result3 = px.wait(id3);
print("Result3=");
print(result3);
```

XSLT transformation example:

```
io.input("test.clcin");
request = io.read("*all");
print("Request=");
print(request);
parameters = {};
parameters["key1"] = "value1";
parameters["key2"] = "value2";
result = px.transform(request, "listAllTypesAndDeployment.xml",
"AMF-AMF-VVO-GEEN-OFFERTE", "", "", parameters);
print("Result=");
print(result);
```

XSLT calculate and transform example:

```
io.input("test.clcin");
request = io.read("*all");
print("Request=");
print(request);
parameters = {};
parameters["key1"] = "value1";
parameters["key2"] = "value2";
result = px.calculateAndTransform(request,
"listAllTypesAndDeployment.xml", parameters);
print("Result=");
print(result);
```

XSLT transform calculate and transform example:

```
io.input("test.clcin");
xmlInput = io.read("*all");
print("XML Input=");
print(xmlInput);
parameters1 = {};
parameters1["key1"] = "value1";
parameters1["key2"] = "value2";
parameters2 = {};
parameters2["key1"] = "value1";
parameters2["key2"] = "value2";
result = px.transformCalculateAndTransform(xmlInput,
"createRequest.xml",
"afterCalculate.xml", "AMF-AMF-VVO-GEEN-OFFERTE", "", "",
parameters1, parameters2);
print("Result=");
print(result);
```

XSLT parameter file for CalcServerTest.exe

A parameter file for CalcServerTest has the following structure:

XML element	Meaning
Parameters	Main element. Mandatory. Contains zero or more Parameter elements.
Parameter	Specifies a parameter for XSLT processing. Contains a name attribute. This name attribute has as value the name of the parameter. The content of the element is the value of the parameter. Note that the string parameter values should be placed between single quotes. See the following example.

Example:

```
<?xml version="1.0"?>
<Parameters>
  <Parameter name="myName">'My value'</Parameter>
</Parameters>
```

px Lua library

This section describes the PX Lua library functions. Note that all string parameters should be UTF8 encoded.

px.addCalculatorToDistrList(name, url, nbConcurrentCalcs)

Adds a Calculator to the list of Calculators for distributed calculations.

The **name** parameter is the name with which this Calculator is registered (must be unique).

The **url** parameter defines how the Calculator can be reached.

Format:

```
<protocol>://<host>:<port>/ClcSrv
```

Where :

<protocol> is the protocol to be used, can be fia (px specific protocol), sfia (px specific protocol over secured socket connections), http (uses SOAP formatted requests) or shhttp (uses SOAP formatted request over secured socket connections).

<host> is the host name or ip address of host on which the Calculator is running.

<port> the port number used by the Calculator.

The **nbConcurrentCalcs** parameter defines the maximum number of concurrent calculation requests sent to the Calculator.

px.asyncCalculate(request, priority)

Starts an asynchronous calculation request for the provided Calculator Input XML in the **request** parameter and returns the ID of the asynchronous calculation. The **priority** parameter is optional and is an integer which can have the following values: (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

px.calculate(request, priority)

Performs a calculation with the provided **request** (Calculator Input XML) and returns the result of the calculation (calculator Output XML). The **priority** parameter is optional and is an integer which can have the following values: (0=default priority, 1=low priority, 2=normal priority,

3=high priority).

The settings files NamedObject.Settings and ServerURL.Settings are used to determine how to contact the Service Calculator.

px.calculateAndTransform(request, xslFile, parameters, priority)

Performs a calculation with the provided **request** (Calculator Input XML) and performs an XSLT transformation on the result of the calculation (calculator Output XML) using the style-sheet with the provided **xslFile** name and the provided parameters in **parameters**. The **priority** parameter is optional and is an integer which can have the following values: (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

px.clearCalculatorDistrList()

Clears the list of Calculators that is configured for asynchronous calculations requests (added through px.addCalculatorToDistrList function).

px.queueSize()

Returns the number of requests that are queued for asynchronous calculation.

px.ready(id)

Checks if the asynchronous calculation with the provide **ID** is finished. Returns **true** if it is finished, otherwise **false**.

px.transform(input, xslFile, resourceConfigName, resourceConfigVersion, resourceConfigDate, parameters, priority)

Performs a transformation on the provided **input** XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided **xslFile** name and the provided parameters in **parameters**. To be able to know what resource configuration to use, a selection must be made with the name, version and date parameters. The **priority** parameter is optional and is an integer that can have the following values: (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

px.transformCalculateAndTransform(input, xslFile1, xslFile2, resourceConfigName, resourceConfigVersion, resourceConfigDate, parameters1, parameters2, priority)

Performs a transformation on the provided input XML and performs an XSLT transformation on this XML data using the style-sheet with the provided xslFile1 name and the provided parameters in parameters1. The result of this first transformation should be a calculation request. The calculation request is executed, and on the result an XSLT transformation is performed using the style-sheet with the provided xslFile2 name and the provided parameters in parameters2. To be able to know what resource configuration to use, a selection must be made with the name, version and date parameters. The priority parameter is optional and is an integer that can have the following values: (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

px.wait(id)

Waits until the asynchronous calculation request with the provided **ID** is finished. Returns the result of the calculation (calculator Output XML).

px.waitForAny()

Waits until the result of an asynchronous calculation request is available.

The PxTest.exe Testing Tool

It can be invaluable to test the newly created financial products as soon as possible from the functional or calculations perspective, without waiting for the completion of the technical integration efforts to link the production system(s) with the Calculator.

Regardless, of the status of the actual Calculator integration with the production system, independent tests can be performed for a deployment package utilizing the PxTest tool that is delivered with Calculator.

PxTest is a command-line utility that can be used to send requests to the Calculator. It provides a simple interface to the most used functionalities of Calculator, and allows easy distribution of requests across multiple Calculators. It simulates the production system by making calls to one or more Calculators, which can reside both on the local machine and on a remote server.

❶ Another command line tool, CalcServerTest is also available that provides access to other functionality of Calculator.

PxTest Syntax

PxTest can be used with the following options:

```
PxTest [-config=<config-file>] <command> <options> <input-files>
```

The config-file describes the configuration that PxTest uses. This includes the calculators to use and the number of concurrent requests sent to each calculator. When no config-file is specified, the file PxTest.Settings file in the Calculator/Etc directory will be used.

```
PxTest help commands
```

PxTest recognizes different commands. One of the commands is the help command, which will give help for all available commands with the applicable options. The help command will give an overview of all commands.

PxTest Commands

The following commands are available in PxTest:

aggr:	perform an aggregation with multiple detail input files
calc:	send input to the Calculator(s) for one or more calculations
list:	list all configured calculators

Each command will be described in detail in the following sections.

PxTest list

The list command prints a list of all configured Calculators:

```
PxTest list [-status]
```

It lists the following information for each Calculator:

Id:	a unique name assigned to the Calculator
Hostname:	hostname where the Calculator is running

Port:	tcp/ip port number where the Calculator is listening
Threads:	number of concurrent requests to be sent
SSL:	whether secure sockets are used
Status:	the current state of the calculator (only with <code>–status</code>): <ul style="list-style-type: none"> ▶ Running: Calculator is operable ▶ StartingUp: Calculator is starting up, not ready for requests ▶ Suspended: Calculator is suspended ▶ Unreachable: Calculator is down or there is no network connection

PxTest calc

The calc command will send one or more calculate requests to the Calculator.

```
PxTest calc [<options>] <input-source> [ <input-source> ... ]
```

Each input-source can be one of the following:

- ▶ A *.clcin file containing the Calculator input xml
- ▶ The root of a directory tree containing *.clcin files
- ▶ A *.zip file containing *.clcin files

All found *.clcin files are sent concurrently to all configured calculators.

The resulting xml output (clcout) is written to the file system.

In case of a single input request, the output is written to a file with the same name as the input file, but with file-type *.clcout.

In case of multiple inputs, the results are written in a directory structure under ActualResults.

The location of the output-file(s) can be overridden with the `–output` and `–output-dir` options.

–output=<output-file>

This option specifies the name of the Calculator output xml file (clcout) in case of a single input.

–output-dir=<output-directory>

This option specifies the name of the directory where the Calculator output xml files (clcout) is stored in case of a directory or zip input-source, or multiple input-sources.

–verify

When the `–verify` option is specified the calculator output xml is compared with the expected results.

The expected results should be in a file with the same name as the input clcin but with extension *.clcexp in case of a single clcin.

In case of a directory or zip input-source the expected results should be in the ExpectedResults directory, and have the *.clcout extension.

The default location of the expected output(s) can be overridden with the `–expected` and `–expected-dir` options.

-expected=<expected-file>

This option specifies the name of the expected result file in case of a single input.

-expected-dir=<expected-directory>

This option specifies the name of the expected results directory in case of a directory or zip input-source, or multiple input sources.

-progress[=<interval>]

When this option is specified, PxTest prints a progress message every 'interval' seconds. If the interval is omitted, the default value of 10 is used.

-timeout=<value>

This option specifies the timeout value. When no Calculator returns a reply within this period, a 'no progress' message is printed.

-time[=<csv-file-name>]

Prints timing information for each request on console, or when 'csv-file-name' is specified, writes it to a file in csv format.

PxTest aggr

The aggr command sends an output aggregation to the configured Calculator(s). An output aggregation exists of one master-file and multiple detail-requests.

```
PxTest aggr [<options>] <master-clcin>
<link>=<detail-input>[,<detail-input> ...]
[<link>=<detail-input>[,<detail-input> ...] ...]
```

master-clcin is the Calculator input xml containing the master request.

link refers to the link-id of an aggregated link as specified in the master-clcin.

detail-input are input-sources containing clcins for the details connected to the link.

Each input-source can be one of the following:

- ▶ An *.clcin file containing the detail input xml
- ▶ The root of a directory tree containing detail *.clcin files
- ▶ A *.zip file containing detail *.clcin files

All found *.clcin files are sent concurrently to all configured Calculators with the AggregationMap function.

The resulting xml output (clcout) is written to the file with the same name as the master-input file but with extension *.clcout.

-output=<output-file>

This option specifies the name of the Calculator output xml file (clcout) of the aggregation.

-output-map=<output-directory>

When this option is specified, the results of all AggregationMap requests are written to files in the specified directory. This can be useful for troubleshooting (see the `-verify-map` (see "[-verify-map](#)" on page [37](#)) option).

-reduce=<reduce-barrier>

This option can be used to set the reduce-barrier. The reduce-barrier is the number of map-results that are collected before an AgregationReduce call to the Calculator is done.

-pre-xslt[=<xslt-file>]

When this option is specified, each detail-input is processed by the specified XSLT script before it is sent to a Calculator.

When no xslt-file is specified, the default file clcpre.xslt located in the Calculator/Etc directory is used.

This script will:

1. Set the request-id attribute of the clc:CalculationInput element to a value identifying the origin of the input (input-source + name/path within the input-source).
2. Substitute all <clc:Count/> elements with the sequence number assigned to the detail. This counter starts at one for each input-source.

-verify

When the –verify option is specified, the final Calculator output xml is compared with the expected results.

Note that due to the parallel nature of the algorithm used, variation in calculated values caused by rounding and truncation should be expected.

The expected results should be in a file with the same name as the maste-input clcin but with extension *.clcexp. This can be overridden by the –expected option.

-expected=<expected-file>

This option specifies the name of the expected result file.

-verify-map

When this option is specified, all map-results are compared to expected map-results. The expected map-results can be the map-results of a previous run gathered with the –output-map option.

The expected results should reside in the ExpectedResults directory, and have the *.clcmmap extension. This can be overridden by the –expected-dir option.

-expected-dir=<expected-directory>

This option specifies the name of the expected results directory used by the –verify-map option.

-progress[=<interval>]

When this option is specified, PxTest prints a progress message every 'interval' seconds. If the interval is omitted, the default value of 10 is used.

-timeout=<value>

This option specifies the timeout value. When no Calculator returns a reply within this period, a 'no progress' message is printed.

PxTest Configuration File

This section describes the format of the PxTest configuration file. The configuration file can be passed to PxTest with the –config option. When the –config option is omitted, the default PxTest.Settings file in the Calculator/Etc directory is used.

- The PxTest config file is an XML file.
- The root element should be PxTest.

Element	Sub-Element	Attribute	Meaning
Calculators			Defines the Calculators to be used.
	Calculator		Defines one Calculator.
		id	Unique identifier of the Calculator.
		host	The hostname of the computer where the Calculator is running.
		port	The Tcp/ip port number where the Calculator is listening.
		threads	The number of concurrent requests that are sent to the Calculator.
		ssl	Whether the Calculator is using secured sockets.
Aggregations			Contains options specific to Aggregations
	ReduceBarrier	value	The reduce-barrier is the number of map-results that are collected before an AggregationReduce call to the Calculator is done. The default value is 100.
	Preprocessor	xslt	Pre-process all detail-inputs with the specified xslt script.
Progress			Contains options related to progress reporting.
	Report	Interval	The interval in seconds that a progress message is printed.
		Timeout	When no Calculator returns a reply within the timeout period, a 'no progress' message is printed.
OutputDir			Sets the default value for the <code>-output-dir</code> option.
ExpectedDir			Sets the default value for the <code>-expected-dir</code> option.

SNMP Access to the Calculator

The ProductXpress Calculator Server has a built in SNMP Server. This server can be enabled by adding the following line to the *Calculator.Settings* file in the Calculator Etc directory.

```
<EnableSNMP value="true"/>
```

To prevent interference with other SNMP servers on the same machine, Calculator SNMP server is not listening at the normal SNMP port (161) but can be reached via port 33161. This port number can be changed by editing the *CalculatorSNMP.conf* file in the Calculator SNMP directory. To change the listen port, change the line:

```
agentaddress localhost:33161
```

i The Calculator has to be restarted before any changes take effect.

Configure your SNMP client to connect to the assigned port using either SNMP v1 or v2c and community name *public*.

Defined MIBs

All defined MIB IDs are under *iso.org.dod.internet.private.enterprises.hp.hpAppMgt.hpProductXpress.pxCalculator.pxcServer* (1.3.6.1.4.1.11.14.8.1.1) and defined in the files HP-MIB.txt and PX-MIB.txt, which can be found in the Calculator SNMP\share\mibs directory.

The following information can be obtained from the Calculator via the SNMP interface:

MIB ID	Name	Type	Description
1.3.6.1.4.1.11.14.8.1.1.1	pxcMaxClients	Integer	The maximum number of clients that can have a connection with the ProductXpress Calculator at the same time. If more clients are connecting, the oldest connection(s) will be closed.
1.3.6.1.4.1.11.14.8.1.1.2	pxcCurrClients	Integer	The actual number of clients that currently have a connection with the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.3	pxcServerUpTime	Time Ticks	The time (in hundredths of a second) since the ProductXpress Calculator was (re)started.
1.3.6.1.4.1.11.14.8.1.1.4	pxcListenPort	Integer	The TCP/IP port number where the ProductXpress Calculator waits for new clients to connect.
1.3.6.1.4.1.11.14.8.1.1.5	pxcHostname	Display String	Hostname of the machine where the ProductXpress Calculator is running.
1.3.6.1.4.1.11.14.8.1.1.6	pxcMaxWorkerThreads	Integer	The maximum number of worker threads used by the ProductXpress Calculator server.
1.3.6.1.4.1.11.14.8.1.1.7	pxcMaxIdleWorkers	Integer	The maximum number of idle worker threads. If a worker thread is ready, and

MIB ID	Name	Type	Description
			the work queue is empty, then the worker thread will be terminated.
1.3.6.1.4.1.11.14.8.1.1.8	pxcActWorkerThreads	Integer	The actual number of worker threads in the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.9	pxcIdleWorkers	Integer	The actual number of idle worker threads in the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.10	pxcListenAddress	Display String	The ip-addresses where the ProductXpress Calculator is listening for new connections.
1.3.6.1.4.1.11.14.8.1.1.11	pxcServerName	Display String	The name of ProductXpress Calculator Server.
1.3.6.1.4.1.11.14.8.1.1.12	pxcUsingSSL	Integer	True if the ProductXpress Calculator Server is using Secured Sockets.
1.3.6.1.4.1.11.14.8.1.1.13	pxcHomeDir	Display String	The installation directory of the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.14	pxcMemUsage	Integer (kBytes)	The amount of physical memory in use by the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.15	pxcCPUUsage	Integer	The percentage of CPU usage by the ProductXpress Calculator.
1.3.6.1.4.1.11.14.8.1.1.16	pxcQueueSize	Integer	The size of the internal work queue of the ProductXpress Calculator.

Calculator Web Portal

The Calculator Web Portal offers the ability to connect to a Calculator and run requests and scripts through a Web browser. It also includes links to all current Calculator documentation in HTML format and download possibilities of the Calculator Examples (including integration).

Connect to a Calculator

In order to connect to a Calculator through a Web browser, you must do the following:

- ▶ Open up a Web browser such as Internet Explorer.
- ▶ Type in the url `http://CalculatorServer:Portnumber`. On a local server, the default installation is:

`http://localhost:31000/`

Perform a Calculation

In order to perform a calculation, navigate to the menu item CALCULATOR CONSOLE, TOOLS, CALCULATOR TESTER.

Within this interface, it is possible to select either the request file or script, set the calculator options, specify expected results if required, and view the output and results. For a detailed usage description, please see the online help within the Calculator (Web) Console.

View the Documentation

In order to view the Calculator documentation from the Web portal, select the menu item CALCULATOR GUIDES. Click on a link to view the chosen document.

Download Calculator Examples

In order to download Calculator Example Code, select CALCULATOR EXAMPLES from the menu, then select one of the following:

- ▶ External functions – This page has a link to the External function Example.
- ▶ Embedded Push – This page contains links to the C++ and .Net examples for the Embedded Push Calculator.
- ▶ Service Calculator – This page contains links to all the examples for the Service Calculator.(C++, COBOL, Java, .Net, iSeries, Web Service, etc.).
- ▶ Production Publishing – This page contains a link to the Production Publishing Example.

Interfacing the Service Calculator

This section describes how to interface with the Service Calculator from a Client Application. There are several ways to interface with the Service Calculator from a client application:

- ▶ Using the raw TCP/IP calling protocol
This requires raw socket programming in the client application.
- ▶ Using the SOAP calling protocol
This requires the client application to send HTTP POST requests containing SOAP encoded calculation requests.
- ▶ Using the CalcClient library
This library provides a C interface to the client application. The advantages of using this library instead of TCP/IP and SOAP protocols are:
 - ◆ No low level socket programming required
 - ◆ Has a string based interface
 - ◆ Supports asynchronous calculate calls
 - Multiple requests at once
 - Load balancing between configured Calculators
- ▶ Using the COM interface
Only available on Windows platform.
- ▶ Using the Perl interface
Only XSLT transformation functions are supported.

The Raw TCP/IP Calling Protocol

A TCP/IP Request sent to the Calculator consists of a header containing the character sequence "ConnV3.0\n" (without the double quotes), followed by a CalculationInput.

The result consists of a CalculationOutput.

In the TCP/IP interface, a socket is used to make a connection to the Calculator.

Socket

Any ANSI-compliant socket implementation can be used. On the Windows platform for instance the Winsock2 library, on the UNIX platform the BSDSock library.

Open

Connects to the Calculator socket by opening a connection using the right host name and port number.

Host

The host name identifies the computer where the Calculator is running, either by IP number or by the DNS name it has in the network.

Port number

The port number is the port number where the Calculator socket is listening for clients. It is default 30001 and can be set by the 'ListenPort' variable in Calculator.Settings.

Example:

```
<Calculator>
  <ListenPort value="32670" />
</Calculator>
```

The port number is logged when the Calculator starts.

Example:

```
Server Calculator: listening on port 32670 for requests. (logged in
Server::listen(), module FiaSrv)
```

Sending Requests

A request is sent by writing a header followed by the XML request to the socket. See the request section for further information.

Header

The header for a Calculator 3.5.x TCP/IP request consists of :

```
ConnV3.0\n
```

Where \n stands for the newline character.

Receiving Results

After sending the request, read from the socket until the complete result is received. This can be achieved by parsing the XML as a stream from the socket, or by checking the stream for the end XML tag.

Authentication

When the Calculator is configured such that authentication is required, a client must logon first before sending a calculation request to the Calculator.

A logon request looks like:

```
SecurityV1.0\n
<?xml version="1.0" encoding="UTF-8"?>
<SecurityRequest>
  <Logon password="userPassword" username="user"/>
</SecurityRequest>
```

Where \n stands for the newline character. The password XML attribute contains the password of the user and the username XML attribute contains the name of the user. The result of a logon request looks like :

```
<?xml version='1.0'?>
<SecurityResult username='user' loggedOn='true'/>
```

Where the username XML attribute indicates the user for which this is the logon result and the loggedOn XML attribute has value 'true' if the logon is succeeded or value 'false' if the logon failed .

❶ The logon is only valid for the socket connection on which the logon is performed. When using multiple socket connections, a logon must be performed for each socket connection .

Considerations

The TCP/IP interface has less overhead than the (D)COM and SOAP interfaces.

The SOAP Calling Protocol

This section describes aspects of the SOAP calling protocol.

SOAP Request

A soap request consists of the following header:

```
POST Some-URI/optional priority (0=default, 1=low, 2=normal, 3=high)
HTTP/1.1
Host: server address
Content-Type: text/xml; charset= "utf-8"
Content-Length: length of message starting directly after this line
```

```
SOAPAction: "/FiaNetV1.0/RMI"
```

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <SOAP-ENV:Header>
    <a:authentication
xmlns:a='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
      <name>UserName</name>
      <passwd></passwd>
    </a:authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

```
<!--The body of the SOAP request, which is specific for the kind of
SOAP request that has to been sent (See specific SOAP requests for
details).-->
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- ❶ The SOAP header contains either an 'authentication' part or a 'sessionInfo' part. In the first SOAP request, after establishing the connection with the Calculator, the 'authentication' part must contain user/client authentication information and no sessionInfo part should be present.
 - ❶ For following SOAP requests using the same connection and session, the information from the 'sessionInfo' part of the result of the first request must be contained in the 'sessionInfo' part of the new/next SOAP request. This means that such a request contains only a 'sessionInfo' part and no 'authentication' part.
 - ❶ For following SOAP requests using the same connection that contains an 'authentication' part will be treated as starting a new session. The corresponding SOAP response contains the 'sessionInfo' of this new session. Note that using 'authentication' part in each request using the same connection, results in more overhead in handling the processing of the request.
-

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset= "utf-8"
Content-Length: length of message starting directly after this line

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <SOAP-ENV:Header>
    <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
      <id>11111111-2222-3333-4444-555555555555</id>
      <server-host>hostname</server-host>
      <server-port>30001</server-port>
    </s:sessionInfo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <!--The body of the SOAP response, which is specific for the kind
of SOAP request that has been done (See specific SOAP responses for
details).-->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- ❶ The values in the 'sessionInfo' part of the SOAP header are example values.
-

SOAP Fault

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: length of message starting directly after this line

<SOAP-ENV:Envelope
```

```

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
<SOAP-ENV:Header>
  <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet' >
    <id>11111111-2222-3333-4444-555555555555</id>
    <server-host>hostname</server-host>
    <server-port>30001</server-port>
  </s:sessionInfo>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Client</faultcode>Footnotel
    <faultstring>.....</faultstring>
    <detail>
      Detailed description of the error
    </detail>
  </SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

i The values in the 'sessionInfo' part of the SOAP header are example values.

SOAP calculate request body

The body of the SOAP calculation request looks like:

```

<m:calculate
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <request>
    CalculationInput XML formatted without XML header: <?xml
    version='1.0'?>
  </request>
</m:calculate>

```

The following is an XML example of a complete SOAP calculation request:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <SOAP-ENV:Header>
    <a:authentication
xmlns:a='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
      <name>UserName</name>
      <passwd></passwd>
      <processId>1</processId>
    </a:authentication>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:calculate
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
      <request>

```

```

        <clc:CalculationInput
xmlns:clc="http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement">
        <clc:DeplR dep-name="Simple Product Deployment in Simple
Administration"
            ver-sel="last"/>
        <clc:CalculationData>
            <simplePolicy id="POL1" ext-id="1">
                <Features>
                    <PolicyFieldA>10</PolicyFieldA>
                    <PolicyFieldB>20</PolicyFieldB>
                </Features>
            </simplePolicy>
        </clc:CalculationData>
        <clc:Calculation>
            <simplePolicy ref="POL1">
                <Features>
                    <PolicySimpleCalculation/>
                </Features>
            </simplePolicy>
            <clc:CalculationDates>
                <clc:At>2000-01-01</clc:At>
            </clc:CalculationDates>
        </clc:Calculation>
    </clc:CalculationInput>
</request>
</m:calculate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP calculate response body

The body of the SOAP calculation response looks like:

```

<m:calculateResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
    <return type='xml'>
        CalculationOutput XML formatted without XML header: <?xml
version='1.0'?>
    </return>
</m:calculateResponse>

```

The following is an XML example of a complete SOAP calculation response:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
    <SOAP-ENV:Header>
        <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
            <id>b244b2ec-672d-417a-b431-48d796339ae6</id>
            <server-host>hostname</server-host>
            <server-port>30001</server-port>
        </s:sessionInfo>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:calculateResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/FiaNet">

```

```

    <return type='xml'>
      <clc:CalculationOutput
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns='Deployment:Simple%20Product%20Deployment%20in%20Simple%20Administration'>
        <clc:RuntimeConfiguration name='Simple Product Deployment
in Simple Administration' version='0.1'
id='ID:7f1f8a97-a418-4ef0-9403-7216b9d9748d' />
        <clc:Calculation>
          <simplePolicy ref='POL1' ext-id='1'>
            <Features>

<PolicySimpleCalculation>30</PolicySimpleCalculation>
          </Features>
        </simplePolicy>
      </clc:Calculation>
    </clc:CalculationOutput>
  </return>
</m:calculateResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP transform request body

The body of the SOAP transform request looks like:

```

<transform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
  <sourceXml>
    The source XML for the transformation (for example calculation
output)
  </sourceXml>
  <xslFileName>The name of the XSLT template file</xslFileName>
  <parameters>
    The parameters for the transformation:
    <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
    <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
  </parameters>
  <resourceConfigName>
    The name of the resource Configuration to use for the
transformation.
  </resourceConfigName>
  <resourceConfigVersion>
    The version of the resource Configuration to use for the
transformation.
    (Not to use in combination with date)
  </resourceConfigVersion>
  <resourceConfigDate>
    The date of the resource Configuration to use for the
transformation.
    (Not to use in combination with version)
  </resourceConfigDate>
</transform>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

The following is an XML example of a transform SOAP request:

```

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <a:authentication
xmlns="http://www.solcorp.com/ns/ProductXpress/FiaNet">
      <name xmlns="" />
      <passwd xmlns="" />
    </a:authentication>
  </soap:Header>
  <soap:Body>
    <transform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
      <sourceXml>
        <clc:CalculationOutput
xmlns:clc="http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="Deployment:Simple%20Product%20Deployment%20in%20Simple%20Administration"><clc:RuntimeConfiguration name="Simple Product
Deployment in Simple Administration" version="0.1"
id="ID:7f1f8a97-a418-4ef0-9403-7216b9d9748d"/>
          <clc:Calculation>
            <simplePolicy ref="POL1" ext-id="1">
              <Features>

<PolicySimpleCalculation>30</PolicySimpleCalculation>
                </Features>
              </simplePolicy>
            </clc:Calculation>
          </clc:CalculationOutput>
        </sourceXml>
        <xslFileName>Calculation results.xsl</xslFileName>
        <parameters>
          <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
          <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
        </parameters>
        <resourceConfigName>Simple Product Deployment in Simple
Administration</resourceConfigName>
        <resourceConfigVersion />
        <resourceConfigDate />
      </transform>
    </soap:Body>
  </soap:Envelope>

```

SOAP transform response body

The body of the SOAP transform response looks like:

```

<m:transformResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
  <return>
    Base64 encoded result of the transformation.
  </return>
</m:transformResponse>

```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

The following is an XML example of a transform SOAP response:

```
<SOAP-ENV:Envelope>
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <SOAP-ENV:Header>
    <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
      <id>e450a054-54fd-467b-b0d3-4f6c63c33dcc</id>
      <server-host>hostname</server-host>
      <server-port>30001</server-port>
    </s:sessionInfo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:transformResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
      <return>PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPGFydG
ljBjG
U+CiAgPGluZm8+CiAgICA8cHVIZGF0ZT4KPD9kYnRpbWVzdGFtcCAgZm9ybWFO
PSJZLW0tZCI/PgogICAgPC9wdWJkYXRlPgogIDwvaW5mbz4KICA8dG10bGU+VG
VzdGxhYiByZXBvcnQ8L3RpdGxlPgogIDxzZWNOaW9uPgogICAgPHRpdGxlPlN0
cnVjdHVyZS1lbGVtZW50OiBzaW1wbGVQb2xpY31bMV08L3RpdGxlPgogICAgPG
luZm9ybWFs dGFibGUGyYm9yZGVyPSIxIiB3aWR0aD0iMTAwJSI+CiAgICAgIDxj
b2xncm91cCB3aWR0aD0iMTAwJSI+CiAgICAgICAgPGNvbCB3aWR0aD0iMzAlIi
8+CiAgICAgICAgPGNvbCB3aWR0aD0iNzAlIi8+CiAgICAgIDwvY29sZ3JvdXA+
CiAgICAgIDx0aGVhZD4KICAgICAgICA8dHI+CiAgICAgICAgICA8dGg+RmVhdH
VyZTwvdGg+CiAgICAgICAgICA8dGg+VmFs dWU8L3RoPgogICAgICAgIDwvdHI+
CiAgICAgIDwvdGhlYWQ+CiAgICAgIDx0cj48dGg+UG9saWN5U21tcGxlQ2FsY3
VsYXRpb248L3RoPmludGVnZXI8dGQ+MzA8L3RkPjwvdHI+CiAgICA8L2luZm9y
bWFs dGFibGUG+CiAgPC9zZWNOaW9uPgo8L2FydGljBjBjGUG+Gg
      </return>
    </m:transformResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP calculate and transform request body

The body of the SOAP calculate and transform request looks like:

```
<calculateAndTransform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
  <calcInput>
    CalculationInput XML formatted without XML header: <?xml
version='1.0'?>
  </calcInput>
  <xslFileName>The name of the XSLT template file</xslFileName>
  <parameters>
    The parameters for the transformation:
    <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
    <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
  </parameters>
</calculateAndTransform>
```

See SOAP Request (on page 43) for the generic part of the SOAP request.

The following is an XML example of a calculate and transform SOAP request:

```

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <a:authentication
xmlns="http://www.solcorp.com/ns/ProductXpress/FiaNet">
      <name xmlns="" />
      <passwd xmlns="" />
    </a:authentication>
  </soap:Header>
  <soap:Body>
    <calculateAndTransform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
      <calcInput>
        <clc:CalculationInput
xmlns:clc="http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement">
          <clc:DeplR dep-name="Simple Product Deployment in Simple
Administration" ver-sel="last" />
          <clc:CalculationData>
            <simplePolicy id="POL1" ext-id="1" xmlns="">
              <Features>
                <PolicyFieldA>10</PolicyFieldA>
                <PolicyFieldB>20</PolicyFieldB>
              </Features>
            </simplePolicy>
          </clc:CalculationData>
          <clc:Calculation>
            <simplePolicy ref="POL1" xmlns="">
              <Features>
                <PolicySimpleCalculation />
              </Features>
            </simplePolicy>
            <clc:CalculationDates>
              <clc:At>2000-01-01</clc:At>
            </clc:CalculationDates>
          </clc:Calculation>
        </clc:CalculationInput>
      </calcInput>
      <xslFileName>Calculation results.xsl</xslFileName>
      <parameters>
        <KVPair><Key>key_0</Key><Value>value_0</Value></KVPair>
        <KVPair><Key>key_1</Key><Value>value_1</Value></KVPair>
      </parameters>
    </calculateAndTransform>
  </soap:Body>
</soap:Envelope>

```

SOAP calculate and transform response body

The body of the SOAP calculate and transform response looks like:

```

<m:calculateAndTransformResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
  <return>
    Base64 encoded result of the transformation.
  </return>

```


</m:calculateAndTransformResponse>

See SOAP Response (on page 44) for the generic part of the SOAP response.

The following is an XML example of a calculate and transform SOAP response:

```
<SOAP-ENV:Envelope>
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  >
    <SOAP-ENV:Header>
      <s:sessionInfo
        xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
        <id>f899e7a5-db41-4ccb-b1b1-940abf832a93</id>
        <server-host>hostname</server-host>
        <server-port>30001</server-port>
      </s:sessionInfo>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <m:calculateAndTransformResponse
        xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">

<return>PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPGFydG
ljBjG
U+CiAgPGluZm8+CiAgICA8cHViZGF0ZT4KPD9kYnRpbWVzdGFtcCAgZm9ybWF0
PSJZLW0tZCI/PgogICAgPC9wdWJkYXRlPgogIDwvaW5mbz4KICA8dG10bGU+VG
VzdGxhYiByZXBvcnQ8L3RpdGxlPgogIDxzZW50aW9uPgogICAgPHRpdGxlPlN0
cnVjdHVyZS1lbGVtZW50OiBzaW1wbGVQb2xpY31bMV08L3RpdGxlPgogICAgPG
luZm9ybWFs dGFibGUGyYm9yZGVyPSIxIiB3aWR0aD0iMTAwJSI+CiAgICAgIDxj
b2xncm91cCB3aWR0aD0iMTAwJSI+CiAgICAgICAgPGNvbCB3aWR0aD0iMzAlIi
8+CiAgICAgICAgPGNvbCB3aWR0aD0iNzAlIi8+CiAgICAgIDwvY29sZ3JvdXA+
CiAgICAgIDx0aGVhZD4KICA8dHI+CiAgICAgICAgICA8dGg+RmVhdH
VyZTwvdGg+CiAgICAgICAgICA8dGg+VmFs dWU8L3RoPgogICAgICAgIDwvdHI+
CiAgICAgIDwvdGhlYWQ+CiAgICAgIDx0cj48dGg+UG9saWN5U2ltcGxlQ2FsY3
VsYXRpb248L3RoPmludGVnZXI8dGQ+MzA8L3RkPjwvdHI+CiAgICA8L2luZm9y
bWFs dGFibGUG+CiAgPC9zZW50aW9uPgo8L2FydG1jBjBjGUG+Gg

      </return>
    </m:calculateAndTransformResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP transformCalculateAndTransform request body

The body of the SOAP transform request looks like:

```
<transformCalculateAndTransform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
  <sourceXml>
    The source XML for the transformation (for example calculation
output)
  </sourceXml>
  <xslFileName1>The name of the XSLT template file that transforms
the input XML into a calculation request</xslFileName1>
  <parameters1>
    The parameters for the first transformation:
    <KVPair><Key>key_0</Key><Value>value_0</Value></KVPair>
    <KVPair><Key>key_1</Key><Value>value_1</Value></KVPair>
  </parameters1>
</transformCalculateAndTransform>
```

```

    <xslFileName2>The name of the XSLT template file that should be
used to transform the result of the calculation</xslFileName2>
    <parameters2>
        The parameters for the second transformation:
        <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
        <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
    </parameters2>
    <resourceConfigName>
        The name of the resource Configuration to use for the
transformation.
    </resourceConfigName>
    <resourceConfigVersion>
        The version of the resource Configuration to use for the
transformation.
        (Not to use in combination with date)
    </resourceConfigVersion>
    <resourceConfigDate>
        The date of the resource Configuration to use for the
transformation.
        (Not to use in combination with version)
    </resourceConfigDate>
</transformCalculateAndTransform>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

The following is an XML example of a transform SOAP request:

```

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Header>
        <a:authentication
xmlns="http://www.solcorp.com/ns/ProductXpress/FiaNet">
            <name xmlns="" />
            <passwd xmlns="" />
        </a:authentication>
    </soap:Header>
    <soap:Body>
        <transformCalculateAndTransform
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
            <sourceXml>
                <clc:CalculationOutput
xmlns:clc="http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="Deployment:Simple%20Product%20Deployment%20in%20Simple%20Administration"><clc:RuntimeConfiguration name="Simple Product Deployment in Simple Administration" version="0.1" id="ID:7f1f8a97-a418-4ef0-9403-7216b9d9748d"/>
                    <clc:Calculation>
                        <simplePolicy ref="POL1" ext-id="1">
                            <Features>

<PolicySimpleCalculation>30</PolicySimpleCalculation>
                            </Features>
                        </simplePolicy>
                    </clc:Calculation>
                </sourceXml>
            </transformCalculateAndTransform>
        </soap:Body>
    </soap:Envelope>

```

```

        </clc:CalculationOutput>
    </sourceXml>
    <xslFileName1>CreateRequest.xsl</xslFileName1>
    <parameters1>
        <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
        <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
    </parameters1>
    <xslFileName2>TransformCalcResult.xsl</xslFileName2>
    <parameters2>
        <KeyValuePair><Key>key_0</Key><Value>value_0</Value></KeyValuePair>
        <KeyValuePair><Key>key_1</Key><Value>value_1</Value></KeyValuePair>
    </parameters2>
    <resourceConfigName>Simple Product Deployment in Simple
Administration</resourceConfigName>
    <resourceConfigVersion />
    <resourceConfigDate />
</transformCalculateAndTransform>
</soap:Body>
</soap:Envelope>

```

SOAP transformCalculateAndTransform response body

The body of the SOAP transform response looks like:

```

<m:transformCalculateAndTransformResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
    <return>
        Base64 encoded result of the transformation.
    </return>
</m:transformCalculateAndTransformResponse>

```

See SOAP Response (on page 44) for the generic part of the SOAP response.

The following is an XML example of a transform SOAP response:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
    <SOAP-ENV:Header>
        <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
            <id>e450a054-54fd-467b-b0d3-4f6c63c33dcc</id>
            <server-host>hostname</server-host>
            <server-port>30001</server-port>
        </s:sessionInfo>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:transformCalculateAndTransformResponse
xmlns:m="http://www.solcorp.com/ns/ProductXpress/theCalcServer">
            <return>PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPGFydG
ljbG
U+CiAgPGluZm8+CiAgICA8cHViZGF0ZT4KPD9kYnRpbWVzdGFtcCAgZm9ybWFO
PSJZLW0tZCI/PgogICA8PC9wdWJkYXRlPgogIDwvaW5mbz4KICA8dG10bGU+VG
VzdGxhYiByZXBvcnQ8L3RpdGxlPgogIDxzZW50aW9uPgogICA8PHRpdGxlPlN0
cnVjdHVsZS1lbGVtZW50OiBzaW1wbGVyY2xpY3lzMV08L3RpdGxlPgogICA8PG
luZm9ybWVsdGFibGUgYm9yZGVyPSIxIiB3aWR0aD0iMTAwJSI+CiAgICA8IDxj
b2xncm91cCB3aWR0aD0iMTAwJSI+CiAgICA8ICAgPGNvbCB3aWR0aD0iMzAlIi

```

```

8+CiAgICAgICAgPGNvbCB3aWR0aD0iNzAlIi8+CiAgICAgIDwvY29sZ3JvdXA+
CiAgICAgIDx0aGVhZD4KICAgICAgICA8dHI+CiAgICAgICAgICA8dGg+RmVhdH
VyZTWvdGg+CiAgICAgICAgICA8dGg+VmFsdWU8L3RoPgogICAgICAgIDwvdHI+
CiAgICAgIDwvdGhlYWQ+CiAgICAgIDx0cj48dGg+UG9saWN5U2ltcGxlQ2FsY3
VsYXRpb248L3RoPmludGVnZXI8dGQ+MzA8L3RkPjwvdHI+CiAgICA8L2luZm9y
bWFSdGFibGU+CiAgPC9zZWN0aW9uPgo8L2FydG1jbGU+CG
    </return>
  </m:transformCalculateAndTransformResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP runtime configuration request body

The body of the SOAP runtime configuration request looks like:

```

<configurationAsXml
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalculatorRuntim
eConfig">
  </configurationAsXml>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

The following is an XML example of a runtime configuration SOAP request:

```

<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <a:authentication
xmlns:a="http://www.solcorp.com/ns/ProductXpress/FiaNet">
      <name xmlns="" />
      <passwd xmlns="" />
    </a:authentication>
  </soap:Header>
  <soap:Body>
    <configurationAsXml
xmlns="http://www.solcorp.com/ns/ProductXpress/theCalculatorRuntim
eConfig">
      </configurationAsXml>
    </soap:Body>
  </soap:Envelope>

```

SOAP runtime configuration response body

The body of the SOAP runtime configuration response looks like:

```

<m:configurationAsXmlResponse
xmlns:m="CalcRuntimeConf:f6d906d6-940a-40e5-a807-bb1c2406fe13.Calc
ulatorRuntimeConf">
  <return>
    XML encoded runtime configuration
  </return>
</m:configurationAsXmlResponse>

```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

The following is an XML example of a runtime configuration SOAP response:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

```

```

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
<SOAP-ENV:Header>
<s:sessionInfo xmlns:s='FiaNet:SessionInfo'>
<id>7ffebe29-d734-4e6d-82fc-15746c3f6208</id>
<server-host>hostname</server-host>
<server-port>30001</server-port>
</s:sessionInfo>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:configurationAsXmlResponse
xmlns:m='CalcRuntimeConf:f6d906d6-940a-40e5-a807-bb1c2406fe13.Calc
ulatorRuntimeConf">
<return type='xml'>
  <RuntimeConfig>
    <TypeConfig
id='CalcRuntimeConf:94d7280e-2eb6-4235-a083-47ba5272ff8e.TypeConfi
g'><DeploymentPackageFile>ab4d8e3c-f242-4f24-98d9-89cd17ef14aa0.1.
pxdp</DeploymentPackageFile><Name>ProductA
deployment</Name><Version>0.1</Version><Documentation>User: john
Company: CompanyA
Date: 7-1-2008
19:09:28</Documentation><DeploymentId>FPM:05457c82-76db-403d-883d-
f2fd996697a5.TypeDeployment</DeploymentId></TypeConfig>
    <TypeConfig
id='CalcRuntimeConf:2cce7c7-8313-4987-a55d-d852ac952126.TypeConfi
g'><DeploymentPackageFile>0ed933a0-0ce7-4af3-9e7b-5912d6a77df80.1.
pxdp</DeploymentPackageFile><Name>TypeA
deployment</Name><Version>0.1</Version><Documentation>User: john
Company: CompanyA
Date: 7-1-2008
19:09:36</Documentation><DeploymentId>FPM:68285e34-7263-4010-9afc-
4072b78dcce9.TypeDeployment</DeploymentId></TypeConfig>
    <TypeConfig
id='CalcRuntimeConf:5febb72c-bcd1-4a91-9b7b-93467a2f5185.TypeConfi
g'><DeploymentPackageFile>0bcb0ea5-c8fc-4aa7-aac7-3d070d2385c01.2.
pxdp</DeploymentPackageFile><Name>TypeB
deployment</Name><Version>1.2</Version><Documentation>User: jane
Company: ACME
Date: 08-02-2008
11:47:19</Documentation><DeploymentId>FPM:bf20e8f1-4420-4496-9dfc-
48d2aadbe0c4.TypeDeployment</DeploymentId></TypeConfig>
    <TypeConfig
id='CalcRuntimeConf:e7a66858-0995-437c-91e5-697cd4512562.TypeConfi
g'><DeploymentPackageFile>0bcb0ea5-c8fc-4aa7-aac7-3d070d2385c01.1.
pxdp</DeploymentPackageFile><Name>TypeB
deployment</Name><Version>1.1</Version><Documentation>User: jane
Company: ACME
Date: 08-02-2008
11:47:32</Documentation><DeploymentId>FPM:47a2dcae-ed08-4a25-a86e-
76f55ede1dab.TypeDeployment</DeploymentId></TypeConfig>
    <TypeConfig
id='CalcRuntimeConf:44653f97-2979-4db9-b51f-5f2b223a491f.TypeConfi
g'><DeploymentPackageFile>fb57dd53-f4f6-4f96-97bc-b55ec2691ad81.0.
pxdp</DeploymentPackageFile><Name>TypeC
deployment</Name><Version>1.0</Version><Documentation>User: jane
Company: ACME

```

```

Date: 08-02-2008
11:31:24</Documentation><DeploymentId>FPM:c8a79563-3ac8-4a84-8f37-
f3908bec643b.TypeDeployment</DeploymentId></TypeConfig>
  <ResourceConfig
id='CalcRuntimeConf:a42dbaca-0dd8-4449-8b33-ad1cf6ce88f4.ResourceC
onfig'><DeploymentPackageFile>c771a99c-aeab-464c-9475-46deae7da43d
1.2.pxdp</DeploymentPackageFile><Name>config1</Name><ActiveDate>20
08-01-31</ActiveDate><Version>1.2</Version><ExpiryDate/><Documenta
tion>User: jane
Company: ACME
Date: 08-02-2008
11:33:20</Documentation><URI>Deployment:config1</URI><DeploymentId
>FPM:8cc4352d-8167-4359-8a26-f6967d1fe739.AutoDeployment</Deployme
ntId>
  <Status>Valid</Status><PreOptimize>>false</PreOptimize>
  <TypeConfig
id='CalcRuntimeConf:e7a66858-0995-437c-91e5-697cd4512562.TypeConfi
g'>/>
  <TypeConfig
id='CalcRuntimeConf:44653f97-2979-4db9-b51f-5f2b223a491f.TypeConfi
g'>/>
  </ResourceConfig>
  <ResourceConfig
id='CalcRuntimeConf:1113ccf2-7e87-4391-8f32-177976098252.ResourceC
onfig'><DeploymentPackageFile>c771a99c-aeab-464c-9475-46deae7da43d
1.2.pxdp</DeploymentPackageFile><Name>config2</Name><ActiveDate>20
08-01-31</ActiveDate><Version>1.2</Version><ExpiryDate/><Documenta
tion>User: jane
Company: ACME
Date: 08-02-2008
11:33:20</Documentation><URI>Deployment:config2</URI><DeploymentId
>FPM:8cc4352d-8167-4359-8a26-f6967d1fe739.AutoDeployment</Deployme
ntId>
  <Status>Valid</Status><PreOptimize>>false</PreOptimize>
  <TypeConfig
id='CalcRuntimeConf:5febb72c-bcd1-4a91-9b7b-93467a2f5185.TypeConfi
g'>/>
  <TypeConfig
id='CalcRuntimeConf:44653f97-2979-4db9-b51f-5f2b223a491f.TypeConfi
g'>/>
  </ResourceConfig>
  <ResourceConfig
id='CalcRuntimeConf:39207429-2490-4a29-842a-baae55b8a261.ResourceC
onfig'><DeploymentPackageFile>7f1f8a97-a418-4ef0-9403-7216b9d9748d
0.1.pxdp</DeploymentPackageFile><Name>Simple Product Deployment in
Simple
Administration</Name><ActiveDate>2003-04-25</ActiveDate><Version>0
.1</Version><ExpiryDate/><Documentation/><URI>Deployment:Simple%20
Product%20Deployment%20in%20Simple%20Administration</URI><Deployme
ntId>FPM:04b4911f-23da-4016-aald-4554cb77548e.StaticDeployment</De
ploymentId>

<Status>Valid</Status><PreOptimize>>false</PreOptimize></ResourceCo
nfig>
  <ResourceConfig
id='CalcRuntimeConf:36b1badf-7978-4a4a-85d3-47b4a4ef3a59.ResourceC
onfig'><DeploymentPackageFile>b15bb182-2f34-41ae-a8f1-b982fa560330

```

```

0.6.pxdp</DeploymentPackageFile><Name>regr. test dyn depl
b267</Name><ActiveDate>2006-09-22</ActiveDate><Version>0.6</Versio
n><ExpiryDate/><Documentation>User: james
Company: CompanyB
Date: 9/22/2006 11:21:40
AM</Documentation><URI>Deployment:regr.%20test%20dyn%20depl%20b267
</URI><DeploymentId>FPM:af5319f2-2a2f-4bb6-a675-65e23ca47a72.AutoD
eployment</DeploymentId>
    <Status>Invalid</Status><PreOptimize>true</PreOptimize>
    <ExternalResourceSelector
externalRoot='FPM:23013666-07c2-4773-ad86-0c9c70d3716c.StructureEl
ement'><DeploymentName>Simple Product Deployment in Simple
Administration</DeploymentName>
    <VersionNumber>0.1</VersionNumber>
    </ExternalResourceSelector>
  </ResourceConfig>
</RuntimeConfig>
</return>
</m:configurationAsXmlResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP aggregation start request body

The body of the SOAP aggregation start request looks like:

```

<aggregationStart
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <masterInput>
    the master clcin of the aggregation
  </masterInput>
  <_PxPrio>
    priority
  </_PxPrio>
</aggregationStart>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

The following is an XML example of a runtime configuration SOAP request:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
>
  <SOAP-ENV:Header>
    <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'
SOAP-ENV:mustUnderstand='1'>
      <id>20059631-0452-4413-8cbc-aa3ce9182d70</id>
      <server-host>hostname</server-host>
      <server-port>30001</server-port>
    </s:sessionInfo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:aggregationStart
xmlns:m='ClcSrv:ecdd302d-b372-4cbf-8125-2208860c83c6.CalcServer'>
      <masterInput>

```

```

        <clc:CalculationInput
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns='http://www.example.org/AggrTest1' detailed-errors='true'
request-id='1'>
        <clc:DeplR dep-name='AggrTest1' ver-sel='last'/>
        <clc:CalculationData>
            <Master id='ID0'>
                <Features>
                    <factor>0.3</factor>
                </Features>
                <Links>
                    <Detail aggregate='true' id='Aggr1'/>
                </Links>
            </Master>
        </clc:CalculationData>
        <clc:Calculation>
            <Master red='ID1'>
                <Features>
                    <aggr1/>
                </Features>
            </Master>
        </clc:Calculation>
    </clc:CalculationInput>
</masterInput>
    <_PxPrio>0</_PxPrio>
</m:aggregationStart>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP aggregation start response body

The body of the SOAP aggregation start response looks like:

```

    <aggregationStartResponse
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
    <return>
        the aggregation data XML structure
    </return>
</aggregationStartResponse>

```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

SOAP aggregation map request body

The body of the SOAP aggregation start request looks like:

```

    <aggregationStart
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
    <aggregationData>
The aggregation data XML structure returned by the      aggregation
start function.
    </aggregationData>
    <link>

```


The name of the aggregated link as indicated in the master clcin.

```

    </link>
    <detailInput>
      The clcin of the aggregation detail.
    </detailInput>
    <_PxPrio>
      priority
    </_PxPrio>
  </aggregationStart>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

The following is an XML example of a runtime configuration SOAP request:

```

</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <m:aggregationMap
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
    <aggregationData>
      . . .
    </aggregationData>
    <link>Aggr1</link>
    <detailInput>
      <clc:CalculationInput
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'

xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'

xmlns='http://www.example.org/AggrTest1'
request-id='detail-000010.clcin'>
      <clc:CalculationData>
        <Detail id='ID10'>
          <Features>
            <a>27</a>
            <b>80</b>
          </Features>
        </Detail>
      </clc:CalculationData>
    </clc:CalculationInput>
  </detailInput>
  <_PxPrio>0</_PxPrio>
</m:aggregationMap>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

SOAP aggregation map response body

The body of the SOAP aggregation map response looks like:

```

<aggregationMapResponse
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <return>
    the aggregation map data XML structure
  </return>

```

```
</aggregationMapResponse>
```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

SOAP aggregation reduce request body

The body of the SOAP aggregation reduce request looks like:

```
<aggregationReduce
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <aggregationData>
The aggregation data XML structure returned by the      aggregation
start function.
  </aggregationData>
  <mapResultSet>
    <Item>
      First aggregation map XML structure of mapResultSet
    </Item>
    . . .
    <Item>
      Last aggregation map XML structure of mapResultSet
    </Item>
  </mapResultSet>
  <_PxPrio>
    priority
  </_PxPrio>
</aggregationStart>
```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

SOAP aggregation reduce response body

The body of the SOAP aggregation reduce response looks like:

```
<aggregationReduceResponse
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <return>
    one aggregation map data XML structure
  </return>
</aggregationReduceResponse>
```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

SOAP aggregation start finish body

The body of the SOAP aggregation finish request looks like:

```
<aggregationFinish
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
  <aggregationData>
The aggregation data XML structure returned by the aggregation start
function.
  </aggregationData>
  <reduceValue>
The aggregation map data XML structure returned by the last aggregation
reduce call.
  </reduceValue>
  <masterInput>
```

```

        the master clcin of the aggregation
      </masterInput>
      <_PxPrio>
        priority
      </_PxPrio>
    </aggregationStart>

```

See SOAP Request (on page [43](#)) for the generic part of the SOAP request.

SOAP aggregation finish response body

The body of the SOAP aggregation finish response looks like:

```

    <aggregationFinishResponse
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
      <return>
        the aggregation result XML structure (clcout)
      </return>
    </aggregationFinishResponse>

```

See SOAP Response (on page [44](#)) for the generic part of the SOAP response.

The following is an XML example of a complete SOAP calculation response:

```

<SOAP-ENV:Envelope
xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
SOAP-ENV:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
>
  <SOAP-ENV:Header>
    <s:sessionInfo
xmlns:s='http://www.solcorp.com/ns/ProductXpress/FiaNet'>
      <id>ff6068ec-e38e-41dd-851e-57339985f57d</id>
      <server-host>hostname</server-host>
      <server-port>30001</server-port>
    </s:sessionInfo>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:aggregationFinishResponse
xmlns:m='http://www.solcorp.com/ns/ProductXpress/theCalcServer'>
      <return>
        <clc:CalculationOutput
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns='http://www.example.org/AggrTest1' request-id='1'>
          <clc:RuntimeConfiguration name='AggrTest1' version='0.1'
id='ID:f378c854-6036-4c35-aab7-680b837289fa' />
          <clc:Calculation>
            <Master ref='ID0'>
              <Features>
                <aggr1>28.2</aggr1>
              </Features>
            </Master>
          </clc:Calculation>
          <clc:AggregationDetails>
            <clc:LinkStatus link='Aggr1' included-instances='10000'
skipped-instances='0' />

```

```

        </clc:AggregationDetails>
    </clc:CalculationOutput>
</return>
</m:aggregationFinishResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Web Services Description Language (WSDL)

This section describes the Web Services Description Language (WSDL).

Calculate service operations

As the Service Calculator supports the SOAP protocol, all that needs to be provided with the Calculator is a WSDL document describing the 'calculate' service operation, in order for the Service Calculator to be Web Service enabled. This follows from the W3C definition of a Web Service:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Therefore several WSDL documents are provided with the Service Calculator in <Calculator Install Dir>/Etc/WSDL. They can be used to generate client proxy code for a specific Web Service. See Appendix A - Examples for the Service Calculator (on page [283](#)) for information on how to create a simple console application, written in C# or Java, that calls out to the Service Calculator acting as a Web Service. These examples are limited to the service operation named 'calculate', but client applications for other the service operations mentioned in the following paragraphs can be created in a similar way.

Calculation service operations

The WSDL document named 'CalcServer.wsdl' is provided with the Service Calculator in <Calculator Install Dir>/Etc/WSDL. This document describes the service operation 'calculate' and the aggregation related operations 'aggregationStart', 'aggregationMap', 'aggregationReduce' and 'aggregationFinish'. The WSDL document 'Calculator.wsdl' is provided for backward compatibility reasons and describes the service operation 'calculate'.

XSLT Transformation service operations

The WSDL document named 'CalcServer.wsdl' is provided with the Service Calculator in <Calculator Install Dir>/Etc/WSDL. This document describes the service operations 'transform', 'calculateAndTransform' and 'transformCalculateAndTransform'. The WSDL document named 'Transformer.wsdl' is provided for backward compatibility reasons and describes the service operations 'transform' and 'calculateAndTransform'.

Runtime configuration service operations

A WSDL document named 'CalculatorRuntimeConf.wsdl' is provided with the Service Calculator in <Calculator Install Dir>/Etc/WSDL. This document describes the service operation 'configurationAsXml'.

Administration service operations

A WSDL document named 'Administration.wsdl' is provided with the Service Calculator in <Calculator Install Dir>/Etc/WSDL. This document describes the service operation 'version'.

The CalcClient Library

To simplify the interface with the Service Calculator, a library is provided (Dynamic Link Library CalcClient.dll on Windows and Shared Library libCalcClient.so on AIX and Solaris platforms) which implements a C interface for communication with the Service Calculator. When using this library, there is no low level socket programming required as this is implemented in this library.

The library provides C functions for:

- ▶ performing calculations on the local Service Calculator
- ▶ performing XSLT transformations on calculation results through the local Service Calculator
- ▶ performing asynchronous calculations distributed over several Service Calculators (Load balancing)

❗ The C-interface uses UTF-8 encoded character strings at the interface.

Calculate functions

This section describes Calculate functions.

px_calculate

```
int px_calculate(const char* calcInput, char** calcOutput);
```

Performs a calculation with the provided Calculator Input XML and returns the result of the calculation (calculator Output XML). The settings files NamedObject.Settings and ServerURL.Settings are used to determine how to contact the Service Calculator.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
calcOutput	out	The result of the calculation. The returned character string must be freed with the px_freeCharBuffer when it is not needed anymore.

Returns 0 if successful.

px_calculateP

```
int px_calculateP(const char* calcInput, int priority, char** calcOutput);
```

Performs a calculation with the provided Calculator Input XML and priority and returns the result of the calculation (calculator Output XML). The settings files NamedObject.Settings and ServerURL.Settings are used to determine how to contact the Service Calculator.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Parameter	Input/Output	Description
priority	in	The priority with which this calculation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority)
calcOutput	out	The result of the calculation. The returned character string must be freed with the <code>px_freeCharBuffer</code> when it is not needed anymore.

Returns 0 if successful.

XSLT transformation functions

The XSLT transformation function allows you to perform XSLT transformations on the result of a calculation.

`px_calculateAndTransform`

```
int px_calculateAndTransform(const char* calcInput, const char*
xslFileName, const char** parameters, char** transformResult);
```

Performs a calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (calculator Output XML) using the style-sheet with the provided name.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
transformResult	out	The result of the transformation. The returned character string must be freed with the <code>px_freeCharBuffer</code> when it is not needed anymore.

Returns 0 if successful.

`px_calculateAndTransformP`

```
int px_calculateAndTransformP(const char* calcInput, const char*
xslFileName, const char** parameters, char** transformResult, int
priority);
```

Performs a calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (Calculator Output XML) using the style-sheet with the provided name.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
transformResult	out	The result of the transformation. The returned character string must be freed with the px_freeCharBuffer when it is not needed anymore.
priority	in	The priority with which this calculation and transformation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

Returns 0 if successful.

px_transform

```
int px_transform(const char* xmlInput, const char* xslFileName, const
char** parameters, const char* deploymentName, const char*
deploymentVersion, const char* deploymentDate, char**
transformResult);
```

Performs a transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.

Parameter	Input/Output	Description
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
transformResult	out	The result of the transformation. The returned character string must be freed with the <code>px_freeCharBuffer</code> when it is not needed anymore.

Returns 0 if successful.

px_transformP

```
int px_transformP(const char* xmlInput, const char* xslFileName,
const char** parameters, const char* deploymentName, const char*
deploymentVersion, int priority, const char* deploymentDate, char**
transformResult);
```

Performs a transformation, with a specified priority, on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
transformResult	out	The result of the transformation. The returned character string must be freed

Parameter	Input/Output	Description
		with the px_freeCharBuffer when it is not needed anymore.
priority	in	The priority with which this transformation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).

Returns 0 if successful.

px_transformCalculateAndTransform

```
int px_transformCalculateAndTransform(const char* xmlInput, const
char* xslFileName1, const char** parameters1, const char*
xslFileName2, const char** parameters2, const char* deploymentName,
const char* deploymentVersion, const char* deploymentDate, char**
transformResult, int* size);
```

Performs an XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName1	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters1	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
xslFileName2	in	This is the filename of the XSLT style-sheet to use for the second transformation, this is on the result of the calculation. This file must be located in the XSL directory of the Calculator installation.
parameters2	in	This is a list of parameter keys and values to be used for the second transformation. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.

Parameter	Input/Output	Description
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
transformResult	out	The result of the transformation. The returned character string must be freed with the <code>px_freeCharBuffer</code> when it is not needed anymore.
size	out	The size in bytes of the result.

Returns 0 if successful.

px_transformCalculateAndTransformP

```
int px_transformCalculateAndTransformP(const char* xmlInput, const
char* xslFileName1, const char** parameters1, const char*
xslFileName2, const char** parameters2, const char* deploymentName,
const char* deploymentVersion, const char* deploymentDate, int
priority, char** transformResult, int* size);
```

Performs an XSLT transformation on the provided input XML using the style-sheet with the provided name (`xslFileName1`) and using the provided parameters (`parameters1`) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (`xslFileName2`) and using the provided parameters (`parameters2`). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName1	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters1	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
xslFileName2	in	This is the filename of the XSLT style-sheet to use for the second transformation, this is on the result of the calculation. This file must be located in the XSL directory of the Calculator installation.
parameters2	in	This is a list of parameter keys and values to be used for the second transformation. The odd entries in the list are the keys and

Parameter	Input/Output	Description
		the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
priority	in	The priority with which this transformation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
transformResult	out	The result of the transformation. The returned character string must be freed with the px_freeCharBuffer when it is not needed anymore.
size	out	The size in bytes of the result.

Returns 0 if successful.

Aggregation functions

This section describes aggregation functions.

px_aggregationStart

```
int px_aggregationStart(const char* calcInput, int priority, char** aggregationDef);
```

Starts an aggregation.

Parameter	Input/Output	Description
calcInput	In	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	In	The priority with which this call must be handled. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
aggregationDef	Out	The data defining the aggregation.

Returns 0 if successful.

px_aggregationMap

```
int px_aggregationMap(const char* aggregationDef, const char* link, const char* calcInput, int priority, char** mapResult);
```

Adds to an aggregation.

Parameter	Input/Output	Description
aggregationDef	in	The aggregationDef as returned by the px_aggregationStart call.
link	In	The applicable aggregated link.
calcInput	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
Priority	In	The priority with which this call must be handled. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
mapResult	Out	The calculation output of the map function.

Returns 0 if successful.

px_aggregationReduce

```
int px_aggregationReduce(const char* aggregationDef, PX_MAPRESULTSET
mapResultSet, in priority, char** mapResult);
```

Finishes an aggregation.

Parameter	Input/Output	Description
aggregationDef	In	The aggregationDef as returned by the px_aggregationStart call.
mapResultSet	In	Collected mapResults. (see Aggregation Map Result Set functions)
Priority	In	The priority with which this call must be handled. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
mapResult	Out	The calculation output of the reduce function.

Returns 0 if successful.

px_aggregationFinish

```
int px_aggregationFinish(const char* aggregationDef, const char*
mapResult, const char* calcInput, int priority, char** calcResult);
```

Finishes an aggregation.

Parameter	Input/Output	Description
aggregationDef	In	The aggregationDef as returned by the px_aggregationStart call.

Parameter	Input/Output	Description
mapResult	In	The mapResult returned by the final px_aggregationReduce call.
calcInput	In	The original request.
Priority	In	The priority with which this finishAggregation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
calcResult	Out	The calculation output of the aggregation.

Returns 0 if successful.

Aggregation Map Result Set functions

The Map Result Set functions should be used to create a PX_MAPRESULTSET from multiple mapResults, as returned by the px_aggregationMap and px_aggregationReduce functions. The created mapResultSet should be passed to px_mapReduce. All PX_MAPRESULTSETs should be deleted with px_freeMapResultSet when they are no longer used.

px_aggregationMapResultSetCreate

```
PX_MAPRESULTSET px_aggregationMapResultSet();
```

This function creates an empty mapResultSet.

px_aggregationMapResultSetAdd

```
int px_aggregationMapResultSetAdd(PX_MAPRESULTSET mapResultSet, const char* mapResult);
```

This function adds one mapResult to a mapResultSet.

Parameter	Input/Output	Description
mapResultSet	In	The mapResultSet where the mapResult will be added.
mapResult	In	The mapResult to be added.

Returns 0 if successful.

px_aggregationMapResultSetJoin

```
int px_aggregationMapResultSetAdd(PX_MAPRESULTSET mapResultSet1, const PX_MAPRESULTSET mapResultSet2);
```

This function joins two mapResultSets, namely, mapResultSet2 will be added to mapResultSet1. mapResultSet2 will be unchanged.

Parameter	Input/Output	Description
mapResultSet1	In	The mapResultSet where the mapResult2 will be added.
mapResult2	In	The mapResult2 to be added.

Returns 0 if successful.

px_freeMapResultSet

```
void px_aggregationMapResultAdd(PX_MAPRESULTSET mapResultSet);
```

This function frees all resources allocated to the given mapResultSet.

Parameter	Input/Output	Description
mapResultSet	In	The mapResultSet to which the allocated resources can be freed.

Asynchronous calculation functions

Asynchronous calculations are calculations that are performed in background and distributed over the Calculators configured (In CalcClient.Settings or via px_addCalculatorToDistrList call) for asynchronous calculations.

If one of the configured Calculators is not reachable/off line, the requests are processed by the remaining Calculators. When this Calculator is available again, it will automatically be used again.

px_asyncCalculate

```
int px_asyncCalculate(const char* calcInput, int* id);
```

Starts an asynchronous calculation request for the provided Calculator Input XML and returns the ID of the calculation.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
id	out	The ID of the asynchronous calculation

Returns 0 if successful.

px_asyncCalculateP

```
int px_asyncCalculateP(const char* calcInput, int* id, int priority);
```

Starts an asynchronous calculation request for the provided Calculator Input XML and returns the ID of the calculation.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
id	out	The ID of the asynchronous calculation.
priority	in	The priority with which this calculation must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority)

Returns 0 if successful.

Asynchronous transform functions

This section describes asynchronous transform functions.

px_asyncCalculateAndTransform

```
int px_asyncCalculateAndTransform(const char* calcInput, const char*
xslFileName, const char** parameters, int* id);
```

Starts an asynchronous with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (calculator Output XML) using the style-sheet with the provided name.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
id	out	The ID of the asynchronous calculate and transform

Returns 0 if successful

px_asyncCalculateAndTransformP

```
int px_asyncCalculateAndTransformP(const char* calcInput, const
char* xslFileName, const char** parameters , int priority, int* id);
```

Starts an asynchronous with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (calculator Output XML) using the style-sheet with the provided name at the provided priority.

Parameter	Input/Output	Description
calcInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.

Parameter	Input/Output	Description
priority	in	The priority with which this <code>asyncCalculateAndTransform</code> must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority)
id	out	The ID of the asynchronous calculate and transform

Returns 0 if successful.

px_asyncTransform

```
int px_asyncTransform(const char* xmlInput, const char* xslFileName,
const char** parameters, const char* deploymentName, const char*
deploymentVersion, const char* deploymentDate, int* id);
```

Starts an asynchronous transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	In	The name of the configuration to use for the transformation.
deploymentVersion	In	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
id	out	The ID of the asynchronous transform

Returns 0 if successful.

px_asyncTransformP

```
int px_asyncTransformP(const char* xmlInput, const char* xslFileName,
const char** parameters, const char* deploymentName, const char*
deploymentVersion, const char* deploymentDate, int priority, int*
id);
```


Starts an asynchronous transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
priority	in	The priority with which this asyncTransform must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority)
id	out	The ID of the asynchronous transform

Returns 0 if successful.

px_asyncTransformCalculateAndTransform

```
int px_asyncTransformCalculateAndTransform(const char* xmlInput,
const char* xslFileName1, const char** parameters1, const char*
xslFileName2, const char** parameters2, const char* deploymentName,
const char* deploymentVersion, const char* deploymentDate, int* id);
```

Starts an asynchronous XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for

Parameter	Input/Output	Description
		the Push Calculator (on page 194)).
xslFileName1	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters1	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
xslFileName2	in	This is the filename of the XSLT style-sheet to use for the second transformation, this is on the result of the calculation. This file must be located in the XSL directory of the Calculator installation.
parameters2	in	This is a list of parameter keys and values to be used for the second transformation. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
id	out	The ID of the asynchronous transform

Returns 0 if successful.

px_asyncTransformCalculateAndTransformP

```
int px_asyncTransformCalculateAndTransformP(const char* xmlInput,
const char* xslFileName1, const char** parameters1, const char*
xslFileName2, const char** parameters2, const char* deploymentName,
const char* deploymentVersion, const char* deploymentDate, int
priority, int* id);
```

Starts an asynchronous XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
xmlInput	in	This is the input for the calculation in (See Calculator Input-Output Specification for

Parameter	Input/Output	Description
		the Push Calculator (on page 194)).
xslFileName1	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
parameters1	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
xslFileName2	in	This is the filename of the XSLT style-sheet to use for the second transformation, this is on the result of the calculation. This file must be located in the XSL directory of the Calculator installation.
parameters2	in	This is a list of parameter keys and values to be used for the second transformation. The odd entries in the list are the keys and the even entries in the list are the values.
deploymentName	in	The name of the configuration to use for the transformation.
deploymentVersion	in	The version of the configuration to use for the transformation (optional).
deploymentDate	in	The date of the configuration to use for the transformation (optional).
priority	in	The priority with which this asyncTransformCalculateAndTransform must be performed. (0=default priority, 1=low priority, 2=normal priority, 3=high priority)
id	out	The ID of the asynchronous transform

Returns 0 if successful.

Asynchronous aggregation functions

This section describes asynchronous aggregation functions.

px_asyncAggregationMap

```
int px_asyncAggregationMap(const char* aggregationDef, const char* link, const char* calcInput, int priority, int* id);
```

Adds to an aggregation.

Parameter	Input/Output	Description
aggregationDef	In	The aggregationDef as returned by the px_aggregationStart call.
link	In	The applicable aggregated link.
calcInput	In	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
Priority	In	The priority with which this call must be handled. (0=default priority, 1=low priority, 2=normal priority, 3=high priority).
id	Out	The ID of the asynchronous aggregation.

Returns 0 if successful.

Asynchronous result functions

This section describes asynchronous result functions.

px_ready

```
int px_ready(int id, int* result);
```

Checks if an asynchronous request is ready (finished).

Parameter	Input/Output	Description
id	in	The ID of the asynchronous result to be checked.
result	out	The result is !0 if the calculation is finished, otherwise result is 0.

Returns 0 if successful.

px_wait

```
int px_wait(int id, char** calcResult);
```

Waits until an asynchronous calculation request is finished and returns the result of the calculation (calculator Output XML).

Parameter	Input/Output	Description
id	in	The ID of the asynchronous calculation to wait for.
calcOutput	out	The result of the calculation or aggregation. The returned character string must be freed with the px_freeCharBuffer when it is not needed anymore.

Returns 0 if successful.

px_waitn

```
int px_waitn(int id, char** calcResult, int* resultSize);
```

Waits until an asynchronous calculation request is finished and returns the result of the calculation (calculator Output XML).

Parameter	Input/Output	Description
id	in	The ID of the asynchronous calculation to wait for.
calcResult	out	The result of the calculation, transformation or aggregation. The returned character string must be freed with the px_freeCharBuffer when it is not needed anymore.
resultSize	out	The size in bytes of the result.

Returns 0 if successful.

px_waitForAny

```
int px_waitForAny();
```

Waits until the result of an asynchronous calculation request is available.

Returns 0 if successful.

px_queueSize

```
int px_queueSize(int* size);
```

Returns the number of requests that are queued for asynchronous calculation.

Parameter	Input/Output	Description
size	out	The number of requests queued.

Returns 0 if successful.

px_clearCalculatorDistrList

```
int px_clearCalculatorDistrList();
```

Clears the list of Calculators that is configured for asynchronous calculation requests (Defined in CalcClient.Settings and/or added through px_addCalculatorToDistrList call). All asynchronous calculation requests in queue remain queued.

Returns 0 if successful.

px_addCalculatorToDistList

```
int px_addCalculatorToDistList(const char* name, const char* url, int nbConcurrentCalcs);
```

Adds a Calculator to the list of Calculators for executing asynchronous calculations requests.

Parameter	Input/Output	Description
name	in	The name of the calculator (must be unique within the configuration).

Parameter	Input/Output	Description
url	in	<p>The URL describes how the Calculator can be reached.</p> <p>Format:</p> <p><protocol>://<host>:<port>/ClcSrv</p> <p>Where :</p> <p><protocol> is the protocol to be used, can be fia (px specific protocol), sfia (px specific protocol over secured socket connections), http (uses SOAP formatted requests) or shhttp (uses SOAP formatted request over secured socket connections).</p> <p><host> is the host name or ip address of host on which the calculator is running.</p> <p><port> the port number used by the Calculator.</p>
nbConcurrentCalcs	in	The number of concurrent calculations for this calculator.

Returns 0 if successful.

CalcClient.Settings

The CalcClient.Settings file is an XML file with the following structure:

```
start = calcClient

calcClient = element CalcClient
{
    Calculator *
}

calculator = element Calculator
{
    attribute name { xsd:string },
    # The name of the calculator (unique)
    attribute url { xsd:string },
    # The url of the calculator
    # format: <protocol>://<host>:<port>/ClcSrv
    # see call px_addCalculatorToDirstList)
    attribute nbConcurrentCalcs { xsd:integer }
    # The number of concurrent calculations for this calculator
}
```

Example:

```
<?xml version="1.0"?>
<CalcClient>
  <Calculator name="MyCalc1"
    url="fia://localhost:30001/ClcSrv"
    nbConcurrentCalcs="3"/>
</CalcClient>
```

```
<Calculator name="MyCalc2"
    url=http://localhost:30002/ClcSrv
    nbConcurrentCalcs="2"/>
</CalcClient>
```

Miscellaneous functions

This section describes miscellaneous functions.

px_freeCharBuffer

```
void px_freeCharBuffer(char* buffer);
```

Frees an output character buffer.

Parameter	Input/Output	Description
buffer	In	Pointer to the character buffer to free.

px_getLastError

```
const char* px_getLastError();
```

Returns the last occurred error.

The Java Interface

The Java interface is located in the PxCalcClient.jar and PxCalcClient_jni.dll. It uses the CalcClient Library C-interface internally to communicate with the Calculator.

Class CalcClient

com.solcorp.productxpress.calcclient.CalcClient

instance

```
public enum Priority
{
    Default,
    Low,
    Medium,
    High,
}
```

instance

```
public static synchronized CalcClient instance()
```

Obtain an instance of the CalcClient class.

Returns the CalcClient object.

calculate

```
public native String calculate(String request) throws PxException
```

Perform a calculation.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for

Parameter	Description
	the Push Calculator (on page 194)).

Returns the result XML of the calculation.

```
public native String calculate(String request, Priority priority)
throws PxxException
```

Perform a calculation with a specific priority.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	The priority at which to execute this function.

Returns the result XML of the calculation.

asyncCalculate

```
public native int asyncCalculate(String request) throws PxxException
```

Perform asynchronous calculation.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Returns the ID of the async call.

```
public native int asyncCalculate(String request, Priority priority)
throws PxxException
```

Perform asynchronous calculation with specific priority.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	The priority at which to execute this function.

Returns the ID of the async call.

ready

```
public native boolean ready(int id) throws PxxException
```

Checks if an asynchronous request is ready (finished).

Parameter	Description
id	The ID of the asynchronous request to be checked.

Returns true if asynchronous request is ready.

wait

```
public native String wait(int id) throws PxException
```

Waits until a specific asynchronous request is ready (finished).

Parameter	Description
id	The ID of the asynchronous request to be checked.

Returns the result of the asynchronous request.

waitn

```
public native byte[] waitn(int id) throws PxException
```

Waits until a specific asynchronous request is ready (finished).

Parameter	Description
id	The ID of the asynchronous request to be checked.

Returns the result of the asynchronous request.

waitForAny

```
public native void waitForAny() throws PxException
```

Waits until an asynchronous request is ready (finished).

queueSize

```
public native int queueSize() throws PxException
```

Returns the number of asynchronous requests in queue.

clearCalculatorDistrList

```
public native void clearCalculatorDistrList() throws PxException
```

Clears the list of Calculators that is configured for asynchronous calculation requests (Defined in CalcClient.Settings and/or added through px_addCalculatorToDistrList call). All asynchronous calculation requests in queue remain queued.

addCalculatorToDistrList

```
public native void addCalculatorToDistrList(String name, String url,
int nbConcurrentCalcs) throws PxException
```

Waits until a specific asynchronous request is ready (finished).

Parameter	Description
name	The name of the calculator (must be unique)

Parameter	Description
	within the configuration).
url	<p>The URL describes how the Calculator can be reached.</p> <p>Format:</p> <p><protocol>://<host>:<port>/ClcSrv</p> <p>Where :</p> <p><protocol> is the protocol to be used, can be fia (px specific protocol), sfia (px specific protocol over secured socket connections), http (uses SOAP formatted requests) or shhttp (uses SOAP formatted request over secured socket connections).</p> <p><host> is the host name or ip address of host on which the calculator is running.</p> <p><port> the port number used by the Calculator.</p>
nbConcurrentCalcs	The number of concurrent calculations for this calculator.

calculateAndTransform

```
public native byte[] calculateAndTransform(String request, String
xslFileName, Map<String, String> parameters) throws PxxException
```

Performs a calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (Calculator Output XML) using the style-sheet with the provided name.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.

Returns the result of the transformation.

```
public native byte[] calculateAndTransform(String request, String
xslFileName, Map<String, String> parameters, Priority priority)
throws PxxException
```

Performs a calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (Calculator Output XML) using the style-sheet with the provided name.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
priority	The priority at which to execute this function.

Returns the result of the transformation.

transform

```
public native byte[] transform(String xmlInput, String xslFileName,
    Map<String, String> parameters, String deploymentName, String
    deploymentVersion, Date deploymentDate) throws PxException
```

Performs a transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).

Returns the result of the transformation.

```
public native byte[] transform(String xmlInput, String xslFileName,
    Map<String, String> parameters, String deploymentName, String
    deploymentVersion, Date deploymentDate, Priority priority) throws
    PxException
```

Performs a transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).
priority	The priority at which to execute this function.

Returns the result of the transformation.

transformCalculateAndTransform

```
public native byte[] transformCalculateAndTransform(String xmlInput,
String xslFileName1, Map<String, String> parameters1, String
xslFileName2, Map<String, String> parameters2, String
deploymentName, String deploymentVersion, Date deploymentDate)
throws PxException
```

Performs an XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName1	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters1	This is a map of parameter keys and values

Parameter	Description
	for the transformation.
xslFileName2	This is the filename of the XSLT style-sheet to use for the second transformation. This file must be located in the XSL directory of the Calculator installation.
parameters2	This is a map of parameter keys and values for the second transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).

Returns the result of the transformation.

```
public native byte[] transformCalculateAndTransform(String xmlInput,
String xslFileName1, Map<String, String> parameters1, String
xslFileName2, Map<String, String> parameters2, String
deploymentName, String deploymentVersion, Date deploymentDate,
Priority priority) throws PxException
```

Performs an XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName1	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters1	This is a map of parameter keys and values for the transformation.
xslFileName2	This is the filename of the XSLT style-sheet to use for the second transformation. This file must be located in the XSL directory of the Calculator installation.
parameters2	This is a map of parameter keys and values for the second transformation.

Parameter	Description
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).
priority	The priority at which to execute this function.

Returns the result of the transformation.

asyncCalculateAndTransform

```
public native int asyncCalculateAndTransform(String request, String
xslFileName, Map<String, String> parameters) throws PxException
```

Performs an asynchronous calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (Calculator Output XML) using the style-sheet with the provided name.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.

Returns the ID of the asynchronous request.

```
public native int asyncCalculateAndTransform(String request, String
xslFileName, Map<String, String> parameters, Priority priority)
throws PxException
```

Performs an asynchronous calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (Calculator Output XML) using the style-sheet with the provided name.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Parameter	Description
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
priority	The priority at which to execute this function.

Returns the id of the asynchronous request.

asyncTransform

```
public native int asyncTransform(String xmlInput, String xslFileName,
    Map<String, String> parameters, String deploymentName, String
    deploymentVersion, Date deploymentDate) throws PxException
```

Performs an asynchronous transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).

Returns the id of the asynchronous request.

```
public native int asyncTransform(String xmlInput, String xslFileName,
    Map<String, String> parameters, String deploymentName, String
    deploymentVersion, Date deploymentDate, Priority priority) throws
    PxException
```

Performs an asynchronous transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with

the provided name. To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters	This is a map of parameter keys and values for the transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).
priority	The priority at which to execute this function.

Returns the ID of the asynchronous request.

asyncTransformCalculateAndTransform

```
public native int asyncTransformCalculateAndTransform(String
xmlInput, String xslFileName1, Map<String, String> parameters1,
String xslFileName2, Map<String, String> parameters2, String
deploymentName, String deploymentVersion, Date deploymentDate)
throws PxException
```

Performs an asynchronous XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName1	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters1	This is a map of parameter keys and values for the transformation.

Parameter	Description
xslFileName2	This is the filename of the XSLT style-sheet to use for the second transformation. This file must be located in the XSL directory of the Calculator installation.
parameters2	This is a map of parameter keys and values for the second transformation.
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).

Returns the ID of the asynchronous request.

```
public native int asyncTransformCalculateAndTransform(String
xmlInput, String xslFileName1, Map<String, String> parameters1,
String xslFileName2, Map<String, String> parameters2, String
deploymentName, String deploymentVersion, Date deploymentDate,
Priority priority) throws PxException
```

Performs an asynchronous XSLT transformation on the provided input XML using the style-sheet with the provided name (xslFileName1) and using the provided parameters (parameters1) to create a calculate request. On the result of the calculation an XSLT transformation is performed XML using the style-sheet with the provided name (xslFileName2) and using the provided parameters (parameters2). To be able to know what resource configuration to use, a selection must be made with the name version and date parameters.

Parameter	Description
xmlInput	This is the input for transform.
xslFileName1	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
parameters1	This is a map of parameter keys and values for the transformation.
xslFileName2	This is the filename of the XSLT style-sheet to use for the second transformation. This file must be located in the XSL directory of the Calculator installation.
parameters2	This is a map of parameter keys and values for the second transformation.

Parameter	Description
deploymentName	The name of the configuration to use for the transformation.
deploymentVersion	The version of the configuration to use for the transformation (optional).
deploymentDate	The date of the configuration to use for the transformation (optional).
priority	The priority at which to execute this function.

Returns the ID of the asynchronous request.

aggregationStart

```
public native String aggregationStart(String request) throws
PxEception
```

Open an aggregation.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Returns the aggregationDef.

```
public native String aggregationStart(String request, Priority
priority) throws PxEception
```

Open an aggregation with specific priority.

Parameter	Description
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	The priority at which to execute this function.

Returns the aggregationDef.

aggregationMap

```
public native String aggregationMap(String aggregationDef, String
link, String request) throws PxEception
```

Add to an aggregation.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.

Parameter	Description
link	The link used for the aggregation
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Returns the aggregation mapResult.

```
public native String addToAggregation(String aggregation, String
link, String request, Priority priority) throws PxxException
```

Add to an aggregation with specific priority.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.
link	The link used for the aggregation
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	The priority at which to execute this function.

Returns the aggregation mapResult.

aggregationReduce

```
public native String aggregationReduce(String aggregationDef,
String[] mapResultSet) throws PxxException
```

Add to an aggregation.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.
mapResultSet	An array of mapResult as returned by the aggregationMap function.

Returns one aggregation mapResult.

```
public native String aggregationReduce(String aggregationDef,
String[] mapResultSet, Priority priority) throws PxxException
```

Add to an aggregation with specific priority.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.
mapResultSet	An array of mapResult as returned by the

Parameter	Description
	aggregationMap function.
priority	The priority at which to execute this function.

Returns one aggregation mapResult.

aggregationFinish

```
public native String aggregationFinish(String aggregationDef, String
mapResult, String request) throws PxException
```

Finish the aggregation and return the final result.

Parameter	Description
aggregationDef	The aggregation aggregationDef as returned by aggregationStart.
mapResult	The final mapResult returned by the last aggregationReduce function.
request	The original request, namely, the same request as passed to aggregationStart.

Returns the aggregation result.

```
public native String aggregationFinish(String aggregation, String
mapResult, String request, Priority priority) throws PxException
```

Finish an aggregation with specific priority and return the final result.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.
mapResult	The final mapResult returned by the last aggregationReduce function.
request	The original request, namely, the same request as passed to aggregationStart.
priority	The priority at which to execute this function.

Returns the aggregation result.

asyncAggregationMap

```
public native int asyncAggregationMap(String aggregationDef, String
link, String request) throws PxException
```

Add to an aggregation.

Parameter	Description
aggregationDef	The aggregationDef as returned by aggregationStart.
link	The link used for the aggregation
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

Returns the id of the asynchronous request.

```
public native int asyncAddToAggregation(String aggregation, String
link, String request, Priority priority) throws PxException
```

Add to an aggregation with specific priority.

Parameter	Description
aggregation	The aggregation
link	The link used for the aggregation
request	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
priority	The priority at which to execute this function.

Returns the ID of the asynchronous request.

The COM Interface

The Calculator provides a COM interface.

This interface is located in the Dynamic Link Library Conn.dll, which is part of the custom merge modules installation of the Calculator; for further details please see the *ProductXpress Calculator Installation Guide*.

This dll provides COM classes ObjectManager and Connector. From the ObjectManager class the Connector class can be obtained. The Connector class provides the interface method 'calculate', which uses the Calculator Input-Output xml.

The Conn.dll COM objects use the multi-threaded apartment model.

Appendix A - Examples (see "[Appendix A - Examples for the Service Calculator](#)" on page [283](#)) contains a complete example of how to connect to the COM interface.

The COM interface of Calculator has the following interface (Microsoft IDL notation):

```
interface IConnector;
interface IObjectManager;

typedef [helpstring("enumeration of Conn types")] enum ConnType
{
    Conn_Connector = 1
} ConnType;
```

```

[
    object,
    hidden,
    uuid(d92c6f2d-53de-43a5-945e-30075795ecb8),
    dual,
    helpstring("IHandle Interface"),
    pointer_default(unique)
]
interface IHandle : IDispatch
{
    [propget, helpstring("get type")] HRESULT type([out, retval]
ConnType *pVal);
    [propget, helpstring("get type in string form")] HRESULT
typeString([out, retval] BSTR *pVal);
    [propget, helpstring("get RefId in string form")] HRESULT
refId([out, retval] BSTR *pVal);
    HRESULT saveChanges();
};
[
    object,
    uuid(22e1feaa-4263-4e5b-aa68-1802679374ad),
    dual,
    helpstring("IConnector Interface"),
    pointer_default(unique)
]
interface IConnector : IHandle
{
    HRESULT calculate([in] BSTR request, [out, retval] BSTR *pVal);
};
[
    object,
    uuid(30d8402d-6b4e-43e6-a81a-890a8a87a8dc),
    dual,
    helpstring("IObjectManager Interface"),
    pointer_default(unique)
]
interface IObjectManager : IDispatch
{
    HRESULT login([in] BSTR username, [in] BSTR password);
    HRESULT getObject([in] BSTR refId, [out, retval] IHandle*
*pVal);
    HRESULT getNamedObject([in] BSTR name, [out, retval] IHandle*
*pVal);
    HRESULT refresh();
    HRESULT clearCache();
    HRESULT clearSession();
};

[
    uuid(f797e809-06c1-4f16-975e-8d3d9a8fffc),
    version(1.0),
    helpstring("Conn 1.0 Type Library")
]
library CONNLib
{

```

```

importlib("stdole32.tlb");
importlib("stdole2.tlb");

[
    uuid(10b330c8-6872-4076-81b9-8600b018fc0f),
    noncreatable,
    helpstring("Connector Class")
]
coclass Connector
{
    [default] interface IConnector;
};
[
    uuid(82953da7-4563-48e2-befa-a63c18e0dffc),
    helpstring("ObjectManager Class")
]
coclass ObjectManager
{
    [default] interface IObjectManager;
};
};

```

- ① Make sure to set the environment variable FIA_CALCULATOR_DIR to the installation directory of the Calculator, like: C:\Program Files\Hewlett-Packard\ProductXpress\Calculator. Setting of environment variables in Windows XP is done in the Control Panel -> Performance and Maintenance -> System -> Advanced tab.

Connection is the default priority.

Perl Interface

The Perl interface has only bindings for the XSLT transformation API functions.

It is located in the module "PxTransform.pm" from the location <Calculator Install Dir>\Perl\lib. Note that it uses dll's (Shared libraries for platforms other than Microsoft Windows) from the "<Calculator Install Dir>\Bin" directory. Also note that PxTransform.dll requires Perl version 5.8 and will not work with other versions of Perl.

XSLT Transformation Functions

This section describes XSLT transformation functions.

calculateAndTransform

```
$result = PxTransform::calculateAndTransform($inputString, $xslt, @params);
```

Performs a calculation with the provided Calculator Input XML and performs an XSLT transformation on the result of the calculation (calculator Output XML) using the style-sheet with the provided name.

Parameter	Input/Output	Description
inputString	in	This is the input for the calculation (See Calculator Input-Output Specification for the

Parameter	Input/Output	Description
		Push Calculator (on page 194)).
xslt	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the calculator installation.
params	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.

Returns the resulting text of the transformation.

transform

```
$result = PxTransform::transform($inputString, $xslt, @params,
$name, $version, $date);
```

Performs a transformation on the provided input XML (usually the result of a calculation) and performs an XSLT transformation on this XML data using the style-sheet with the provided name. To be able to know what runtime configuration to use, a selection must be made with the name version and date parameters.

Parameter	Input/Output	Description
inputString	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslt	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
params	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
name	in	The name of the configuration to use for the transformation.
version	in	The version of the configuration to use for the transformation (optional).
date	in	The date of the configuration to use for the transformation (optional).

Returns the resulting text of the transformation.

transformCalculateAndTransform

```
$result = PxTransform::transformCalculateAndTransform($inputString,
$xslt1, @params1, $xslt2, @params2, $name, $version, $date);
```


Performs an XSLT transformation on the provided XML input using the style-sheet with the provided name (xslt1) and the provided parameters (params1) to create a calculation request. After the calculation is performed, an XSLT transformation is performed on the result using the provided name (xslt2) with parameters provided in 'params2'. To be able to know what runtime configuration to use, a selection must be made with the name, version and date parameters.

Parameter	Input/Output	Description
inputString	in	This is the input for the calculation (See Calculator Input-Output Specification for the Push Calculator (on page 194)).
xslt1	in	This is the filename of the XSLT style-sheet to use for the transformation. This file must be located in the XSL directory of the Calculator installation.
params1	in	This is a list of parameter keys and values. The odd entries in the list are the keys and the even entries in the list are the values.
xslt2	in	This is the filename of the XSLT style-sheet to use for the second transformation to create the result. This file must be located in the XSL directory of the Calculator installation.
params2	in	This is a list of parameter keys and values to be used for the second transformation. The odd entries in the list are the keys and the even entries in the list are the values.
name	in	The name of the configuration to use for the transformation.
version	in	The version of the configuration to use for the transformation (optional).
date	in	The date of the configuration to use for the transformation (optional).

Returns the resulting text of the transformation.

Changing Client Priority

It is possible to change the priority of requests received on a specific socket connection. To change the priority for a specific socket connection, the client must send a message with the following format over that socket connection to the Calculator:

```
AdminV1.0
<?xml version="1.0"?>
<AdminRequest xmlns="http://www.solcorp.com/ns/Admin">
  <Priority>The new priority</Priority>
</AdminRequest>
```

In this message "The new priority" can have the values: "Low", "Normal" or "High". The default socket connection priority is "Normal".

The priority change is only effective for the duration of the socket connection. When a socket connection is closed and opened again, the priority for that socket will be "normal".

Runtime Considerations for the Service Calculator

The Service Calculator provides a number of runtime possibilities depending on the requirements for the phase of the project (testing or production) and the required architecture. The following runtime options are discussed in this section:

- ▶ Deployment Package Decryption
- ▶ Multithreading
- ▶ Multiple Instances

Deployment Package Decryption

ProductXpress Calculator can load and decrypt encrypted packages.

Prior to decrypting an encrypted deployment package, the key that will be used during the decryption process must be imported into Calculator. Keys used for decryption can also be removed, and the Calculator can provide a list of keys known to a Calculator instance.

Importing a certain key needs to be done once per key. Once a key is imported into the Calculator, it will decrypt any new package that is encrypted with it.

Package Decryption Syntax

To import a key use:

```
DepImp -keyimport=<key export file>
```

To remove a previously imported key use:

```
DepImp -keyremove=<key label>
```

To view the list of keys previously imported into the Calculator use:

```
DepImp -keylist
```

To change the label (name) of a previously imported key use:

```
DepImp -keyrelabel=<old label> <new label>
```

Multithreading

Parallel and efficient processing of requests can be achieved through using a multi CPU machine. The manner in which the requests are balanced over the CPUs is discussed in the *Calculator User Guide*. The number of CPUs used is defined by the MaxWorkers setting. By default the MaxWorkers setting is set to the number of available CPUs. This includes hyperthreaded and dual core CPUs.

When other tasks must also run on the Calculator host, the Calculator can be set to use less CPUs by decreasing the MaxWorkers setting, thus leaving more processor time for the other tasks. This decreases the Calculator performance in terms of scalability and increases the performance of the other tasks.

- ▶ The Calculator uses a pool of threads to handle Requests from clients. Each request is handled in a single thread.
- ▶ The size of the pool of threads is configurable through the setting MaxWorkers in the Calculator.Settings file. The default setting is equal to the number of processors on the host machine, because the Calculator uses almost 100% CPU of a processor during the calculation of a Request.

- ▶ When all threads of the thread pool are in use, incoming Requests are queued until a thread becomes available.

❗ On an AIX platform with multiple CPUs the environment variables `MALLOCMULTIHEAP` and `AIXTHREAD_MNRATIO` improves the performance of the Calculator drastically (Do not use this settings for the Embedded Calculator for Java).

Usage:

export `MALLOCMULTIHEAP = true`, and export `AIXTHREAD_MNRATIO=1:1`

On the AIX platform the `LIBPATH` environment variable must not contain a reference to the `/usr/lib` directory while this result in using the non multi thread safe system libraries, instead of using the thread safe system libraries.

On the AIX platform setting the `MALLOCTYPE` environment variable to `buckets` improves the Calculator performance drastically (Do not use this settings for the Embedded Calculator for Java).

Usage:

export `MALLOCTYPE=buckets`.

Multiple Instances of Calculator

It is possible to run several ProductXpress Calculators on one machine. The procedures for creating additional instances on the Windows and UNIX platform differ, due to differences in operating system architecture. For the Windows platform there are two options to add a new instance:

- ▶ using the Calculator Management Console
- ▶ using the command line utility

On UNIX, creating additional instances is a manual process, similar to installing the default (first) instance of Calculator.

❗ After creating an additional instance using the command line, you might need to restart the Calculator Console in order to be able to manage the new instance.

Multiple instances are not advised for production use. Apart from some additional overhead of manually upgrading each Calculator instance separately, concurrent usage of multiple instances can potentially cause performance degradation or scalability issues due to underlying contention of limited system resources.

Each Calculator instance is installed separately and has its own precision setting. However caution must be exercised when comparing tests results (both in Designer and Calculator) using different precision settings. It is possible for tradeoffs between performance and precision to be made.

Multiple Instances on Windows

In order to run multiple Calculators on one machine, the instance should be created using the Calculator Management Console or the command line utility, `CalcInstance.exe`, located in the `Bin` directory of the default (first) Calculator installation directory.

CalcInstance.exe Usage

With the `CalcInstance` tool it is possible to create, list and delete Calculator Instances.

Before creating the instance it is very important to consider the following:

- ▶ The Calculator installation from which the instance will be copied must be running in production mode, as such it must not have any tracing set. The easiest way to check and set this is to open the Calculator Console and open the Settings dialogue under the Calculator menu. The 'Trace Value' must be set to 'Production Mode (no trace)'.
- ▶ If an existing Calculator instance has been created (and deleted) and you wish to recreate this instance using the same name, you must ensure that the original directory has been deleted. So for example if you have created an instance called 'MyInstance' in C:\Program Files\Hewlett-Packard\ProductXpress\Calculator\Bin\MyInstance' and then subsequently deleted this instance, you must make sure that the directory MyInstance has been deleted before recreating 'MyInstance'. Note that this is only necessary when recreating the instance with the same name.

Creating an instance through the Calculator Management Console:

1. Select 'Create Instance' from the Calculator menu.
2. Enter a name and optionally a port number for the instance.

All Calculator instances are listed as 'Managed Calculators' in the Calculator Console.

Creating a New Instance with CalInstance.exe Tool

1. Open a DOS prompt and type 'CalInstance' at the command line.
2. Enter 'create' at the prompt and press return.
3. Enter a name for the instance and press return.
4. Enter a new port number or press return to accept the suggested port number.
5. Select the base directory. This should be the install directory of the default Calculator. If the default Calculator is installed at another place, you must enter the full path here. Press return

The new instance is now created.

Listing Instances with the CalInstance Tool

1. Open a DOS prompt and type 'CalInstance' at the command line.
2. Enter 'list' at the prompt and press return.

Deleting an Instance with the Calculator Management Console

1. Select 'delete instance' from the Calculator menu.
2. In the dialog box select the instance to delete and select 'delete'.

Deleting an Instance with the CalInstance Tool

1. Open a DOS prompt and type 'CalInstance' at the command line.
2. Enter 'delete' at the prompt and press return.
3. Enter the name of the instance that you want to delete and when prompted enter 'yes' or 'no'.

Multiple Instances on UNIX

In order to run several Calculators on one UNIX machine, follow the steps below:

1. All Calculators should have their own set of files. In order to support that, a copy of the complete installation directory should be made.

i On **UNIX** it is possible instead of copying, to create symbolic links to all the sub-directories except for the etc and deployment directories.

2. After copying the directory, edit the **Calculator.Settings** file in the **Etc** directory and add or change the setting for the Listen port:

```
<ListenPort value='49444' />
```

Every Calculator should have a unique port number.

i **Calculator.Settings** is an XML file and it should therefore contain valid XML. The ListenPort element should be a direct child of the Calculator element. XML is case sensitive.

3. Set the environment variable FIA_CALCULATOR_DIR and let it point to the new Root directory, and add the bin directory to your path.

i This variable should set only for the processes that it applies to and not at a global level.

sh example:

```
FIA_CALCULATOR_DIR=/usr/local/tstcalc
PATH=$FIA_CALCULATOR_DIR/bin:$PATH
export FIA_CALCULATOR_DIR PATH
```

4. Run Deplmp to load the required deployment packages. When using multiple instances, the `-server=<instance name>` option should be used in order to indicate the Calculator instance on which Deplmp should be run.

i Make sure the FIA_CALCULATOR_DIR environment variable points to the correct Calculator root directory prior to executing Deplmp (This is the installation directory of the Calculator for which the deployment packages must be loaded). Deplmp will load the deployment package for the Calculator to which the FIA_CALCULATOR_DIR environment variable points. To establish this it is possible to open a shell window specific for each configured/installed Calculator and set the FIA_CALCULATOR_DIR environment variable in each shell window to point to the desired Calculator. It is also possible to create simple script files for the execution of Deplmp for each configured/installed Calculator. Inside such a script the FIA_CALCULATOR_DIR environment variable can be set to point to the desired Calculator root directory.

Example of the content of such a script for ksh shell:

```
export FIA_CALCULATOR_DIR=<calculator root directory>.
```

The Embedded Calculator

This section looks at the technical details of the Embedded Calculator, such as:

- ▶ The configuration and maintenance of the Embedded Calculator. It gives details of how to configure and tune the Embedded Calculator through the settings files after installation.
- ▶ How to interface the Embedded Calculator using the C programming language.
- ▶ How to interface the Embedded Calculator using Java.
- ▶ How to interface the Embedded Calculator using .Net.
- ▶ The runtime considerations for the Embedded Calculator such as multithreading and the creation of deployment objects.

Embedded Calculator Configuration and Maintenance

- ❗ The Embedded Calculator requires that at least 24 MBytes of stack space is available for it (not counting stack space needed for the rest of the application that makes use of the Calculator). When the available stack size is less than this amount, the Embedded Calculator could crash when performing calculations that require a lot of stack space (such as recursive calculations).

Configuration of the Calculator is done through the following files:

- ▶ Operational Settings files
- ▶ Error Conversion (on page [219](#)) Configuration settings file
- ▶ **ExternalFunctions.Settings** file

- ❗ Configuration files are loaded on initialization of the Embedded Calculator (or on first usage of the configuration information). Therefore, the process in which the Calculator is embedded must be restarted after changing the configuration.

Operational Settings Files

The Embedded Calculator can be configured through the **.Settings** files, which are located in the **Etc** directory of the installation. The **.Settings** files are saved in XML⁵ format.

The main element of a **.Settings** file has the same name as the file name without the extension.

Each "setting" is represented by an XML-element in the **.Settings** file. The value is specified by the "value" attribute.

Example: **Calculator.Settings** file, containing setting "Trace" with value "Clc" :

```
<?xml version="1.0"?>
<Calculator>
    <Trace value="Clc"/>
</Calculator>
```

The Embedded Calculator can be configured through the following operational settings files:

- Calculator.Settings
- TableImp.Settings

Calculator.Settings

Setting Name	Description	Default Value
Trace	It logs extra information. For further details please see the Tracing (on page 277) section.	-
TraceFile	It specifies the filename, relative to the working directory, where the tracing output is placed. Note that this file is created or overwritten when the embedded calculator is initialized, and should not be moved or deleted when the process that uses the Embedded Calculator is running.	-
Mode	It specifies the mode in which Calculator runs; the following two modes are available: test: The Test Mode of Calculator allows packages of checked out deployments to be loaded and tested in Calculator. production: The Production Mode of Calculator allows packages of checked in deployments to be loaded and tested in Calculator.	production
ProductCacheCheckInterval	It determines the interval, in seconds, between the checks on the size of the Global cache.	5 seconds
PolicyCacheUpperBound	It specifies the maximum size of the Policy cache. When the Policy Cache exceeds this size, it is reduced to the PolicyCacheLowerBound size. The size of the Policy Cache is checked each time an element is added.	1000000
PolicyCacheLowerBound	It specifies the size to which the Policy cache is reduced, when it exceeds the PolicyCacheUpperBound size.	800000
NoProductCache	When this setting has the value "true", no Global cache is used.	false

Setting Name	Description	Default Value
ProductCacheCleanup	It determines if the ProductCache size is kept within bounds.	true
ProductCacheAgingFactor	It ensures that values that are not reused anymore will be removed eventually. Increase this value if you suspect that values that were cached in the beginning are preventing other, more reusable values from staying cached. Value may be set between 0 and 100.	10
MaxNodeDepth	<p>Maximum expression node depth (Used to limit stack usage for recursive calculations).</p> <p>Can only be set lower than the default value.</p> <p>Note that in C/C++, it's possible for the stack space to grow indefinitely and there is no proactive mechanism for restricting this space before a failure may occur. Therefore, a safeguard is introduced with this setting whereby the inevitable error situation can be potentially captured and reported in a user-friendly manner rather than a failure.</p> <hr/> <p>i It is the responsibility of the application that uses the Embedded Calculator to configure the stack space for the Calculator. Use trial-and-error tuning to determine the right value for this setting for a specific stack size configuration.</p>	50000

i **product cache:** contains items of which the value does not change between instances of a product (policies); this cache remains in memory between requests

i **policy cache:** contains items of which the value can change between instanced of a product; this cache is emptied after each request

TableImp.Settings

Setting Name	Description	Default Value
Trace	It logs extra information. For further details please see the Tracing (on page 277) section.	-
TraceFile	It specifies the filename, relative to working directory, where tracing output is placed.	-

Interfacing the Embedded Calculator

There are three kinds of embedded Calculators:

- ▶ Embedded Calculator with a C-interface (Calculator.dll, libCalculator.so on AIX and Solaris) can be used from C, C++ and other languages that enable to interface with C-functions like Cobol.
- ▶ Embedded Calculator for Java (CalculatorJNI.dll (libCalculatorJNI.so on AIX and Solaris), and jar file containing Java classes).
- ▶ Embedded Calculator for .NET (CalculatorCOM.dll).

The following sections describe each interface in detail.

Embedded Calculator with a C-interface

The Embedded Calculator (Calculator.dll, libCalculator.so on AIX and Solaris) with the C-interface is using mainly character strings in its interface. These character strings should all be UTF8 encoded.

The interface consists of:

- ▶ Utility functions for: Initialization, Memory management and Error retrieval
- ▶ Functions to load/unload deployment packages/objects
- ▶ Functions for external table management
- ▶ Functions for performing calculations according pull model
- ▶ Functions for performing calculations according push-model
- ▶ Callback function definitions for performing calculation according pull-model
- ▶ Format definitions used by the various functions

The Interface specification is described as follows.

Interface specification for C

The following sections describe the interface specification for C.

Utility Functions

The following describes the utility functions for C.

initializeCalculator

```
int initializeCalculator(const char* calculatorDir);
```

Allocate a memory buffer for use by the Calculator.

Parameter	Input/Output	Description
calculatorDir	in	The installation directory of the Calculator, like "C:\Program Files\Hewlett-Packard\ProductXpress\Calculator". Note that if no path is provided, the path from the environment variable FIA_CALCULATOR_DIR is used as the installation root directory of the Calculator.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

allocBuffer

```
int allocBuffer(int size, char** buffer);
```

Allocate a memory buffer for use by the Calculator.

Parameter	Input/Output	Description
size	in	The size of the buffer to allocate.
buffer	out	Pointer to the allocated buffer.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

freeBuffer

```
void freeBuffer(char* buffer);
```

Free a memory buffer that was allocated by the Calculator.

Parameter	Input/Output	Description
buffer	In	Pointer to the memory buffer that must be freed.

freeArrayOfBuffers

```
void freeBuffer(char** array);
```

Free an array of buffers that was allocated by the Calculator.

Parameter	Input/Output	Description
array	in	Pointer to the memory array that must be freed.

getLastPxError

```
const char* getLastPxError();
```

Get the last error that has been occurred.

Note that each thread has its own error buffer. This means that this function should be called in the same thread as the error occurred to obtain the corresponding error. It also means that other threads do not interfere/overwrite this error when an error occurs in that thread.

Returns 0 if no last error is available, or a pointer to a character string that contains the error. The use of this character pointer is limited up to the next function call (in the same thread) to the calculator dll.

loadDeploymentPackage

```
int loadDeploymentPackage(const char* filename, const char* date,
const char* name, const char* uri);
```

Parameter	Input/Output	Description
filename	in	The filename (including path) of the deployment package file (*.pxdpz or *.pxdp) to load.
date	in	This is an optional parameter (0 if not used) that contains a new deployment date (By default the deployment date from the deployment package file shall be used. Format: YYYY-MM-DD).
name	in	This is an optional parameter (0 if not used) that contains a new deployment name (By default the deployment name from the deployment package file shall be used).
uri	in	This is an optional parameter (0 if not used) that contains an URI that shall be used by the calculator in the input and output XML of push calculations (by default it is the URI as provided at the creation of the deployment package in Designer or "http://www.example.org/<name of the deployment component>" when non is provided in designer>, where non NCName ³ characters in the deployment component name are replaced by "_").

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

loadDeploymentObject

```
int loadDeploymentObject(const char* filename, const char* date,
const char* name, const char* uri);
```

Load a deployment object package (*.pxdo file) in the Calculator.

Parameter	Input/Output	Description
filename	in	The filename (including path) of the deployment object package file (*.pxdo) to load (Created through the DepComp utility).

Parameter	Input/Output	Description
date	in	This is an optional parameter (0 if not used) that contains a new deployment date (By default the deployment date from the deployment package file shall be used. Format: YYYY-MM-DD).
name	in	This is an optional parameter (0 if not used) that contains a new deployment name (By default the deployment name from the deployment package file shall be used).
uri	in	This is an optional parameter (0 if not used) that contains an URI that shall be used by the calculator in the input and output XML of push calculations (by default it is the URI as provided at the creation of the deployment package in Designer or "http://www.example.org/<name of the deployment component>" when non is provided in designer>, where non NCName ³ characters in the deployment component name are replaced by "_").

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

unloadDeploymentPackages

```
int unloadDeploymentPackages();
```

Unloads all loaded deployment packages.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

getRuntimeConfigInfo

```
int getRuntimeConfigInfo(char** info);
```

Returns information about loaded deployment packages and their runtime configuration.

Parameter	Input/Output	Description
info	out	An XML formatted character string that contains info about loaded deployment packages and their runtime configuration (see "Runtime Configuration Info (see Runtime Configuration Information " on page 141)"). The character pointer (if not 0) should be freed with the <code>freeBuffer</code> function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

addTypeConfig

```
int addTypeConfig(const char* resourceConfigId, const char* typeConfigId);
```

Add a type configuration (deployed type deployment) to a resource configuration (configuration of a deployed entry point deployment).

Parameter	Input/Output	Description
resourceConfigId	in	The Id of the resource configuration (can be obtained by calling "getRuntimeConfigInfo").
typeConfigId	in	The Id of the type configuration (can be obtained by calling "getRuntimeConfigInfo").

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

removeTypeConfig

```
int removeTypeConfig(const char* resourceConfigId, const char* typeConfigId);
```

Removes a type configuration (deployed type deployment) from a resource configuration (configuration of a deployed entry point deployment).

Parameter	Input/Output	Description
resourceConfigId	in	The ID of the resource configuration (can be obtained by calling "getRuntimeConfigInfo").
typeConfigId	in	The ID of the type configuration (can be obtained by calling "getRuntimeConfigInfo").

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

setExternalResourceSelector

```
int setExternalResourceSelector(const char* resourceConfigId, const char* externalResourceRootId, const char* name, const char* version, const char* date);
```

Sets a resource selector for an external resource root in the resource configuration of a deployed entry point deployment.

Parameter	Input/Output	Description
resourceConfigId	in	The Id of the resource configuration (can be obtained by calling "getRuntimeConfigInfo").
externalResourceRootId	in	The Id of the external resource root (can be obtained by calling "getRuntimeConfigInfo").
name	in	The name of the resource configuration (deployed entry point deployment) to use for the external resource root.

Parameter	Input/Output	Description
version	in	This is an option parameter (0 if not used) that contains the version of the resource configuration that has to be used for the external resource root.
date	in	This is an option parameter (0 if not used) that contains the date at which the resource configuration that has to be used for the external resource root must be valid (format: YYYY-MM-DD).

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

resetExternalResourceSelector

```
int resetExternalResourceSelector(const char* resourceConfigId,
const char* externalResourceRootId);
```

Resets the resource selector for an external resource root in the resource configuration of a deployed entry point deployment.

Parameter	Input/Output	Description
resourceConfigId	in	The Id of the resource configuration (can be obtained by calling "getRuntimeConfigInfo").
externalResourceRootId	in	The Id of the external resource root (can be obtained by calling "getRuntimeConfigInfo").

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

deployExternalTableDataToFile

```
int deployExternalTableDataToFile(const char* filename, int
preload);
```

Deploys table data, for an external table, that is located in the provided file to the Calculator, and stores the data in a locally managed file.

Parameter	Input/Output	Description
filename	in	The filename (including path) of the file that contains the external table data that is to be deployed.
preload	in	Indicates if the external table data must be preloaded (loaded at initialization, value is !0) or loaded at first use (value is 0).

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

deployExternalTableDataToDatabase

```
int deployExternalTableDataToDatabase(const char* filename, int
preload, const char* dbSpec, int cacheSize, const char* tableName);
```

Deploys table data, for an external table, that is located in the provided file to the Calculator, and stores the data in the provided database.

Parameter	Input/Output	Description
filename	in	The filename (including path) of the file that contains the external table data that is to be deployed.
preload	in	Indicates if the external table data must be preloaded (loaded at initialization, value is !0) or loaded at first use (value is 0).
dbSpec	in	Specifies the connection string for the database in which to put the table data.
cacheSize	in	The amount of database rows that should be cached by the calculator.
tableName	in	The name of the table to be used. (If not provided (0) the name from the file is taken) (Note that this is only needed when there is already a table with the same name existing in the database).

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

updateExternalTableData

```
int updateExternalTableData(const char* filename);
```

Updates table data of an external table for which the table data is already deployed.

Parameter	Input/Output	Description
filename	in	The filename (including path) of the file that contains the updated table data for an external tab

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

removeExternalTableData

```
int removeExternalTableData(const char* tableId);
```

Removes deployed table data of an external table.

Parameter	Input/Output	Description
tableId	in	The ID of the external table for which the table data has to be removed.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

deployedExternalTableDataInfo

```
int deployedExternalTableDataInfo(char** tableDataInfo);
```

Returns info on all deployed external tables.

Parameter	Input/Output	Description
tableDataInfo	out	An XML formatted character string that contains info about deployed external tables (see "External Table Info (see External Table Information " on page 143)"). The character pointer (if not 0) should be freed with the freeBuffer.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

clearGlobalCalculatorCache

```
int clearGlobalCalculatorCache ();
```

Clears the global Calculator cache (cache which contains info that is shared between different Calculator sessions).

This can be used in certain situations, such as when an external table is updated or when the database accessed by an external function has changed, to ensure the Calculator uses the new data.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

Pull-interface Functions

This section describes the Pull-interface functions.

openCalcSession

```
int openCalcSession(const char* sharedLib, const char* deplName,
const char* deplVersion,
const char* deplDate, long* sessionId);
```

Open a session with the Calculator.

Parameter	Input/Output	Description
sharedLib	in	The name of the shared library (dll) that implements the callback functions. These callback functions must be named.
deplName	in	The name of the resource configuration (deployed entry point deployment) to use for the calculations.
deplVersion	in	The version of the resource configuration to use for the calculations (optional, 0 if not used and cannot be used in combination with deplDate).
deplDate	in	The activation date of the resource configuration to use for the calculations (optional, 0 if not used and cannot be used in combination with deplVersion).

Parameter	Input/Output	Description
sessionId	out	Returns the ID of the opened calculator session.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

openRecordingCalcSession

```
int openRecordingCalcSession(const char* sharedLib, const char*
deplName, const char* deplVersion,
const char* deplDate, const char* recordingFilename, long*
sessionId);
```

Open a recording session with the Calculator.

Parameter	Input/Output	Description
sharedLib	in	The name of the shared library (dll) that implements the callback functions. These callback functions must be named.
deplName	in	The name of the resource configuration (deployed entry point deployment) to use for the calculations.
deplVersion	in	The version of the resource configuration to use for the calculations (optional, 0 if not used and cannot be used in combination with deplDate).
deplDate	in	The activation date of the resource configuration to use for the calculations (optional, 0 if not used and cannot be used in combination with deplVersion).
recordingFilename	in	The filename including path of the recording file to create for this session.
sessionId	out	Returns the id of the opened calculator session.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

closeCalcSession

```
int closeCalcSession(long sessionId);
```

Close a Calculator session.

After calling this function, the sessionId cannot be used to reference a Calculator session.

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session that is to be closed.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

calculateValues

```
int calculateValues(long sessionId, const char* features, const char* times, char** Values);
```

Calculate the values for the provided features on the provided times (optional).

Parameter	Input/Output	Description
sessionId	in	The ID of the Calculator session to be used.
features	in	An XML formatted input string that provides references to the features that must be calculated (see "feature references (on page 144)" for XML format).
times	in	An optional (0 if no specific time must be used) XML formatted input string that provides the times for which the provided features must be calculated (see "time list (on page 143)" for XML format).
values	out	An XML formatted output string that contains the result of the calculation (see "feature values" for XML format). The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

calculateValue

```
int calculateValue(long sessionId, const char* instanceId, const char* feature, const char* times, const char* arguments, char** Values);
```

Calculate the value for the provided feature and arguments on the provided times (optional).

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session to be used.
instanceId	in	The ID of the instance to which the feature belongs.
feature	in	The ID of the feature to calculate (can be obtained via getId function).
times	in	An optional (0 if no specific time must be used) XML formatted input string that provides the times for which the provided features must be calculated (see "time list (on page 143)" for XML format).
arguments	in	The arguments needed to calculate the feature value(s).

Parameter	Input/Output	Description
values	out	An XML formatted output string that contains the result of the calculation (see "feature values (on page 144)" for XML format). The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

validate

```
int validate(long sessionId, const char* instanceId, const char*
validationId, char*** result);
```

Perform validation of the provided validation reference.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator to be used.
instanceId	In	The ID of the instance for which the validation is referenced.
validationId	In	The ID of the validation to validate. Uniquely defines the validation within the provided instance.
result	Out	Pointer to a list of validation messages (XML formatted). The list and the contained character pointers (if not 0) should be freed with the freeArrayOfBuffers function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

invalidateInstance

```
int invalidateInstance(long sessionId, const char* instanceId);
```

Invalidates the cached values for all features of the given instance, as well as all cached values that depend on the value of those features.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator session to be used.
instanceId	In	The ID of the instance that has to be invalidated.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

invalidateFeature

```
int invalidateFeature(long sessionId, const char* instanceId, const
char* featureId);
```

Invalidates the cached value for the given feature, as well as all cached values that depend on the feature's value.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator session to be used.
instanceId	In	The ID of the instance for which a feature has to be invalidated.
featureId	In	The ID of the feature to be invalidated.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

invalidateLink

```
int invalidateLink(long sessionId, const char* instanceId, const char* linkId);
```

Invalidates any cached data that is dependent on this link.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator session to be used.
instanceId	In	The ID of the instance for which link has to be invalidated.
linkId	In	The ID of the link to be invalidated.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

invalidateAllFilteredData

```
int invalidateAllFilteredData(long sessionId);
```

Invalidates any cached data that is dependent on filtered data, including all results from previous calls to getFilteredData.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator session to be used.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

endInvalidation

```
int endInvalidation(long sessionId);
```

Signals the end of cache invalidation prior to a new calculation.

Parameter	Input/Output	Description
sessionId	In	The ID of the Calculator session to be used.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

getId

```
int getId(long sessionId, const char * path, char **id);
```

Get the ID of a component given the path to the component.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
path	in	The path of the component that should be returned. (see <i>ProductXpress Path Language Reference Guide</i> for the syntax).
id	out	Pointer to the returned id. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

aggregatedLink

```
int aggregatedLink(long sessionId, const char* reference, char** result);
```

Returns the path to the aggregated link of the aggregation that is referenced.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference to the component.
result	Out	The path to the aggregated link of the aggregation. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

aggregationType

```
int aggregationType(long sessionId, const char* reference, char** result);
```

The type of an aggregation.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference of the link.
result	Out	The type of the aggregation. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

allowsSubtypes

```
int allowsSubtypes(long sessionId, const char* reference, int* result);
```

Returns whether or not the link that is referenced allows subtypes of the structure element it refers to.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the link.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

componentID

```
int componentID(long sessionId, const char* reference, char** result);
```

Returns the ID of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned component type. The character pointer (if not 0) should be freed with the <code>freeBuffer</code> function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

componentType

```
int componentType(long sessionId, const char* reference, char** result);
```

Returns the type of the component that is referenced.

("attribute" | "dynamic deployment environment" | "error message" | "external function" | "function" | "function variable" | "link" | "operation" | "static deployment environment" | "stereotype" | "structure element" | "table" | "tag" | "text block" | "validation" | "validation group" | "value definition" | "variable")

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned component type. The character pointer (if not 0) should be freed with the <code>freeBuffer</code> function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

creationDate

```
int creationDate(long sessionId, const char* reference, char** result);
```

Returns the *creation date* of the component that is referenced; the format is yyyy-mm-dd.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned creation date. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

dataType

```
int dataType(long sessionId, const char* reference, char** result);
```

Returns the *data type* of the value definition that is referenced.

```
("boolean" | "date" | "datetime" | "identity" | "integer" | "quantity"  
| "real" | "string" | "tableId")
```

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the value definition.
result	out	Pointer to the returned data type. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

defaultValue

```
int defaultValue(long sessionId, const char* reference, char** result);
```

Returns the *default value* of a feature or variable that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the feature or variable.
result	out	Pointer to the returned default value. The character pointer (if not 0) should be freed with the freeBuffer function.

documentation

```
int documentation(long sessionId, const char* reference, char** result);
```

Returns the *documentation* of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned documentation. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

entryPoint

```
int entryPoint(long sessionId, char** result);
```

Returns the *entry point definition* of the selected deployment that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
result	out	Pointer to the returned entry point reference. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

evaluatePath

```
int evaluatePath(long sessionId, const char* reference, const char* path, char*** result);
```

Returns zero or more *references to components* that result from the evaluation of the path where the optional Reference parameter is used as the starting point.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	Reference of the component to start the path evaluation from (null if starting at root element).
path	In	The path to evaluate (See the <i>ProductXpress Path Language Reference Guide</i>).
result	Out	Pointer to the returned list of references. The list and the contained character pointers (if not 0) should be freed with the freeArrayOfBuffers function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

instantiable

```
int instantiable(long sessionId, const char* navigationName, char*** result);
```

Returns zero or more *references to instantiable structure elements with a given navigation name*.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
navigationName	In	The navigation name of the structure element to search for.
result	Out	Pointer to the returned list of references. The list and the contained character pointers (if not 0) should be freed with the <code>freeArrayOfBuffers</code> function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

instantiables

```
int instantiables(long sessionId, char*** result);
```

Returns the references of all structure elements that can be instantiated.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
result	Out	Pointer to the returned list of references. The list and the contained character pointers (if not 0) should be freed with the <code>freeArrayOfBuffers</code> function.

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

isAbstract

```
int isAbstract(long sessionId, const char* reference, int* result);
```

Returns whether or not the structure element that is referenced is abstract.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the structure element.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true).

Returns 0 if successful, 1 if error (use `getLastPxError` to obtain details about the error).

isDeferred

```
int isDeferred(long sessionId, const char* reference, int* result);
```

Returns whether or not the mapping that is referenced is deferred (a.k.a. external).

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the mapping.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true).

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isEntryPoint

```
int isEntryPoint(long sessionId, const char* reference, int* result);
```

Returns whether or not the structure element that is referenced is a resource root candidate.

Parameter	Input/Output	Description
sessionId	in	The session id of the Calculator to be used.
reference	in	The reference of the structure element.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isExternal

```
int isExternal(long sessionId, const char* reference, int* result);
```

Returns whether or not the table that is referenced is external.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the table.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true).

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isFinal

```
int isFinal(long sessionId, const char* reference, int* result);
```

Returns whether or not the link that is referenced is final.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the link.

Parameter	Input/Output	Description
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isInherited

```
int isInherited(long sessionId, const char* reference, const char*
parentReference, int* result);
```

Returns whether or not the Feature or Link (Reference) is inherited in Structure Element.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the feature or link.
parentReference	in	Reference of the structure element.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isInstantiable

```
int isInstantiable(long sessionId, const char* reference, int*
result);
```

Returns whether or not the Structure Element (Reference) is instantiable.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the structure element.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isMandatory

```
int isMandatory(long sessionId, const char* reference, int* result);
```

Returns whether or not the link that is referenced is mandatory.

Parameter	Input/Output	Description
sessionId	in	The session id of the Calculator to be used.
reference	in	The reference of the link.

Parameter	Input/Output	Description
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isMappedHere

```
int isMappedHere(long sessionId, const char* reference, int* result);
```

Returns whether or not the mapping that is referenced is mapped here.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the mapping.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isPartial

```
int isPartial(long sessionId, const char* reference, int* result);
```

Returns whether or not the tuple mapping that is referenced is partially mapped.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the tuple mapping.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isResourceRootCandidate

```
int isResourceRootCandidate(long sessionId, const char* reference, int* result);
```

Returns whether or not the structure element that is referenced, or any of its super types, is a resource root candidate (see isEntryPoint).

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference to the component.
result	Out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isTimeDependent

```
int isTimeDependent(long sessionId, const char* reference, int* result);
```

Returns whether or not the value definition that is referenced is time dependent.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the value definition.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

isEntered

```
int isEntered(long sessionId, const char* reference, int* result);
```

Returns whether or not the value source that is referenced is entered.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the value source.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

linkMultiplicity

```
int linkMultiplicity(long sessionId, const char* reference, char** result);
```

Returns the multiplicity of the link that is referenced.

("multiple" | "single" | "zero")

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the link.
result	out	Pointer to the returned multiplicity. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

linkType

```
int linkType(long sessionId, const char* reference, char** result);
```

Returns the type of the link that is referenced.

```
("complement part" | "navigation" | "owning" | "resource")
```

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the link.
result	out	Pointer to the returned link type. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

mappingPath

```
int mappingPath(long sessionId, const char* reference, char** result);
```

Returns the *mapping path* of the mapping or mapping slice that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the mapping or mapping slice.
result	out	Pointer to the returned mapping path. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

mapResultParameterName

```
int mapResultParameterName(long sessionId, const char* reference, char** result);
```

Returns the *name of the map result parameter used in the reduce expression of a custom aggregation that is referenced*.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference of the component.
result	Out	Pointer to the returned name. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

navigationList

```
int navigationList(long sessionId, const char* structReference, const char* linkReference, char*** result);
```

Returns a list of references to the components that are referenced through the navigation link.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
structRefence	in	The reference of the structure element.
linkReference	in	The reference of the navigation link
result	out	Pointer to the returned list of references. The list and the contained character pointers (if not 0) should be freed with the freeArrayOfBuffers function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

navigationName

```
int navigationName(long sessionId, const char* reference, char** result);
```

Returns the *navigation name* of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned navigation name. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

name

```
int name(long sessionId, const char* reference, char** result);
```

Returns the *name* of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned name. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

overrides

```
int overrides(long sessionId, const char* reference, int* result);
```

Returns whether or not the component (link, feature or value source) that is referenced overrides its base type.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the link feature or value source.
result	out	Pointer to the returned boolean value. (0 is false and !0 is true)

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

reduceResultParameterName

```
int reduceResultParameterName(long sessionId, const char* reference,
char** result);
```

Returns the *name of the reduce result parameter used in the finalize expression of a custom aggregation that is referenced*.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference of the component.
result	Out	Pointer to the returned name. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

remoteMappingMode

```
int remoteMappingMode(long sessionId, const char* reference, char**
result);
```

Returns the *mapping mode* of the mapping that is referenced.

("in composition ancestor" | "in subtype")

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the mapping.
result	out	Pointer to the returned mapping mode. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

shortName

```
int shortName(long sessionId, const char* reference, char** result);
```

Returns the *short name* of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned short name. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

subtypesOf

```
int subtypesOf(long sessionId, const char* reference, char*** result);
```

Returns zero or more *references to structure elements that are sub-types of the provided reference*.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	Reference of the enumeration value.
result	Out	Pointer to the returned list of references. The list and the contained character pointers (if not 0) should be freed with the freeArrayOfBuffers function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

symbol

```
int symbol(long sessionId, const char* reference, char** result);
```

Returns the *symbol* of the enumeration that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the enumeration.
result	out	Pointer to the returned symbol. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

table

```
int table(long sessionId, const char* reference, char** result);
```

Returns the contents of the internal table that is referenced as XML.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the internal table.
result	out	Pointer to the returned table content. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

tablePropertyValue

```
int tablePropertyValue(long sessionId, const char* reference, const char* propertyName, char** result);
```

Returns the *value* of the table property of the table that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the internal table.
propertyName	in	The name of the property.
result	out	Pointer to the returned property value. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

tagValue

```
int tagValue(long sessionId, const char* reference, char** result);
```

Returns the *value* of the tagged value that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the tagged value.
result	out	Pointer to the returned tag value. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

text

```
int text(long sessionId, const char* reference, char** result);
```

Returns the *text* of a textblock that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the text block.
result	out	Pointer to the returned text. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

typeExtension

```
int typeExtension(long sessionId, const char* reference, char** result);
```

Returns the *type extension* of the document that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the document.
result	out	Pointer to the returned type extension. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

valueDomain

```
int ValueDomain(long sessionId, const char* reference, char** result);
```

Returns the *value domain* of a value definition that is referenced.

Output examples:

```
<1;10]; [20;30>
```

```
a;b;c
```

Parameter	Input/Output	Description
sessionId	in	The session ID of the Calculator to be used.
reference	in	The reference of the value definition.
result	out	Pointer to the returned value domain. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

valueSourceType

```
int valueSourceType(long sessionId, const char* reference, char** result);
```

Returns the type of the value source that is referenced.

Parameter	Input/Output	Description
sessionId	In	The session ID of the Calculator to be used.
reference	In	The reference of the component.
result	Out	Pointer to the returned value source type. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

versionNumber

```
int versionNumber(long sessionId, const char* reference, char** result);
```

Returns the version number of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned version number string. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

visibility

```
int visibility(long sessionId, const char* reference, char** result);
```

Returns the visibility of the component that is referenced.

Parameter	Input/Output	Description
sessionId	in	The session ID of the calculator to be used.
reference	in	The reference of the component.
result	out	Pointer to the returned version number string. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

Callback Functions for Pull-interface

This section describes the Callback functions for the Pull-interface.

PxGetDefinitionId

```
typedef int (*PxGetDefinitionId)(long sessionId, const char* instanceId, char** definitionId);
```

Interface definition of the callback function to obtain the definition ID of a certain instance.

Parameter	Input/Output	Description
sessionId	in	The ID of the Calculator session for which the callback is called.
instanceId	in	Pointer to the instance ID character string.
definition	out	Pointer to a character pointer that shall be set to point to a definition id character string. The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !=0 if error.

PxGetParent

```
typedef int (*PxGetParent) (long sessionId, const char* instanceId,
char** parentInstanceId, char** linkId);
```

Interface definition of the callback function to obtain the instance ID of the parent of the provided instance id.

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session for which the callback is called.
instanceId	in	Pointer to the instance ID character string for which the parent is requested.
parentInstanceId	out	Pointer to a character pointer that shall be set to point to the instance id character string of the parent. The character string must have been allocated with the allocBuffer function.
linkId	out	Pointer to a character pointer that shall be set to point to the link ID character string of the parent. The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !=0 if error.

PxGetChildren

```
typedef int (*PxGetChildren) (long sessionId, const char* instanceId,
const char* linkId, char** children);
```

Interface definition of the callback function to obtain the list of instances owned by the provided instance.

Parameter	Input/Output	Description
sessionId	in	The ID of the Calculator session for which the callback is called.

Parameter	Input/Output	Description
instanceId	in	Pointer to the instance ID character string for which the children are requested.
linkId	in	Pointer to the link id character string for which to return the children.
children	out	Pointer to a character pointer that shall be set to point to an XML formatted character string that describes the child instances (see paragraph 0 (see " children returned by getChildren callback function " on page 145)). The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

PxGetEnteredValues

```
typedef int (*PxGetEnteredValues)(long sessionId, const char*
instanceId, const char* featureId, char** Values);
```

Interface definition of the callback function to obtain the values of entered features. Note that entered values can not change within a session.

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session for which the callback is called.
instanceId	in	Pointer to the instance id character string for which the values are requested.
featureId	in	Pointer to the feature ID character string for which the values are requested.
values	out	Pointer to a character pointer that shall be set to point to an XML formatted character string that contains the value of the requested entered feature, and optionally other entered feature values (to limit the number of callback calls) (see feature values (on page 144)). The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

PxGetLinkedResources

```
typedef int (*PxGetLinkedResources)( long sessionId, const char*
instanceId, const char* linkId, char** instanceIds);
```

Interface definition of the callback function to obtain the instance ID of a linked resource.

Parameter	Input/Output	Description
sessionId	in	The ID of the Calculator session for which the

Parameter	Input/Output	Description
		callback is called.
instanceId	in	Pointer to the instance ID character string for which to return the linked resource instance ids.
linkId	in	Pointer to the link ID character string for which to return the linked resource instance IDs.
instanceIds	out	Pointer to a character pointer that shall be set to point to an XML formatted character string that contains the list of instance ids (see linked resources returned by getLinkedResources callback function (on page 145)). The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

PxGetResourceRoots

```
typedef int (*PxGetResourceRoots)( long sessionId, const char*
baseTypeId, char** instanceIds);
```

Interface definition of the callback function to obtain the ID of the selected value source of a feature.

Parameter	Input/Output	Description
sessionId	in	The ID of the Calculator session for which the callback is called.
baseTypeId	in	Pointer to the definition id of the type for which to return the resource roots.
instanceIds	out	Pointer to a character pointer that shall be set to point to an XML formatted character string that contains the list of instance ids (see linked resources returned by getLinkedResources callback function (on page 145)). The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

PxGetSelectedValueSource

```
typedef int (*PxGetSelectedValueSource)( long sessionId, const char*
instanceId, const char* featureId, char** ValueSource);
```

Interface definition of the callback function to obtain the ID of the selected value source of a feature.

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session for which the callback is called.

Parameter	Input/Output	Description
instanceId	in	Pointer to the instance ID character string of the instance that contains the feature.
featureId	in	Pointer to the feature id character string for which to return the value source.
valueSource	out	Pointer to a character pointer that shall be set to point to the selected value source ID character string. The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

PxGetFilteredData

```
typedef int (*PxGetFilteredData)( long sessionId, const char* base,
const char* filterExpression, const char* parameterValues, char**
result);
```

Interface definition of the callback function to obtain the ID of the selected value source of a feature.

Parameter	Input/Output	Description
sessionId	in	The ID of the calculator session for which the callback is called.
base	in	Pointer to the instance used as the base (or '.') of the filter expression.
filterExpression	in	Pointer to the filter, specified as a PxPath expression.
parameterValues	in	Pointer to a list of names of variables used in the filterExpression with their value (encoded as XML).
result	out	Pointer to a character pointer that is a list of instances matching the filter ids (see result returned by result returned by getFilteredData callback function (on page 147)). The result is only valid if pointer !NULL is returned. The character string must have been allocated with the allocBuffer function.

Returns 0 if successful, !0 if error.

Push-interface Functions

This section describes the Push-interface functions.

calculate

```
int calculate( const char* input, char** output);
```

Perform calculations according *push*-model.

Parameter	Input/Output	Description
input	in	An XML formatted request string that complies to the Calculator 3.5.x format.
output	out	An XML formatted result string that complies to the Calculator 3.5.x format. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

Transformation Interface Functions

This section describes Transformation Interface functions.

calculateAndTransform

```
int calculateAndTransform( const char* calcInput, const char*
xsltFilename, const char** parameters, char** transformResult);
```

Perform calculations according *push*-model.

Parameter	Input/Output	Description
calcInput	in	An XML formatted request string that complies to the Calculator 3.5.x format.
xsltFilename	in	The name of the XSLT file to use for the transformation (Should be present in the XSL subdirectory of the installation directory).
parameters	in	The parameters for the XSLT transformation. This is an array of pointers to character strings which contains parameter name and parameter value pairs. The last entry in the array must be a NULL pointer.
transformResult	out	The result of the transformation. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

transform

```
int transform( const char* xmlInput, const char* xsltFilename, const
char** paramaters, const char* configName, const char* configVersion,
const char* configDate, char** transformResult);
```

Perform calculations according *push*-model using Calculator request format.

Parameter	Input/Output	Description
xmlInput	in	An XML formatted string that is used as the

Parameter	Input/Output	Description
		input XML for the transformation.
xsltFilename	in	The name of the XSLT file to use for the transformation (Should be present in the XSL subdirectory of the installation directory).
parameters	in	The parameters for the XSLT transformation. This is an array of pointers to character strings which contains parameter name and parameter value pairs. The last entry in the array must be a NULL pointer.
configName	in	The name of the resource configuration (deployed entry point deployment) to be used by the ProductXpress XSLT extension functions in the transformation.
configVersion	in	The version of the resource configuration to be used by the ProductXpress XSLT extension functions in the transformation (optional, 0 if not used and cannot be used in combination with configDate).
configDate	in	The activation date of the resource configuration to be used by the ProductXpress XSLT extension functions in the transformation (optional, 0 if not used and cannot be used in combination with configVersion).
transforResult	out	The result of the transformation. The character pointer (if not 0) should be freed with the freeBuffer function.

Returns 0 if successful, 1 if error (use getLastPxError to obtain details about the error).

XML Format Definitions

The format definitions are described using RNC notation.

Used Namespaces

In the definitions the following namespaces are used:

```
xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
```

Runtime Configuration Information

The runtime configuration info is XML structured info about the runtime configuration of the Embedded Calculator. This info is returned by the function "getRuntimeConfigInfo":

```
start = runtimeConfig
runtimeConfig = element RuntimeConfig
```

```

{
    resourceConfig *,
    typeConfig *
}
resourceConfig = element ResourceConfig
{
    attribute id { xsd:string },
    attribute name { xsd:string },
    attribute version { xsd:string },
    attribute activeDate { xsd:date },
    attribute expiryDate { xsd:date } ?,
    typeConfigRef *,
    externalResourceRoot *
}
typeConfig = element TypeConfig
{
    attribute id { xsd:string },
    attribute name { xsd:string },
    attribute version { xsd:string }
}
typeConfigRef = element TypeConfig
{
    attribute id { xsd:string }
}
externalResourceRoot = element ExtResRoot
{
    attribute id { xsd:string },
    attribute name { xsd:string },
    attribute version { xsd:string },
    resourceSelector
}
resourceSelector = element ResourceSelector
{
    attribute name { xsd:string },
    attribute version { xsd:string } ?,
    attribute activeDate { xsd:date } ?
}

```

Examples of runtime configuration info XML:

```

<?xml version='1.0'?>
<RuntimeConfig>
  <ResourceConfig id='23a5081b-8f7c-4618-b966-5b1d89c7fd48'
    name='8 base elements dynamic deployment'
    version='0.4' activeDate='2006-07-31'>
    <TypeConfig id='ble226f5-804c-480d-baf4-d73d5ab52120' />
  </ResourceConfig>
  <TypeConfig id='ble226f5-804c-480d-baf4-d73d5ab52120'
    name='8 subchild1 type deployment' version='0.4' />
</RuntimeConfig>

<?xml version='1.0'?>
<RuntimeConfig>
  <ResourceConfig id='23c36793-5a2f-4ed6-ac58-80f2daa84b1f'
    name='4 pt dyn' version='0.4' activeDate='2006-07-26'>
    <ExtResRoot id='8b816bd6-3f6d-4007-9293-a1e92e749b3b'
      name='set output types' version='1.0'>
      <ResourceSelector name='myName' />
    </ExtResRoot>
  </ResourceConfig>
</RuntimeConfig>

```

```

        </ExtResRoot>
    </ResourceConfig>
</RuntimeConfig>

```

External Table Information

The external table info is XML structured info about the deployed external tables for the Embedded Calculator . This info is returned by the function "deployedExternalTableDataInfo":

```

start = extTables
extTables = element ExtTables
{
    extTableInfo *
}
extTableInfo = element ExtTableInfo
{
    id,
    name,
    documentation,
    preload,
    dbSpec ?,
    cacheSize ?
}
id = element Id
{
    xsd:string
}
name = element Name
{
    xsd:string
}
documentation = element Documentation
{
    xsd:string
}
preload = element Preload
{
    "true" | "false"
}
dbSpec = element DbSpec
{
    xsd:string
}
cacheSize = element CacheSize
{
    xsd:integer
}

```

time list

The time list is input for the calculateValues and calculateValue functions. It indicates the calculation times.

```

start = times
times = element Times
{
    time *
}

```

```
time = element Time
{
    xsd:dateTime {pattern = ".*T[^Z+-.]*"}
}
```

Example:

```
<?xml version="1.0"?>
<Times>
    <Time>2003-11-09T00:00:00</Time>
    <Time>2003-11-10T00:00:00</Time>
</Times>
```

feature references

Feature references are used as input for the `calculateValues` function.

```
start = featureRefs

featureRefs = element FeatureRefs
{
    featureRef *
}

featureRef = element FeatureRef
{
    attribute feature-id { xsd:string },
    attribute instance-id { xsd:string }
}

Example:

<?xml version="1.0"?>
<FeatureRefs>
    <FeatureRef instance-id='inputDate'
        feature-id='fff90325-084b-42a4-b5ce-dada2eb07a97' />
</FeatureRefs>
```

feature values

Feature values are used by the `getEnteredValues` and `getExternalResourceValues` functions as input and by the `calculateValues` and `calculateValue` functions as output.

```
start = featureValues
featureValues = element FeatureValues
{
    featureValue *
}

featureValue = element FeatureValue
{
    featureRef,
    value *
    # see Calculator Input-Output Specification for the Push
    # Calculator
}

featureRef = element FeatureRef
{
    attribute feature-id { xsd:string },
    attribute instance-id { xsd:string }
}
```

Example:

```
<?xml version="1.0"?>
<FeatureValues>
  <FeatureValue>
    <FeatureRef instance-id='inputDate'
      feature-id='fff90325-084b-42a4-b5ce-dada2eb07a97' />
    <Value>2003-11-10</Value>
  </FeatureValue>
  <FeatureValue>
    <FeatureRef instance-id='inputBoolean'
      feature-id='3fe85fed-2b95-4f2f-8599-20705bf4e89e' />
    <Value>true</Value>
  </FeatureValue>
  <FeatureValue>
    <FeatureRef
instance-id='inputString'                                feature-id='40cad859-3691-41
c8-afac-cfc56f1e74a9' />
    <Value>entered string</Value>
  </FeatureValue>
  <FeatureValue>
    <FeatureRef
instance-id='inputInteger'                                feature-id='c4c716f3-2fc9-4
d4e-af6e-cc1cf553c1ca' />
    <Value>3</Value>
  </FeatureValue>
  <FeatureValue>
    <FeatureRef instance-id='inputReal'
      feature-id='58a31a4c-33fc-47c2-9446-fe8d5de6e07a' />
    <Value>3.14</Value>
  </FeatureValue>
</FeatureValues>
```

children returned by getChildren callback function

```
start = children

children = element Children
{
  instanceId *
}

instanceId = element InstanceId
{
  xsd:string
}
Example:
<?xml version='1.0'?>
<Children>
  <InstanceId>MyInstanceId1</InstanceId>
  <InstanceId>MyInstanceId2</InstanceId>
</Children>
```

linked resources returned by getLinkedResources callback function

```
start = linkedResources
```

```
linkedResources = element LinkedResources
{
    instanceId *
}
```

```
instanceId = element InstanceId
{
    xsd:string
}
```

Example:

```
<?xml version='1.0'?>
<LinkedResources>

    <InstanceId>MyInstanceId1</InstanceId>

    <InstanceId>MyInstanceId2</InstanceId>

</LinkedResources>
```

resource roots returned by getResourceRoots callback function

```
start = resourceRoots
```

```
resourceRoots = element ResourceRoots
{
    instanceId *
}
```

```
instanceId = element InstanceId
{
    xsd:string
}
```

Example:

```
<?xml version='1.0'?>
<ResourceRoots>
    <InstanceId>MyInstanceId1</InstanceId>
    <InstanceId>MyInstanceId2</InstanceId>
</ResourceRoots>
```

parameter values for getFilteredData callback function

```
start = pxPathBinding
```

```
pxPathBinding = element PxPathBinding
{
    variable *
}
```

```
variable = element clc:Variable
{
    pxType, # see next section for definition
    value
    # see Calculator Input-Output Specification for the Push
    # Calculator
}
```


result returned by getFilteredData callback function

```

start = result

result = element Result
{
    pxType,
    value
    # see Calculator Input-Output Specification for the Push
    # Calculator
}

pxType = element clc:PxType
{
    typeData
}

typeData =
(
    attribute type { xsd:string },
    # can be one of the following values:
    # "boolean", "integer", "real", "string", "guid", "date",
    # "tableid", "valuedomainid", "tuplememberid", "identity",
    # "path", "tuple", "set", "join", "valuedomain", "matchtype",
    # "quantity", "unit", "instance", "function",
    # "tdboolean", "tdinteger", "tdreal", "tdstring", "tdguid",
    # "tddate", "tdtableid", "tdvaluedomainid",
    "tdtuplememberid",
    # "tdidentity", "tdpath", "tdtuple", "tdset", "tdjoin",
    # "tdvaluedomain", "tdmatchtype",
    # "tdquantity", "tdunit", "tdinstance", "tdfunction",

    # For tuple types required
    tupleAttributes ?,
    tupleMembers ?,

    # For types that contain types like set or valuedomain
    containerType ?
)

containerType = element clc:ContainerType
{
    typeData
}

tupleAttributes =
(
    attribute id { xsd:string },
    attribute name { xsd:string }
)

tupleMembers =
(
    tupleMember *
)

tupleMember = element clc:TupleMember

```

```
{
    pxType
}
```

Example:

```
<?xml version='1.0'?>
<ResourceRoots>
    <InstanceId>MyInstanceId1</InstanceId>
    <InstanceId>MyInstanceId2</InstanceId>
</ResourceRoots>
```

error returned by getLastPxError function

```
start = error

error = element Error
{
    attribute date { xsd:dateTime {pattern = ".*T[^Z+-.*]" } } ?,
    attribute id { xsd:string },
    attribute nr { xsd:integer },
    xsd:string
}
```

Example:

```
<?xml version='1.0'?>
<Error date='2003-11-10T13:00:00'
    id='Clc.DEPLOYMENT_NAME_NOT_FOUND' nr='46003'>
    Error FiaStd.FUNCFAILED [001000] in
    DeploymentSettings::getDeployment(), module Clc: Function
    Failed
    Error Clc.DEPLOYMENT_NAME_NOT_FOUND [046003] in
    DeploymentSettings::getDeployment(), module Clc: Deployment
    'MyDeploymentName' is unknown.
</Error>
```

Embedded Calculator for Java

The Embedded Calculator for Java consists of a dll/shared library (CalculatorJNI.dll, libCalculatorJNI.so on AIX and Solaris) that implements a JNI interface for the Calculator, and three jar files (pxjavaruntime.jar, FiaJNI.jar and PxVal.jar) that contain a set of Java classes/interfaces that enable the client to interface with the Calculator.

Interface Specification for Java

This section describes the interface specification for Java.

Interface PxCalculatorSession

Interface definition for a pull interface session with the ProductXpress Calculator.

```
com.solcorp.productxpress.calculator.spec.PxCalculatorSession
```

calculateValues

```
public ArrayList calculateValues(ArrayList features, ArrayList
dates) throws PxException
```

Calculate features values for the provided feature references, using the provided calculation dates (which can be empty).

Parameter	Description
features	The references of features that must be calculated (PxFeatureRef).
dates	The calculation dates (which can be empty) (java.util.Date).

Returns the feature values of the features that are calculated (PxFeatureValue).

calculateValue

```
public ArrayList calculateValue(PxFeatureRef feature, ArrayList
dates, ArrayList args) throws PxxException
```

Calculate feature value for the provided feature reference, using the provided calculation dates (which can be empty) and the provided arguments (which can be empty).

Parameter	Description
feature	The reference of feature that must be calculated.
dates	The calculation dates (which can be empty) (java.util.Date).
args	The arguments (which can be empty) (PxCalcValue).

Returns the feature values of the features that are calculated (PxFeatureValue).

invalidateFeature

```
public void invalidateFeature(PxFeatureRef feature) throws
PxxException
```

Invalidates the cached value for the given feature, as well as all cached values that depend on the feature's value.

Parameter	Description
feature	The feature whose value should be invalidated.

invalidateLink

```
public void invalidateLink(PxLinkRef link) throws PxxException;
```

Invalidates any cached data that is dependent on this link.

Parameter	Description
link	The link to invalidate.

invalidateInstance

```
public void invalidateInstance(String instanceId) throws
PxxException;
```

Invalidates the cached values for all features of the given instance, as well as all cached values that depend on the value of those features.

Parameter	Description
instanceId	The ID for the instance to be invalidated.

endInvalidation

```
public void endInvalidation() throws PxxException;
```

Signals the end of cache invalidation prior to a new calculation.

closeSession

```
public void closeSession() throws PxxException
```

Closes this session with the Calculator. After this, the Calculator session cannot be used anymore.

getId

```
public String getDefinitionId(String path);
```

Returns the ID of a model element identified by its path in the financial product structure.

Parameter	Description
path	ID to get. (See <i>ProductXpress Path Language Reference Guide</i>).

allowsSubtypes

```
public boolean allowsSubtypes(String reference) throws PxxException;
```

Returns whether or not the link that is referenced allows subtypes of the structure element it refers to.

Parameter	Description
reference	Reference of the link.

componentID

```
public String componentID(String reference) throws PxxException;
```

Returns the ID of the component that is referenced.

Parameter	Description
reference	Reference of the component.

componentType

```
public String componentType(String reference) throws PxxException;
```

Returns the type of the component that is referenced.

```
("attribute" | "dynamic deployment environment" | "error message" |
"external function" | "function" | "function variable" | "link" |
"operation" | "static deployment environment" | "stereotype" |
"structure element" | "table" | "tag" | "text block" | "validation"
| "validation group" | "value definition" | "variable")
```

Parameter	Description
reference	Reference of the component.

creationDate

```
public String creationDate(String reference) throws PxxException;
```

Returns the *creation date* of the component that is referenced; the format is yyyy-mm-dd.

Parameter	Description
reference	Reference of the component.

dataType

```
public String dataType(String reference) throws PxxException;
```

Returns the *data type* of the value definition that is referenced.

```
("boolean" | "date" | "datetime" | "identity" | "integer" | "quantity"  
| "real" | "string" | "tableId")
```

Parameter	Description
reference	Reference of the value definition.

defaultValue

```
public String defaultValue(String reference) throws PxxException;
```

Returns the *default value* of a feature or variable that is referenced.

Parameter	Description
reference	Reference of the feature or variable.

documentation

```
public String documentation(String reference) throws PxxException;
```

Returns the *documentation* of the component that is referenced.

Parameter	Description
reference	Reference of the component.

entryPoint

```
public String entryPoint() throws PxxException;
```

Returns the reference to the *entry point* of the selected deployment.

evaluatePath

```
public ArrayList evaluatePath(String reference, String path) throws  
PxxException;
```

Returns zero or more *references to components* that result from the evaluation of the path where the optional Reference parameter is used as the starting point.

Parameter	Description
reference	Reference of the component to start the path evaluation from (null if starting at root element).
path	The path to evaluate (See <i>ProductXpress Path Language Reference Guide</i>).

instantiable

```
public ArrayList instantiable(String navigationName) throws
PxException;
```

Returns zero or more *references to instantiable structure elements* with the provided navigation name.

Parameter	Description
navigationName	The navigation name of the instantiable structure element.

instantiables

```
public ArrayList instantiables() throws PxException;
```

Returns all *references to structure elements* that can be instantiated.

isAbstract

```
public boolean isAbstract(String reference) throws PxException;
```

Returns whether or not the structure element that is referenced is abstract.

Parameter	Description
reference	Reference of the structure element.

isDeferred

```
public boolean isDeferred(String reference) throws PxException;
```

Returns whether or not the mapping that is referenced is deferred (a.k.a. external).

Parameter	Description
reference	Reference of the mapping.

isEntryPoint

```
public boolean isEntryPoint(String reference) throws PxException;
```

Returns whether or not the structure element that is referenced is a resource root candidate.

Parameter	Description
reference	Reference of the structure element.

isExternal

```
public boolean isExternal(String reference) throws PxException;
```

Returns whether or not the table that is referenced is external.

Parameter	Description
reference	Reference of the table.

isFinal

```
public boolean isFinal(String reference) throws PxxException;
```

Returns whether or not the link that is referenced is final.

Parameter	Description
reference	Reference of the link.

isInherited

```
public boolean isInherited(String reference, String parentReference)
throws PxxException;
```

Returns whether or not the Feature or Link (Reference) is inherited in Structure Element.

Parameter	Description
reference	Reference of the feature or link.
parentReference	Reference of the structure element.

isInstantiable

```
public boolean isInstantiable(String reference) throws PxxException;
```

Returns whether or not the Structure Element (Reference) is instantiable.

Parameter	Description
reference	Reference of the link.

isMandatory

```
public boolean isMandatory(String reference) throws PxxException;
```

Returns whether or not the link that is referenced is mandatory.

Parameter	Description
reference	Reference of the link.

isMappedHere

```
public boolean isMappedHere(String reference) throws PxxException;
```

Returns whether or not the mapping that is referenced is mapped here.

Parameter	Description
reference	Reference of the mapping.

isPartial

```
public boolean isPartial(String reference) throws PxxException;
```

Returns whether or not the tuple mapping that is referenced is partially mapped.

Parameter	Description
reference	Reference of the tuple mapping.

isTimeDependent

```
public boolean isTimeDependent(String reference) throws PxException;
```

Returns whether or not the value definition that is referenced is time dependent.

Parameter	Description
reference	Reference of the value definition.

linkMultiplicity

```
public String linkMultiplicity(String reference) throws PxException;
```

Returns the multiplicity of the link that is referenced.

("multiple" | "single" | "zero")

Parameter	Description
reference	Reference of the link.

linkType

```
public String linkType(String reference) throws PxException;
```

Returns the type of the link that is referenced.

("complement part" | "navigation" | "owning" | "resource")

Parameter	Description
reference	Reference of the link.

mappingPath

```
public String mappingPath(String reference) throws PxException;
```

Returns the *mapping path* of the mapping or mapping slice that is referenced.

Parameter	Description
reference	Reference of the mapping or mapping slice.

navigationList

```
public ArrayList navigationList(String structReference, String linkReference) throws PxException;
```

Returns zero or more *references to links* that are part of the navigation list of the referenced structure element and navigation link.

Parameter	Description
structReference	Reference of the structure element.

Parameter	Description
linkReference	Reference of the navigation link.

navigationName

```
public String navigationName(String reference) throws PxException;
```

Returns the *navigation name* of the component that is referenced.

Parameter	Description
reference	Reference of the component.

name

```
public String name(String reference) throws PxException;
```

Returns the *name* of the component that is referenced.

Parameter	Description
reference	Reference of the component.

overrides

```
public boolean overrides(String reference) throws PxException;
```

Returns whether or not the component (link, feature or value source) that is referenced overrides its base type.

Parameter	Description
reference	Reference of the link, feature or value source.

remoteMappingMode

```
public String remoteMappingMode(String reference) throws PxException;
```

Returns the *mapping mode* of the mapping that is referenced.

("in composition ancestor" | "in subtype")

Parameter	Description
reference	Reference of the mapping.

shortName

```
public String shortName(String reference) throws PxException;
```

Returns the *short name* of the component that is referenced.

Parameter	Description
reference	Reference of the component.

subtypesOf

```
public ArrayList subtypesOf(String reference) throws PxException;
```

Returns references to the subtypes of the referenced structure element.

Parameter	Description
reference	Reference of the structure element.

symbol

```
public String symbol(String reference) throws PxxException;
```

Returns the *symbol* of the enumeration that is referenced.

Parameter	Description
reference	Reference of the enumeration.

table

```
public String table(String reference) throws PxxException;
```

Returns the contents of the internal table that is referenced as XML.

Parameter	Description
reference	Reference of the internal table.

tablePropertyValue

```
public String tablePropertyValue(String reference, String
propertyName) throws PxxException;
```

Returns the *value* of the table property of the table that is referenced.

Parameter	Description
reference	Reference of the table.
propertyName	Name of the property.

tagValue

```
public String tagValue(String reference) throws PxxException;
```

Returns the *value* of the tagged value that is referenced.

Parameter	Description
reference	Reference of the tagged value.

text

```
public String text(String reference) throws PxxException;
```

Returns the *text* of a textblock that is referenced.

Parameter	Description
reference	Reference of the text block.

typeExtension

```
public String typeExtension(String reference) throws PxxException;
```

Returns the *type extension* of a document that is referenced.

Parameter	Description
reference	Reference of the document.

valueDomain

```
public String ValueDomain(String reference) throws PxxException;
```

Returns the *value domain* of a value definition that is referenced.

Output examples:

```
<1;10];[20;30>
```

```
a;b;c
```

Parameter	Description
reference	Reference of the value definition.

versionNumber

```
public String versionNumber(String reference) throws PxxException;
```

Returns the version number of a component that is referenced.

Parameter	Description
reference	Reference of a component.

visibility

```
public String visibility(String reference) throws PxxException;
```

Returns the visibility of a component that is referenced.

Parameter	Description
reference	Reference of a component.

Interface PxCalculatorHome

This is the interface definition for the factory that is responsible for the creation of pull Calculator sessions, push Calculator objects, management of the runtime configuration and management of deployed external tables.

```
com.solcorp.productxpress.calculator.spec.PxCalculatorHome
```

initialize

```
public void initialize(String calculatorRootPath)
    throws PxxException
```

Initializes the Calculator.

Parameter	Description
calculatorRootPath	The installation directory of the Calculator, for example "C:\Program Files\Hewlett-Packard\ProductXpress\Calculator". Note that if no path is provided, the path from the environment variable FIA_CALCULATOR_DIR is used as the installation root directory of the Calculator.

createCalculatorSession

```
public PxCalculatorSession createCalculatorSession(String name,
String version, Date date, InstanceStore instanceStore) throws
PxException
```

Creates a session for the pull Calculator. The created session shall use an entry point deployment configuration that is selected by the name, version and date criteria that is provided. The provided instance store interface shall be used to obtain additional data when required by the pull Calculator.

Parameter	Description
name	The name of the entry point deployment configuration to use in the created session.
version	Optional (null if not used) the version of the entry point deployment configuration to use in the created session. Cannot be used in combination with the date parameter.
date	Optional (null if not used) the active date of the entry point deployment configuration to use in the created session. Cannot be used in combination with the version parameter.
instanceStore	The data provider that will be used by the created session to obtain additional data when required by the calculator.

Returns a new object instance of a PxCalculatorSession.

createCalculatorSession

```
public PxCalculatorSession createCalculatorSession(String name,
String version, Date date, InstanceStore instanceStore, PxRecorder
recorder) throws PxException
```

Creates a recording session for the pull Calculator. The created session shall use an entry point deployment configuration that is selected by the name, version and date criteria that is provided. The provided instance store interface shall be used to obtain additional data when required by the pull Calculator. The provided recorder shall record the data for the created session.

Parameter	Description
name	The name of the entry point deployment configuration to use in the created session.
version	Optional (null if not used) the version of the entry point deployment configuration to use in the created session.

Parameter	Description
	Cannot be used in combination with the date parameter.
date	Optional (null if not used) the active date of the entry point deployment configuration to use in the created session. Cannot be used in combination with the version parameter.
instanceStore	The data provider that shall be used by the created session to obtain additional data when required by the calculator.
recorder	The recorder to which the data for the created session shall be recorded.

Returns a new object instance of a `PxCalculatorSession`.

createRecorder

```
public PxRecorder createRecorder(String filename)
    throws PjException
```

Creates a recorder to which data for Calculator pull sessions can be recorded.

Parameter	Description
filename	The name (including the path) of the file to store the recorded data for the calculation session(s) in. configuration to use in the created session.

Returns a new object instance of a `PxRecorder`.

getPushCalculator

```
public PxPushCalculator getPushCalculator();
```

Creates a new push Calculator instance.

Returns a new object instance of a `PxPushCalculator`.

loadDeploymentPackage

```
public PxDeployment loadDeploymentPackage(String filename, Date
    date,
        String name, String uri)
    throws PjException
```

Load a deployment package (*.pxdpz or *.pxdp file) in the Calculator.
After this, calculation requests that make use of this deployment package can be issued.

Parameter	Description
filename	The filename (including path) (*.pxdpz or *.pxdp file) of the deployment package to load.
date	This is an optional parameter (null if not used) that contains a new deployment date (by default the deployment date from the deployment package file shall be used).

Parameter	Description
name	This is an optional parameter (null if not used) that contains a new deployment name (by default the deployment name from the deployment package file shall be used).
uri	This is an optional parameter (null if not used) that contains the URI to be used in the input and output XML of push calculator requests (by default it is the URI as provided at the creation of the deployment package in Designer or "http://www.example.org/<name of the deployment component>" when non is provided in designer>, where non NCName ³ characters in the deployment component name are replaced by "_").

loadDeploymentObject

```
public PxDeployment loadDeploymentObject(String filename, Date date,
    String name, String uri)
    throws PxxException
```

Load a deployment object package (*.pxdo file) in the Calculator.

After this, calculation requests that make use of this deployment object package can be issued.

Note that deployment object files created with older versions of ProductXpress are not supported and could if used cause system instability (crashes).

Parameter	Description
filename	The filename (including path) (*.pxdo file) of the deployment object package to load (Created through the DepComp utility).
date	This is an optional parameter (null if not used) that contains a new deployment date (by default the deployment date from the deployment object package file shall be used).
name	This is an optional parameter (null if not used) that contains a new deployment name (by default the deployment name from the deployment object package file shall be used).
uri	This is an optional parameter (null if not used) that contains the URI to be used in the input and output XML of push calculator requests (by default it is the URI as provided at the creation of the deployment package in Designer or "http://www.example.org/<name of the deployment component>" when non is provided in designer>, where non NCName ³ characters in the deployment component name are replaced by "_").

unloadDeploymentPackages

```
public void unloadDeploymentPackages() throws PxxException
```

Unloads all loaded deployment packages.

Note that PxDeployment objects that were created before this unload are not to be used anymore (It references deployments that are no longer loaded).

getDeployments

```
public ArrayList getDeployments();
```

Get the list of loaded deployments and their runtime configuration.

Returns a list of loaded deployments PxDeployment.

deployExternalTableData

```
void deployExternalTableData(String filename, boolean preload)
throws PxxException;
```

Deploys table data, for an external table, that is located in the provided file to the Calculator, and stores the data in a locally managed file.

Parameter	Description
filename	The filename (including path) of the file that contains the external table data that is to be deployed.
preload	Indicates if the external table data must be preloaded (loaded at initialization) or loaded at first use.

```
public void deployExternalTableData(String filename, boolean
preload, String dbSpec, int cacheSize)
throws PxxException;
```

Deploys table data, for an external table, that is located in the provided file to the Calculator, and stores the data in a database. Stores the data in a database table that has the name of the external table as specified in the provided file (As defined in Designer).

Parameter	Description
filename	The filename (including path) of the file that contains the external table data that is to be deployed.
preload	Indicates if the external table data must be preloaded (loaded at initialization) or loaded at first use.
dbSpec	Specifies the connection string for the database in which to put the table data.
cacheSize	The amount of database rows that should be cached by the Calculator.

```
public void deployExternalTableData(String filename, boolean
preload, String dbSpec, String tableName, int cacheSize) throws
PxxException;
```

Deploys table data, for an external table, that is located in the provided file to the Calculator, and stores the data in a database table with the provided table name.

Parameter	Description
filename	The filename (including path) of the file that contains the external table data that is to be deployed.
preload	Indicates if the external table data must be preloaded (loaded at initialization) or loaded at first use.
dbSpec	Specifies the connection string for the database in which to put the table data.
tableName	The name of the table to be used.
cacheSize	The amount of database rows that should be cached by the Calculator.

updateExternalTableData

```
public void updateExternalTableData(String filename) throws
PxException;
```

Updates table data of an external table for which the table data is already deployed.

Parameter	Description
filename	The filename (including path) of the file that contains the updated table data for an external table.

removeExternalTableData

```
public void removeExternalTableData(String tableId) throws
PxException;
```

Removes deployed table data of an external table.

Parameter	Description
tableId	The ID of the external table for which the table data has to be removed.

deployedExternalTableDataInfo

```
public ArrayList deployedExternalTableDataInfo() throws PxException;
```

Get info on all the deployed external table data.

Returns a list of ExternalTableDataInfo objects that contain info about the deployed external table data.

clearGlobalCalculatorCache

```
public void clearGlobalCalculatorCache() throws PxException;
```

Clears the global Calculator cache (cache which contains info that is shared between different Calculator sessions).

This can be used in certain situations, such as when an external table is updated or when the database accessed by an external function has changed, to ensure the Calculator uses the new data.

setTrace

```
public void setTrace(String traceSpec) throws PxxException;
```

Sets which module:class:function should be traced.

Parameter	Description
traceSpec	The trace specification which is formatted identically to the trace value in the Calculator.Settings file.

addLogger

```
public void addLogger(PxxLogger logger) throws PxxException;
```

Adds a logger that shall receive all log info from the Embedded Calculator.

Parameter	Description
logger	A logger implementation that will receive all log info from the Embedded Calculator.

addLoggerForCurrentThread

```
public void addLoggerForCurrentThread(PxxLogger logger) throws PxxException;
```

Adds a logger that shall receive all log info from the Embedded Calculator for the current thread.

Parameter	Description
logger	A logger implementation that will receive all log info for the current thread from the Embedded Calculator.

removeLogger

```
public void removeLogger(PxxLogger logger) throws PxxException;
```

Removes a previously added logger. After this no log information is sent to the provided logger.

Parameter	Description
logger	The logger to remove.

removeAllLoggers

```
public void removeAllLoggers() throws PxxException;
```

Removes all previously added loggers.

Class PxxCalculatorHomeJNI

This is the factory implementation for Calculator and deployment instances. It is a singleton, meaning that there can be no more than one instance. It implements interface 'PxxCalculatorHome'.

com.solcorp.productxpress.calculator.PxxCalculatorHomeJNI

instance

```
public synchronized static PxxCalculatorHome instance()
```

Returns the instance of PxxCalculatorHomeJNI.

Interface PxDeployment

This defines the interface to the configuration of a deployed (loaded) deployment package.

`com.solcorp.productxpress.calculator.spec.PxDeployment`

name

```
public String name()
```

Returns the name of the deployment.

id

```
public String id()
```

Returns the ID of the deployment.

version

```
public String version()
```

Returns the version of the deployment.

activeDate

```
public Date activeDate()
```

Returns the active date of the deployment.

expiryDate

```
public Date expiryDate()
```

Returns the expiry date of the deployment. Note that type deployments don't have an expiry date.

type

```
public int type()
```

Returns the type of the deployment, `PxDeployment.ENTRY_POINT_DEPLOYMENT` for an entry point deployment, `PxDeployment.TYPE_DEPLOYMENT` for a type deployment and `PxDeployment.FUNCTION_DEPLOYMENT` for a function deployment.

usedTypeDeployments

```
public ArrayList usedTypeDeployments()
```

Returns the list of type deployments (`PxDeploymentPackageInfo`) used by the entry point deployment. Returns an empty list for other types of deployments.

externalResourceRoots

```
public ArrayList externalResourceRoots()
```

Returns the list of external resource roots (`PxExternalResourceRoot`) of an entry point deployment.

getResourceSelector

```
public PxResourceSelector  
getResourceSelector(PxExternalResourceRoot externalResourceRoot)  
throws PxException;
```

Get the resource selector for an external resource root of an entry point deployment.

Parameter	Description
externalResourceRoot	The external resource root for which to return the resource selector.

Returns the resource selector for an external resource root of the entry point deployment.

setResourceSelector

```
public void setResourceSelector(PxExternalResourceRoot
externalResourceRoot, PxResourceSelector selector) throws
PxException
```

Set the resource selector for an external resource root in the configuration of the entry point deployment.

Parameter	Description
externalResourceRoot	The external resource root for which to set the resource selector.
selector	The resource selector.

resetResourceSelector

```
public void resetResourceSelector(PxExternalResourceRoot
externalResourceRoot) throws PxException
```

Reset the resource selector for an external resource root in the configuration of the entry point deployment.

Parameter	Description
externalResourceRoot	The external resource root for which to reset the resource selector.

addTypeDeployment

```
public void addTypeDeployment(PxDeployment typeDeployment) throws
PxException
```

Add a type deployment to the configuration of an entry point deployment.

Parameter	Description
typeDeployment	The type deployment to add to the configuration of the entry point deployment.

removeTypeDeployment

```
public void removeTypeDeployment(PxDeployment typeDeployment) throws
PxException
```

Remove a type deployment from the configuration of an entry point deployment.

Parameter	Description
typeDeployment	The type deployment to remove from the configuration of the entry point deployment.

optimize

```
public void optimize() throws PxException
```

Optimize the calculations for the configuration of an entry point deployment.

Interface PxLogger

The interface for loggers that can be registered to receive log information from the Embedded Calculator.

```
com.solcorp.productxpress.calculator.spec.PxLogger
```

log

```
public void log(PxLogInfo info);
```

This function receives log information from the Embedded Calculator.

Parameter	Description
info	The information of a single log.

Interface InstanceStore

This is an abstraction for a system storing instances. Through this interface, ProductXpress is requesting information about instances. The instances are assumed to be identifiable by an `instanceId`, which is opaque to the ProductXpress runtime.

```
com.solcorp.productxpress.calculator.spec.InstanceStore
```

getDefinition

```
public String getDefinition(String instanceId) throws Exception
```

Given an instance ID, return its definition ID.

Parameter	Description
instanceId	The instance ID for which the definition ID is to be returned.

Returns the definition ID that belongs to the instance ID.

getParent

```
public PxLinkRef getParent(String instanceId) throws Exception
```

Given an instance ID, return the instance ID of the owning instance and the link ID on which the given instance was instantiated.

Parameter	Description
instanceId	The instance ID for which its parent must be returned.

Returns the instance ID of the parent and the link ID on which the provided instance ID was instantiated, or null if no parent

getChildren

```
public ArrayList getChildren(PxLinkRef link) throws Exception
```

Given an instance ID and link, get the children for that instance instantiated on that link.

Parameter	Description
link	the instance id and link ID for which to return the children.

Returns a list of instances (instance ids) that are instantiated on the provided link for the provided instance.

getLinkedResources

`public ArrayList getLinkedResources(PxLinkRef link) throws Exception`

Get the instances (IDs) for the root instances of the resources linked at a given instance (ID).

Parameter	Description
link	The instance ID from which the linked resource should be obtained.

Returns a list of instances (IDs) of the linked resources.

getResourceRoots

`public ArrayList getResourceRoots(String rootType) throws Exception`

Get the resource root instances (IDs) for the given definition id.

Parameter	Description
rootType	The definition ID for which to return the root instances.

Returns list of instances (instance IDs) of the resource roots.

getSelectedValueSource

`public String getSelectedValueSource(PxFeatureRef feature) throws Exception`

Get the ID of the value source that must be used for a given feature.

Parameter	Description
feature	Uniquely identifies/references the feature for which to return the selected value source name.

Returns the ID of the value source to use.

getValues

`public ArrayList getValues(PxFeatureRef feature) throws Exception`

Get the value for the given feature.

For a time-dependent feature, its value will be the complete "schedule", i.e. all values for all known dates.

As a performance-aid, the result might contain the values for other features as well.

Parameter	Description
feature	Reference to the feature for which value(s) is/are requested.

Returns a list of feature values for the requested feature and optionally other features. (may return more than requested)(PxFeatureValue).

getFilteredData

```
public PxFilteredData getFilteredData(String baseInstanceId, String
filterExpression, Map<String, PxCalcValue> parameterValues) throws
Exception
```

Find values matching a filter expression, starting from a base instance.

Parameter	Description
baseInstanceId	The instance used as the base (or '.') of the filter expression.
filterExpression	The filter, specified as a ProductXpress path expression.
parameterValues	A map associating the names of variables used in the filterExpression with their value.

Returns PxFilteredData object containing information about the validity of the result and if the value is valid. The result is considered valid if the getFilteredData was able to execute the request.

Class PxPushCalculator

Interface definition for the ProductXpress push-Calculator.

```
com.solcorp.productxpress.Calculator.spec.PxPushCalculator
```

calculate

```
public String calculate( String input ) throws PjException;
```

Perform calculations based on the provided input.

Parameter	Description
input	The input for the calculation(s). It is an XML formatted string conforming to the Calculator Input specification, see Calculator Input-Output Specification for the Push Calculator (on page 194).

Returns the result of the requested calculation(s). It is an XML formatted string conforming to the Calculator Output specification.

calculateAndTransform

```
public String calculateAndTransform(String calcInput, String
xsltFilename, Map parameters ) throws PjException;
```

Perform calculations based on the provided Calculator formatted input.

Parameter	Description
calcInput	The input for the calculation(s). It is an XML formatted string conforming to the Calculator Input specification, see "Calculator Input-Output Specification for the Push

Parameter	Description
	Calculator (on page 194)".
xsltFilename	The name of the XSLT file to use for the transformation (Should be present in the XSL subdirectory of the installation directory).
parameters	Key/value pairs (String objects) of the parameters for the XSLT processor.

Returns the result of the transformation.

transform

```
public String transform( String xmlInput, String xsltFilename, Map
parameters, String configName, String configVersion, String
configDate ) throws PxException;
```

Perform calculations based on the provided Calculator formatted input.

Parameter	Description
xmlInput	The input for the transformation (XML formatted).
xsltFilename	The name of the XSLT file to use for the transformation (Should be present in the XSL subdirectory of the installation directory).
parameters	Key/value pairs (String objects) of the parameters for the XSLT processor.
configName	The name of the entry point deployment configuration to use by ProductXpress extension functions in the transformation.
configVersion	Optional (null if not used) the version of the entry point deployment configuration to be used by ProductXpress extension functions in the transformation. Cannot be used in combination with the configDate parameter.
configDate	Optional (null if not used) the active date of the entry point deployment configuration to be used by ProductXpress extension functions in the transformation. Cannot be used in combination with the configVersion parameter.

Returns the result of the transformation.

Interface PxRecorder

Interface definition of recorder of Calculator sessions.

```
com.solcorp.productxpress.Calculator.spec.PxRecorder
```

stop

```
public void stop()
```

Stops the recording of Calculator session data.

Class PxException

Base class for ProductXpress exceptions. It extends the `java.lang.RuntimeException` class.

```
com.solcorp.productxpress.std.PxException
```

PxException

```
public PxException(String message)
```

Class PxFeatureRef

Reference to a feature of a specific instance.

```
com.solcorp.productxpress.calculator.spec.data.PxFeatureRef
```

PxFeatureRef

```
public PxFeatureRef(String instanceId, String featureId)
```

Constructor, initializes the `PxFeatureRef` with the provided instance ID and the provided definition ID (the combination of an instance ID and a definition ID uniquely identifies a feature).

Parameter	Description
instanceId	The ID of the instance in which the feature is contained.
featureId	The ID that uniquely identifies the feature within the instance with the provided instance ID.

instanceId

```
public String instanceId()
```

Returns the ID of the instance in which the feature is contained.

featureId

```
public String featureId()
```

Returns the ID that uniquely identifies the feature within the instance with the provided instance ID.

Class PxFeatureValue

Represents a feature value.

```
com.solcorp.productxpress.calculator.spec.data.PxFeatureValue
```

PxFeatureValue

```
public PxFeatureValue(PxFeatureRef feature, PxCalcValue Value)
```

Constructor of a feature value that belongs to the provided feature and has the provided value.

Parameter	Description
feature	Reference to the feature for which this the value is contained.
value	The value of the feature, null for so called "null value".

getFeatureRef

```
public PxFeatureRef getFeatureRef()
```

Returns the reference of the feature for which the value is contained.

getValue

```
public PxCalcValue getValue()
```

Returns the value of the feature.

Class PxFilteredData

Represents a the result of a getFilteredData call.

```
com.solcorp.productxpress.calculator.spec.data.PxFilteredData
```

PxFilteredData

```
public PxFilteredData()
```

Constructor of filtered data that has status of invalid.

```
public PxFilteredData(PxCalcValue value)
```

Constructor of a filtered data that constructs a valid data result with the provided value.

Parameter	Description
value	The value of the filtered data.

isValid

```
public boolean isValid()
```

Returns true if filtered data is valid, and therefore contains PxCalcValue.

getValue

```
public PxCalcValue getValue()
```

Returns the value of the filtered data.

Class PxLinkRef

Represents a reference to an instance via a specific link starting from a source instance.

```
com.solcorp.productxpress.calculator.spec.data.PxLinkRef
```

PxLinkRef

```
public PxLinkRef(String instanceId, String linkId)
```

Constructor of a link reference.

Parameter	Description
instanceId	The ID of the instance that is referenced.
linkId	The ID of the link, via which the instance identified by the provided instance ID can be reached, from a specific source instance.

instanceId

```
public String instanceId()
```

Returns the ID of the instance that is referenced via the link.

linkId

```
public String linkId()
```

Returns the ID of the link.

Class PxLogInfo

Contains information of a single log.

```
com.solcorp. productxpress.calculator.spec.data.PxLogInfo
```

PxLogInfo

```
public PxLogInfo(String context, String text, String severity, Date  
dateTime, String hostName, long processId, long threadId)
```

Constructor, initializes the object with the provided values.

Parameter	Description
context	The context in which the log occurred module, class, function.
text	The log text.
severity	The severity of the log.
dateTime	The date and time of the log.
hostName	The host name of the host on which the log occurred.
processId	The ID of the process on which the log occurred.
threadId	The ID of the thread on which the log occurred.

getContext

```
public String getContext()
```

Returns the context in which the log occurred.

getText

```
public String getText()
```

Returns the text of the log.

getSeverity

```
public String getSeverity()
```

Returns the severity of the log.

getDateTime

```
public String getDateTime()
```

Returns the date and time of the log.

getHostName

```
public String getHostName()
```

Returns the host name on which the log occurred.

getProcessId

```
public long getProcessId()
```

Returns the ID of the process in which the log occurred.

getThreadId

```
public long getThreadId()
```

Returns the ID of the thread in which the log occurred.

Class PxExternalResourceRoot

Represents an external resource root from a deployed entry point deployment package.

```
com.solcorp.productxpress.calculator.spec.data.PxExternalResourceRoot
```

PxExternalResourceRoot

```
public PxExternalResourceRoot(String id, String name)
```

Constructor of a link reference.

Parameter	Description
id	The ID of the external resource root.
name	The name of the external resource root.

id

```
public String id()
```

Returns the ID of the external resource root.

name

```
public String name()
```

Returns the name of the external resource root.

Class PxExternalTableDataInfo

A container of information about a deployed external table.

```
com.solcorp.productxpress.calculator.spec.data.PxExternalTableDataInfo
```

PxExternalTableDataInfo

```
public PxExternalTableDataInfo(String name, String id, String documentation, String dbSpec, Boolean inDatabase, Boolean preload, int cacheSize)
```

Constructor of external table data information.

Parameter	Description
Name	The name of the external table.
id	The ID of the external table.
documentation	The documentation of the external table.
dbSpec	The database specification of the external table, if table is located in database (see "External Table Management (on page 18)").
inDatabase	Indicates if table is located in database.
preload	Indicates if table has to be preloaded, not for tables in database.
cacheSize	The cache size in rows for an external table in database.

name

```
public String name()
```

Returns the name of the external table.

id

```
public String id()
```

Returns the ID of the external table.

documentation

```
public String documentation()
```

Returns the documentation of the external table.

dbSpec

```
public String dbSpec()
```

Returns the database specification of the external table, if table is located in database.

inDatabase

```
public boolean inDatabase()
```

Returns true if table is located in database, otherwise false.

isPreloaded

```
public boolean isPreloaded()
```

Returns true if table is pre-loaded into memory (not for tables in database).

cacheSize

```
public int cacheSize()
```

Returns the number of rows that are cached for this table (only for tables in database).

Class PxResourceIdentificationType

Represents the kind of identification used to identify a deployed entry point deployment package (ID or name).

```
com.solcorp.productxpress.calculator.spec.data.PxResourceIdentificationType
```

PxResourceIdentificationType

```
public PxResourceIdentificationType(int type)
```

Constructor of a resource identification type object.

Parameter	Description
type	The type of resource identification. Can have values PxResourceIdentificationType.v_name or PxResourceIdentificationType.v_id.

isName

```
public boolean isName()
```

Returns true if identification type is name.

isId

```
public boolean isId()
```

Returns true if identification type is id.

Class PxResourceSelectionType

Represents the kind of selection used to select a deployed entry point deployment package (latest version, specific version or by date).

```
com.solcorp.productxpress.calculator.spec.data.PxResourceSelectionType
```

PxResourceSelectionType

```
public PxResourceSelectionType(int type)
```

Constructor of a resource selection type object.

Parameter	Description
type	The type of resource selection. Can have values PxResourceSelectionType.v_latestVersion, PxResourceSelectionType.v_explicitversion or PxResourceSelectionType.v_versionActiveOnDate.

isLatestVersion

```
public boolean isLatestVersion()
```

Returns true if latest version is to be selected.

isExplicitVersion

```
public boolean isExplicitVersion()
```

Returns true if an explicit version is to be selected.

isVersionActiveOnDate

```
public boolean isVersionActiveOnDate()
```

Returns true if a version is to be selected that is active on a specific date.

Class PxResourceSelector

Represents a selection of a deployed entry point deployment.

```
com.solcorp.productxpress.calculator.spec.data.PxResourceSelector
```

PxResourceSelector

```
public PxResourceSelector(PxResourceSelector other)
```

Copy constructor of a resource selector.

Parameter	Description
other	The resource selector to copy.

createForName

```
public static PxResourceSelector createForName(String deploymentName)
```

Creates a resource selector for the latest version of a deployed entry point deployment with the provided name.

createForId

```
public static PxResourceSelector createForId(String deploymentId)
```

Creates a resource selector for the latest version of a deployed entry point deployment with the provided ID.

createForNameAndVersion

```
public static PxResourceSelector createForNameAndVersion(String deploymentName, String version)
```

Creates a resource selector for a deployed entry point deployment with the provided name and with the provided version.

createForIdAndVersion

```
public static PxResourceSelector createForIdAndVersion(String deploymentId, String version)
```

Creates a resource selector for a deployed entry point deployment with the provided ID and with the provided version.

createForNameAndDate

```
public static PxResourceSelector createForNameAndDate(String deploymentName, Date date)
```

Creates a resource selector for a deployed entry point deployment with the provided name and is active on the provided date.

createForIdAndDate

```
public static PxResourceSelector createForIdAndDate(String deploymentName, Date date)
```

Creates a resource selector for a deployed entry point deployment with the provided ID and is active on the provided date.

selectionType

```
public PxResourceSelectionType selectionType()
```

Returns the selection type.

identificationType

```
public PxResourceIdentificationType identificationType()
```

Returns the identification type.

deploymentName

```
public String deploymentName()
```

Returns the deployment name.

deploymentId

```
public String deploymentId()
```

Returns the deployment ID.

version

```
public String version()
```

Returns the version to select.

date

```
public Date date()
```

Returns the date at which the deployed entry point deployment must be active.

Class PxTupleMember

Represents a tuple member of a tuple type definition.

```
com.solcorp.productxpress.val.PxTupleMember
```

PxTupleMember

```
public PxTupleMember(PxType type, String name, int index)
```

Constructor.

Parameter	Description
type	The type of the tuple member.
name	The name of the tuple member.
index	The index of the value of the tuple member.

getType

```
public PxType getType()
```

Returns the type of the tuple member.

getName

```
public String getName()
```

Returns the name of the tuple member.

getIndex

```
public int getIndex()
```

Returns the index of the value of the tuple member.

Class PxType

Class that represents the type of a Calculator value (PxCalcValue).

```
com.solcorp.productxpress.val.PxType
```

integerType

```
public static PxType integerType()
```

Returns the created integer type object.

realType

```
public static PxType realType()
```

Returns the created real type object.

stringType

```
public static PxType stringType()
```

Returns the created string type object.

booleanType

```
public static PxType booleanType()
```

Returns the created boolean type object.

dateType

```
public static PxType dateType()
```

Returns the created date type object.

quantityType

```
public static PxType quantityType()
```

Returns the created quantity type object.

setType

```
public static PxType setType(PxType containedType)
```

Returns the created set type object.

Parameter	Description
containedType	The contained type of the set.

tupleType

```
public static PxType tupleType(String name, ArrayList tupleMembers)
```


Returns the created tuple type object.

Parameter	Description
name	The name of the tuple.
tupleMembers	List of tuple members (PxTupleMember).

tableIdType

```
public static PxType tableIdType()
```

Returns the created table ID type object.

tdIntegerType

```
public static PxType tdIntegerType()
```

Returns the created time dependent integer type object.

tdRealType

```
public static PxType tdRealType()
```

Returns the created time dependent real type object.

tdStringType

```
public static PxType tdStringType()
```

Returns the created time dependent string type object.

tdBooleanType

```
public static PxType tdBooleanType()
```

Returns the created time dependent boolean type object.

tdDateType

```
public static PxType tdDateType()
```

Returns the created time dependent date type object.

tdQuantityType

```
public static PxType tdQuantityType()
```

Returns the created time dependent quantity type object.

tdSetType

```
public static PxType tdSetType(PxType containedType)
```

Returns the created time dependent set type object.

Parameter	Description
containedType	The contained type of the set.

tdTupleType

```
public static PxType tdTupleType(String name, ArrayList tupleMembers)
```

Returns the created time dependent tuple type object.

Parameter	Description
name	The name of the tuple.
tupleMembers	List of tuple members (PxTupleMember).

tdTableIdType

```
public static PxType tdTableIdType()
```

Returns the created time dependent table ID type object.

isIntegerType

```
public final boolean isIntegerType()
```

Returns true if type is integer.

isRealType

```
public final boolean isRealType()
```

Returns true if type is real.

isNumericType

```
public final boolean isNumericType()
```

Returns true if type is real or integer.

isStringType

```
public final boolean isStringType()
```

Returns true if type is string.

isBooleanType

```
public final boolean isBooleanType()
```

Returns true if type is boolean.

isDateType

```
public final boolean isDateType()
```

Returns true if type is date.

isQuantityType

```
public final boolean isQuantityType()
```

Returns true if type is quantity.

isSetType

```
public final boolean isSetType()
```

Returns true if type is set.

isTupleType

```
public final boolean isTupleType()
```

Returns true if type is tuple.

isTableIdType

```
public final boolean isTableIdType()
```

Returns true if type is table ID.

isTdIntegerType

```
public final boolean isTdIntegerType()
```

Returns true if type is time dependent integer.

isTdRealType

```
public final boolean isTdRealType()
```

Returns true if type is time dependent real.

isTdStringType

```
public final boolean isTdStringType()
```

Returns true if type is time dependent string.

isTdBooleanType

```
public final boolean isTdBooleanType()
```

Returns true if type is time dependent boolean.

isTdDateType

```
public final boolean isTdDateType()
```

Returns true if type is time dependent date.

isTdQuantityType

```
public final boolean isTdQuantityType()
```

Returns true if type is time dependent quantity.

isTdSetType

```
public final boolean isTdSetType()
```

Returns true if type is time dependent set.

isTdTupleType

```
public final boolean isTdTupleType()
```

isTdTableIdT Returns true if type is time dependent tuple.

isTdTableIdType

```
public final boolean isTdTableIdType()
```

Returns true if type is time dependent table ID.

isTimeDependent

```
public final boolean isTimeDependent()
```

Returns true if type is time dependent.

containedType

```
public PxType containedType()
```

Returns contained type if type is set.

tupleTypeName

```
public String tupleTypeName()
```

Returns tuple name if type is tuple.

tupleMembers

```
public ArrayList tupleMembers()
```

Returns tuple members if type is tuple.

Class PxCalcValue

Class that represents a value for the Calculator.

```
com.solcorp.productxpress.val.PxCalcValue
```

PxCalcValue

```
public PxCalcValue(PxType type)
```

Constructor for a specific type.

Parameter	Description
type	The type of the Calculator value.

getType

```
public PxType getType()
```

Returns the type of the value.

getIntegerValue

```
public Integer getIntegerValue()
```

Returns the integer value in case of an integer type.

getRealValue

```
public BigDecimal getRealValue()
```

Returns the BigDecimal value in case of a real type or an integer type.

getStringValue

```
public String getStringValue()
```

Returns the String value in case of a string type.

getBooleanValue

```
public Boolean getBooleanValue()
```

Returns the Boolean value in case of a boolean type.

getDateValue

```
public Date getDateValue()
```

Returns the Date value in case of a date type.

getSetValue

```
public ArrayList getSetValue()
```

Returns an ArrayList with the values in the set in case of a set type.

addTupleMember

```
public void addTupleMember(String memberName, PxCalcValue memberValue)
```

Add a member to a tuple.

Parameter	Description
memberName	The name of the tuple member to add.
memberValue	The value of the new tuple member.

getTupleMemberValue

```
public PxCalcValue getTupleMemberValue(String tupleMemberName)
```

Returns the PxCalcValue value of a tuple member in case of a tuple type.

Parameter	Description
tupleMemberName	The name of the tuple member.

```
public PxCalcValue getTupleMemberValue(PxTupleMember tupleMember)
```

Returns the PxCalcValue value of a tuple member in case of a tuple type.

Parameter	Description
tupleMember	The tuple member.

getValueHistory

```
public Map getValueHistory()
```

Returns the value history of a time dependent value as a Map of PxCalcValue and Date pairs.

getUnit

```
public String getUnit()
```

Returns the name of a quantity unit in case of a quantity type.

createIntegerValue

```
public static PxCalcValue createIntegerValue(int Value)
```

Creates a PxCalcValue object of type integer and initialized with the provided value.

Parameter	Description
value	The integer value.

createRealValue

```
public static PxCalcValue createRealValue(BigDecimal Value)
```

Creates a PxCalcValue object of type real and initialized with the provided value.

Parameter	Description
value	The real value.

createStringValue

```
public static PxCalcValue createStringValue(String Value)
```

Creates a PxCalcValue object of type string and initialized with the provided value.

Parameter	Description
value	The string value.

createBooleanValue

```
public static PxCalcValue createBooleanValue(boolean Value)
```

Creates a PxCalcValue object of type boolean and initialized with the provided value.

Parameter	Description
value	The boolean value.

createDateValue

```
public static PxCalcValue createDateValue(Date Value)
```

Creates a PxCalcValue object of type date and initialized with the provided value.

Parameter	Description
value	The date value.

createQuantityValue

```
public static PxCalcValue createQuantityValue(BigDecimal Value,  
String unit)
```

Creates a PxCalcValue object of type quantity and initialized with the provided value and unit.

Parameter	Description
value	The real value.
unit	The unit name.

createSetValue

```
public static PxCalcValue createSetValue(PxType containedType,  
ArrayList Value)
```

Creates a PxCalcValue object of type set and initialized with the provided value.

Parameter	Description
containedType	The type contained in the set.
value	The set value.

createTupleValue

```
public static PxCalcValue createTupleValue(PxType tupleType,
ArrayList tupleMemberValues)
```

Creates a PxCalcValue object of provided tuple type and initialized with the provided member values.

Parameter	Description
tupleType	The type of the tuple.
tupleMemberValues	The tuple member values.

```
public static PxCalcValue createTupleValue(String tupleTypeName)
```

Creates a PxCalcValue object of tuple type with provided name.

Parameter	Description
tupleTypeName	The name of the tuple type definition.

createTableIdValue

```
public static PxCalcValue createTableIdValue(String Value)
```

Creates a PxCalcValue object of type table ID and initialized with the provided value.

Parameter	Description
value	The table ID value.

createTdIntegerValue

```
public static PxCalcValue createTdIntegerValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent integer and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent integer (PxCalcValue/Date pairs).

createTdRealValue

```
public static PxCalcValue createTdRealValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent real and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent real (PxCalcValue/Date pairs).

createTdStringValue

```
public static PxCalcValue createTdStringValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent string and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent string (PxCalcValue/Date pairs).

createTdBooleanValue

```
public static PxCalcValue createTdBooleanValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent boolean and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent boolean (PxCalcValue/Date pairs).

createTdDateValue

```
public static PxCalcValue createTdDateValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent date and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent date (PxCalcValue/Date pairs).

createTdQuantityValue

```
public static PxCalcValue createTdQuantityValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent quantity and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent quantity (PxCalcValue/Date pairs).

createTdSetValue

```
public static PxCalcValue createTdSetValue(PxType containedType, Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent set with provided contained type and initialized with the provided value history.

Parameter	Description
containedType	The type contained in the set.
valueHistory	The value history of the time dependent set (PxCalcValue/Date pairs).

createTdTableIdValue

```
public static PxCalcValue createTdTableIdValue(Map ValueHistory)
```

Creates a PxCalcValue object of type timedependent table ID and initialized with the provided value history.

Parameter	Description
valueHistory	The value history of the time dependent table ID (PxCalcValue/Date pairs).

Embedded Calculator for .NET

The COM based interface to the Embedded Calculator is implemented in CalculatorCOM.dll. This interface to the Calculator can be used with any development environment that supports COM (e.g. any of the .NET languages but also unmanaged C++). The type-library in CalculatorCOM.dll describes types similar to the ones found in the JNI package. For a detailed description, see Embedded Calculator for Java (on page [148](#)).

The types exposed (interface, class or enumeration) by CalculatorCOM are:

Type	Description
PxCalculatorHome	The entry point into the object model. This class provides general management functions (e.g. to load a deployment package) and is a factory for PxCalculator instances. Should be initialized first with a call to <i>initialize()</i> .
PxDeployment	Represents a deployment. References to this type should be obtained by accessing the <i>PxCalculatorHome.Deployments</i> collection.
PxCalculatorSession	Represents a pull calculator session. Contains the <i>getValues()</i> method that performs the actual calculation.
PxPushCalculator	Represents a push Calculator.
PxFeatureValue	Represents an feature (in- or output) value.
PxFeatureRef	Contains data that uniquely identifies a feature.
PxLinkRef	Contains data that uniquely identifies a link.
IInstanceStore	The callback (pull) interface to be implemented by the hosting client code.
PxExternalTableDataInfo	Provides information of deployed external tables. Collection can be accessed from PxCalculatorHome.
PxResourceSelectorFactory, PxResourceSelector, PxResourceSelectionType,	Classes related to deployment selection.

Type	Description
PxResourceIdentificationType	
PxExternalResourceRoot	Represents an external resource root.
PxValueType	An enumeration of all the PxValue types (e.g. INTEGER_TYPE, QUANTITY_TYPE etc.).
IPxValue	Base for all PX-data types.
IPxScalarValue	Base for all PX scalar (non-time dependent) types.
IPxTdValue	Base for all time dependent types.
PxBoolean, PxDate, PxString, PxReal, PxInteger, PxQuantity, PxSet, PxTuple, PxInstanceReference	Non time-dependent data types.
PxTdBoolean, PxDtdDate, PxDtdString, PxDtdReal, PxDtdInteger, PxDtdQuantity, PxDtdSet	Time-dependent data types.

To start using the embedded Calculator in a .NET project, a reference to the "CalculatorCOM 1.0 Type Library" needs to be added to the project. All types exposed by the library reside in the CalculatorCOM namespace. Please note that as COM does not support parameterized constructors (Java does), initial values to the Px* value types cannot be passed on instantiation. Instead, the values must be set after an instance of the type has been created.

As all COM classes in the CalculatorCOM library support the *ISupportErrorInfo* interface, errors reported by the calculator are automatically converted into .NET exceptions. The Message member of the raised .NET exception will contain a detailed description of the error.

Runtime Considerations for the Embedded Calculator

The Embedded Calculator provides for a number of runtime possibilities, depending on the requirements for the phase of the project (testing or production) and the required architecture. The following runtime options are described in this section:

- ▶ Multithreading
- ▶ Depcomp

Multithreading

Parallel and efficient processing of requests can be achieved through using a multi CPU machine. The way in which the requests are balanced over the CPUs is described in the *Calculator User Guide*.

The Embedded Calculators are multi thread safe. This means that multiple Calculator sessions can be handled from multiple threads. Note that closing a session with the Calculator from one thread, and performing calculations on that same session from another thread could lead to undefined behaviour. Therefore you should perform all actions for a specific Calculator session in a single thread.

Creation of deployment object files with DepComp

The DepComp tool converts a deployment package file to a deployment object file. It also allows changing the name and activation date of the deployment.

Deployment object files can be used for the Embedded Calculator. The advantage of loading a deployment object file instead of a deployment package in an Embedded Calculator is that deployment object files load much faster than deployment package files.

Depcomp Syntax

DepComp uses the following syntax:

```
DepComp -in=<local deployment file> [-name=<name>] [-actdate=<date>]
| -describe=<binary deployment file> ]
```

 Run DepComp without any arguments .to print a list of the possible options.

in

Converts the provided deployment package file (.pxdpz or .pxdp) to a deployment object file (.pxdo) with same name but with special extension. It can be used in combination with the **name** and **actdate** options to set the name and activation date in the deployment object file.

describe

Describe the content of the provided deployment object file.

Common Items for Service and Embedded Calculator

The following items are described, which are common for both the Service and the Embedded Calculator:

- ▶ How to deal with huge sets of data. For example, how to request the sum of all cash values of a million policies.
- ▶ The syntax of the Input and Output required for the Embedded Push and Service Calculators. Through a series of descriptions and small examples, it looks at the syntax of a request and syntax of the output of a calculation. it further looks are error handling.
- ▶ Sample input and output XML for both the Push Calculator and the Function Calculator.
- ▶ The details XSTL extension functions. The ProductXpress XSTL extension functions are available for XSLT transformations through the transformation API (see Transformation Interface). These extension functions allow information to be obtained from the run-time configuration. It further describes the table XML syntax.
- ▶ How to create, integrate and maintain external functions. External Functions reside outside of the deployment as an entry point in a Dynamic Link Library (DLL) on a Microsoft Windows platform, or as an entry point in a Shared Library on IBM AIX and SUN Solaris platforms. Administration/External System Providers will develop these functions, which contain logic to access the external systems. ProductXpress Calculator provides facilities to invoke External Functions and use the results of External Functions in any calculation.
- ▶ Common configuration and maintenance including logging, tracing, tuning and redistribution of Calculator components.
- ▶ A troubleshooting guide.

Output Aggregation (dealing with huge sets of data)

In areas, such as profit testing and embedded value calculations, performing calculations over very large sets of data (millions of policies) is required. It is not practical nor technically feasible to send all this data at once in one big XML message. Therefore, the Push Mode of ProductXpress Calculator supports the so-called Output Aggregation Protocol as a solution in these cases.

The term resource is used instead of policy or insured person to address policies and also other types of data that can be huge in quantity.

The Protocol

For Output Aggregations, the Map Reduce programming model is used. The Map Reduce programming model is developed for processing **parallelizable** problems concerning large data sets on Computer Clusters. It basically contains two steps: the Map step, which is performed on each parallelizable sub-problem, and the Reduce step, which will combine the results of any number of Map functions.

On a functional level, the following steps are involved in this protocol when sending large sets of data to the Calculator in push mode:

1. Send an aggregation initiation request that contains the requested calculations for the aggregations and the calculation data that is not aggregated. The links in the calculation data that reference the resources over which are to be aggregated, will be identified as

aggregation links. This happens through an identifier (attribute id) and a flag (attribute aggregate).

The Calculator returns an aggregation-definition xml-structure.

2. Send an aggregation map request to the calculator together with the aggregation-definition. The request contains the data of one or more resources. The request is only required to have a CalculationData section.

The DeplRef as well as all Calculation section will be ignored.

The result of this request is a map-result xml-structure. The client collects the map-results returned by the Calculator.

3. When the number of map-results collected by the client becomes too large to handle, it can send them to the Calculator with a aggregation reduce request.

The result of an aggregation reduce call is one map-result xml structure.

4. Repeat steps 2 and 3 until all inputs are processed. Issue an aggregation reduce request with the remaining map-results.
5. Send an aggregation finish request to the Calculator together with the aggregation-definition and the last remaining map-result. The response contains the results of the requested calculations in step 1.

Example

In this example we will request a calculation that will aggregate certain values of a million policies.

Aggregation Initiation Request

This is the first request we send as part of the protocol. Notice the aggregate and id attributes on link "myPolicy".

```
<clc:CalculationInput
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns='http://www.example.org/myDeployment'>
  <clc:DeplR dep-name='myDeployment' ver-sel='last' />
  <clc:CalculationData>
    <Portfolio id='ID1'>
      <Links>
        <myPolicy aggregate='true' id='Portfolio' />
      </Links>
    </Portfolio>
  </clc:CalculationData>
  <clc:Calculation>
    <Portfolio ref='ID1'>
      <Features>
        <aggregated-values-in-tuple />
      </Features>
    </Portfolio>
  </clc:Calculation>
</clc:CalculationInput>
```

Aggregation Map Requests

This is one of the million requests sent over to be aggregated. Notice that the <DeplR> element and the <Calculation> sections are missing. The aggregation-definition is not inside the request, but is sent along with the request.

```
<clc:CalculationInput
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns='http://www.example.org/myDeployment'>
  <clc:CalculationData>
    <myPolicy id='ID2'>
      <Features>
        <IssueDate>2000-01-01</IssueDate>
        <!-- and more -->
      </Features>
      <Links>
        <Insured>
          <clc:Reference ref='ID52' />
        </Insured>
      </Links>
    </myPolicy>
    <Insured id='ID52'>
      <Features>
        <DateOfBirth>1954-02-03</DateOfBirth>
      </Features>
    </Insured>
  </clc:CalculationData>
</clc:CalculationInput>
```

Aggregation Result

When all million requests have been sent, we send a final request that requests the aggregated results that were requested in the initiation request. The result of this request is shown below. Notice the <Aggregation> element that contains a summary of the aggregation. One of the resources seem to have failed as shown by errors=1 (errors show the number of resources that failed). This resource has been discarded during the evaluation of feature *aggregated-values-in-tuple*.

```
<clc:CalculationOutput
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement'
xmlns='http://www.example.org/myDeployment'>
  <clc:RuntimeConfiguration name='myDeployment' version='0.10'
    id='ID:3eefbce4-1123-4883-8c61-e9b262400712' />
  <clc:Calculation>
    <Portfolio ref='ID1'>
      <Features>
        <aggregated-values-in-tuple>
          <number_of_values>999999</number_of_values>
          <number_of_null_values>0</number_of_null_value
s>
          <sum>-532641.0026073</sum>
          <sum_squared>6056518679.0168</sum_squared>
          <sum_third>-96223892597309</sum_third>
          <sum_fourth>1872401800808700000</sum_fourth>
```

```

        <min>-27579.950002209</min>
        <max>169.70594306923</max>
    </aggregated-values-in-tuple>
</Features>
</Portfolio>
</clc:Calculation>
<clc:Aggregation name='Portfolio' status='ok'
instances='999999' errors='1' />
<clc:AggregationDetails>
    <clc:LinkStatus link='Portfolio'
        included-instances=999999'
        skipped-instances='1'>
    <clc:ErrorInstance instance-id='ID:201000'>
        <clc:Error nr='80014' id='ExprNode.NO_VAR_VALUE'>
            [031077] Error occurred while calculating feature
            '@aggregated-values-in-tuple' in Context[[Instance
            '432313ac-a436-492f-afdc-0a8a2a0ddef5:1' (resource root)]:
/myPolicy].
            [008002] Error in @aggregated-values-in-tuple
            [080014] No value found for variable 'Cost of Capital
            Percentage'.
            In function 'risk margin'
        </clc:Error>

    <clc:ErrorInstance>
</clc:AggregationDetails>

</clc:CalculationOutput>

```

Calculator Input-Output Specification for the Push Calculator

This section refers to the Input and Output specification for the Service Calculator and the Embedded Push Calculator.

Requests are sent to and results are returned from the Service Calculator as XML streams.

The root element in a Request stream is `CalculationInput`. The root element in a Result is `CalculationOutput`.

In the case of **static or dynamic deployments** `CalculationInput` consists of:

- ▶ One element (`Deployment`) that references a static or dynamic deployment package.
- ▶ One `CalculationData` element: the data that is needed to perform calculations that are specified in the Calculation elements, which is usually referred to as the *policy* or *illustration data* (depending on the application).
- ▶ Zero or more `Scenario` elements: overwrites the data that is specified in `CalculationData`
- ▶ Zero or more `Calculation` elements: the data that needs to be calculated and returned and the `CalculationDates` on which this data needs to be calculated. If no specific `Scenario` is selected in here, the calculations will be performed for all the `Scenarios`.

In the case of **function deployments** `CalculationInput` consists of:

- ▶ One element (`DeplR`) that references a function deployment package
- ▶ One `TestData` element that contains one or more `TestSet` elements
- ▶ One or more `TestSet` elements that provide values for (function) variables that are used in the Functions
- ▶ One `RequestedFunctions` element that specifies which Functions need to be calculated. If functions have parameters, their values are also specified in here. Also, a function can reference one or more `TestSets`, which means the function will be calculated for those `TestSets`. If no `TestSet` is referenced, the function will be calculated for all `TestSets`.

In the case of **static or dynamic deployments** `CalculationOutput` consists of:

- ▶ Zero or more `Scenario` elements each containing `Calculation` elements that were performed for the `Scenario`
- ▶ Calculation elements that contain the results of the Features as requested in the corresponding Calculation elements in the input

In the case of **function deployments** `CalculationOutput` consists of:

- ▶ One or more `TestSet` elements each containing the results of the functions that were calculated with the data provided in the `TestSet`

❗ `CalculationOutput` is formatted with a white space and new lines and is terminated with a new line.

`CalculationData` and `Calculation` contain an xml representation of either a Deployment Environment Instances tree or a Financial Product Instances tree depending on the type of deployment (static versus dynamic). The Deployment Environment is the model of the client system as defined in Designer.


❗ A specific Deployment Environment or Financial Product in Designer is made up of Structure Elements. Each Structure Element is identified by a name. In Calculator Input and Output the Structure Elements are identified by the XML element who's name is that of the Structure Element.

The elements of a set are ordered, in that the set can be treated as if it were a list. A value can occur more than once in a set. When a set is mapped onto a structure link the actual set elements take their values through the structure mapping; however the order of the structure elements on which the set is mapped is not preserved. If your calculations require set elements to be in a certain order, then they need to be sorted using one of the sorting functions (i.e. sort, easysort). During calculations the order is preserved.

In order to clearly explain the structure of the XML, the remainder of this chapter includes the following:

- ▶ Characters and Decimal Floating Point in Calculator
- ▶ A complete list in tabular form of the used XML elements and attributes in Calculator Input and Output

- A description of the elements in Calculator Input and Output that are common to all types of deployments
- A description of elements in Calculator Input and Output that are specific to static and dynamic deployments
- An example and a detailed description of a Calculator input file
- A description of elements in Calculator Input and Output that are specific to function deployments
- Options in the Calculator input to control the contents of the Calculator output
- A description of the Error behaviour attributes
- Error Handling

 Examples of Calculator Input and Output can be found in Appendix A - Examples (see "[Appendix A - Examples for the Service Calculator](#)" on page [283](#)).

Characters and Floating Point in Calculator

This section describes:

- character set
- usage of decimal floating point

Character Set

All interfaces use the UTF8 character set. For an explanation of the xsd datatypes see the W3C XML Schema Datatypes.

- Boolean: Values are 'true' or 'false'.
- Integer: xsd:integer
- Real: xsd:decimal
- String: xsd:string
- Date: xsd:date
- DateTime: xsd:datetime

Usage of Decimal Floating Point

Calculator uses Decimal Floating Point: Arithmetic (and not Binary Floating-Point Arithmetic, like IEEE754). The precision of the floating point calculations within Calculator is as follows:

- By default the Decimal Floating Point numbers have a precision of 14 decimal digits. (Decimal Floating Point numbers are defined in decimal digits, so the number of bits used is not relevant). The maximum precision can be 16 digits that includes both integer part and decimal places. For example, 123456789.1234567 and 12345678901.12345 are both supported.

i Please note that different rounding methods are supported.

Some degree of performance degradation is to be expected when using higher precision calculations.

Please refer to the Calculator.Settings file for instructions on the precision and rounding method. The new settings will not come into effect until the Calculator is restarted.

- ▶ By default, decimal Floating Point numbers have an exponent that is greater or equal to -63, and less or equal to 63.
- ▶ By default, the largest positive number that can be represented in the Decimal Floating Point implementation is:
 $0.9999999999999999 * 10E^{63}$, or $9.999999999999999 * 10E^{62}$
- ▶ By default, the smallest positive number that can be represented in the Decimal Floating point implementation is:
 $0.1 * 10E^{-63}$, or $1 * 10E^{-64}$
- ▶ By default, the largest negative number that can be represented in the Decimal Floating point implementation is:
 $-0.1 * 10E^{-63}$, or $-1 * 10E^{-64}$
- ▶ By default, the smallest negative number that can be represented in the Decimal Floating point implementation is:
 $-0.9999999999999999 * 10E^{63}$, or $-9.999999999999999 * 10E^{62}$

XML Reference

This section describes XML references in various deployments.

XML elements in all types of Deployments

XML element	Description
CalculationInput	<p>This is the main element in a Calculator Request. It has an optional timer attribute that when set to the value 'true' the CalculationOutput contains some calculator internally measured times. These times can be used for tuning and performance analysis.</p> <p>Note that these times are not reliable if scenarios are used in the calculation. Scenarios are split into separate calculation tasks that will be executed concurrently in a multi CPU/Core environment. The times in the output will be the sum of the times of each calculation task.</p> <p>Time spent to process the CalculationInput XML queue-time Time the request spent in the Calculator queue calculation-time Time spent to calculate the requested calculations. Note that the external-function-time is part of this</p>

XML element	Description
	<p>time.</p> <p>external-function-time Time spent in external functions during the calculation.</p> <p>An example XML input:</p> <pre><CalculationInput timer="true"/></pre> <p>Example xml output:</p> <pre><CalculationOutput receive-time='0.02' queue-time='0.01' calculation-time='0.12' external-function-time='0.3'></pre>
DeplR	Identifies the deployment for which the calculation request is sent.
CalculationOutput	This is the main element in a Calculator Result.
Error	Contains a ProductXpress error message.
UserError	Contains a user defined error message.
MessageStack	Contains the message stack of a user defined error message.
ValidationMessage	Contains a user defined validation message.
ValidationError	Contains a ProductXpress or user defined error message raised in the validation.
Message	The message of a user defined error or a validation.
MessageParameters	Contains the parameters of a Message with their values.
Value	Time-dependent value of a Feature or Tuple Member. This element always is accompanied by the <i>date</i> attribute.
RuntimeConfiguration	In the output, this element identifies the runtime configuration that has been selected by DeplR in the input.

XML Elements in Static and Dynamic Deployments

XML element	Description
Calculation (in input)	<p>It either lists the data that is to be calculated or has the attribute calculate-all="true" in which case all calculable features will be calculated.</p> <p>It contains CalculationDates and optionally references</p>

XML element	Description
	to Scenarios. If no Scenarios are referenced it will calculate for all the Scenarios (including CalculationData).
Scenario (in input)	Contains data that overwrites data of CalculationData (CalculationData is regarded as the main scenario; it can be referenced in Calculation by using the reserved name 'main').
CalculationDates	The dates for which must be calculated.
Calculation (in output)	It contains the results that were requested in the corresponding Calculation in the input.
Scenario (in output)	Contains the results of the data as requested in the Calculation elements in the input that referenced this Scenario.
Aggregation	When Output Aggregation is requested, the Aggregation element contains the name of the aggregation, the status of the evaluation, the number of instances that succeeded and that failed.
ExcludedInstance	When Output Aggregation is requested and an instance has failed, the result for that failed instance contains the element ExlcudedInstance, which references this instance and holds an Error element explaining the failure.

XML Elements of Deployment Environment and Financial Product Instances

XML element	Deployment Environment/Financial Product
<any element>	The <any element> is the <i>navigation name</i> (see Navigation Names, Short Names and Mangled Names (on page 216)) of the component it represents. This can be a Structure Element, Link, Feature, Validation, Operation Parameter, or Tuple Member.
Reference	References a structure element instance in the input.
Features	Contains elements that represent features (attributes and operations) of the Structure Element instance to which the Features element belongs.
Validations	Contains elements that represent validations of the Structure Element instance to which the Validations element belongs.

XML Attributes in all types of Deployments

XML Attribute	Description	data format
act-date	Active Date attribute of a Deployment.	xsd:date
date	Date of a Value.	xsd:dateTime
dep-name	Used in a "DeplR" element. Name attribute of a Deployment.	xsd:string
nr	Error Number.	xsd:integer
timer	Specifies whether time information is returned.	xsd:boolean
ver-nr	When attribute of element 'DeplR': version number of the requested Deployment. When attribute of ValidationMessage or ValidationError: the version number of the validation. When attribute of UserError: the version number of the user defined error message.	xsd:string
ver-sel	Version Selector. Possible values are "specific", "last" or "active on date".	string
xsi:nil	When the value of a Feature (or the value of a Value of a time-dependent Feature) is a null value, this attribute must have value 'true', otherwise this attribute must not be present. A Feature with datatype string that has no value and no attribute xsi:nil represents an empty string.	xsi:nil

XML Attributes in Static and Dynamic Deployments

XML Attribute	Description	data format
aggregate	This appears on a Structure Element Link. When "true", it indicates the instances under the Structure Element Link will be sent later, as part of the output aggregation protocol. By default, it is "false".	xsd:boolean
value-source-option	Used in a Feature element. Name of the Value Source Option of a Feature.	xsd:NCName

XML Attribute	Description	data format
errors	The number of instances that failed during an output aggregation	xsd:decimal
ext-id	Optional identifier as used in the client system. The value will be reused in the CalculationOutput.	xsd:string
id	<p>In case of an instance: optional identifier of the (Deployment Environment Element) instance. Note that, while the value can be chosen by the client, the xsd:ID format implies that the value must be unique within the CalculationInput. The value will be reused in the CalculationOutput.</p> <p>In case of UserError: the unique ID of all versions of the user defined error message.</p> <p>In case of ValidationMessage or ValidationError: the unique id of all versions of the validation</p> <p>In case of Error: the ID of the ProductXpress error.</p> <p>In case this appears on a Structure Link, it identifies an output aggregation. Namely, the instances that will be sent as part of the output aggregation protocol need to use this ID to be placed under this link.</p>	xsd:ID
instance-ref	Identifies the instance that is referenced by the ValidationMessage or ValidationError.	xsd:IDREF
instances	The number of instances that succeeded in an output aggregation.	xsd:decimal
name	The name of a Calculation, UserError, ValidationMessage or Aggregation.	xsd:string
prerequisite	<p>The prerequisite flag occurs on a ValidationMessage.</p> <p>The prerequisite flag is NOT true, if and only if it is raised by a requested Validation or by the main-part of a Validation Group that is either a requested Validation Group, or a descendant of a requested Validation Group of which all its ancestor's prerequisite-parts are true. In all other cases the prerequisite flag is true. When</p>	"true"

XML Attribute	Description	data format
	the flag is false it is omitted.	
ref	Identifies the instance that is referenced via Reference.	xsd:IDREF
severity	The severity of a validation.	"1" "2" "3" "4"
status	The status of the output aggregation.	"ok" "error"
unit	The unit of a quantity value.	string

Common XML Elements

This section describes the XML elements that are common to static, dynamic and function deployments.

DeplR

The specification of which Deployment is to be used during a calculation is done in the Deployment Reference DeplR.

There are two categories of Deployments. One that contains a definition for the structure of the data and the calculations therein and another that only contains calculations (functions). The first is further split into so-called static deployments versus dynamic deployments. In a static deployment, the structure of the data (e.g. in CalculationData) is determined by the static deployment environment. In a dynamic deployment the structure is determined by the financial product that is deployed.

The elements within the data refer to Structure Elements in the referenced Static Deployment Environment or Financial Product. One of the root elements refers to the referenced Structure Element that acts as the entry point.

dep-name

In DeplR the attribute `dep-name` is always specified. This attribute contains the name of a certain Deployment. This name does not have to be the same as the name of the Deployment as defined in PX Designer. While deploying through Deplmp, this name can be chosen for a specific Deployment as known in PX Designer.

For Example:

```
<clc:DeplR dep-name="Calculator Example - Deployment"
ver-sel="last"/>
```

ver-sel

Note that the previous example using the Version Selector '`ver-sel = "last"`' selects the last version of the Deployment. However, there can be multiple Deployments with the same Deployment Name (thus same logical ID), which implies multiple versions of the same Deployment. ProductXpress Calculator supports three methods to retrieve the appropriate version of a Deployment. The attribute `ver-sel` of DeplR specifies which of these methods have to be used. This attribute is always specified.

ver-sel="specific": Select a Specific Version

In this method the attribute `ver-sel` has value "specific". The attribute Version Number (`ver-nr`) will be used to select the Deployment with name and version number. The version number is the version number of the desired Deployment Deliverable of which the Deployment is part.

-
- ❗ Although it might seem logical to use the version number of the Deployment, the version number of the Deployment Deliverable should be used instead. This is because you can make a new version of a Deployment Deliverable, that contains the same Deployment, but uses a new version of the Financial Product Deliverable.
-

If attribute `ver-nr` is not provided an error will occur. The version number can be found by running the command

```
DepImp -list -details
```

In the below example the version number is 0.7:

```
Configuration Id : 7fba8c3e-c940-40bf-a016-b73d2587ceb4
Name             : VUL
Version          : 0.7
Active Date      : 2007-01-09
Documentation     : User: calcUser
Company          : EDS
Date: 8/12/2006 2:26:20 PM
External Resource Roots:
Used type deployments:
```

ver-sel="last": Select the Last Version

In this method, the attribute `ver-sel` has value "last". The last version of the Deployment with name Deployment Name will be selected. The last version of a Deployment is the one with the highest version number.

ver-sel="active on date" : Select the Version that is Active on a Certain Date

In this method, the attribute `ver-sel` has value "active on date". To support this method, each version of the Deployments must have been registered by the user with a certain date. This date represents the date on which a certain version has been made active. The period in which a specific version of a Deployment is active starts on the date on which it has been made active and ends on the date on which another version of that Deployment has been made active.

The attribute `act-date` will be used to select the Deployment with name that is active on the active date.

If attribute `act-date` is not provided, an error will occur.

The active date can be found by running the command:

```
DepImp -list -details
```

In the example that follows. The active date is 2007-01-09.

```
Configuration Id : 7fba8c3e-c940-40bf-a016-b73d2587ceb4
Name             : VUL
Version          : 0.7
Active Date      : 2007-01-09
Documentation     : User: calcUser
Company          : EDS
Date: 8/12/2006 2:26:20 PM
External Resource Roots:
Used type deployments:
RuntimeConfiguration
```

Specifies which configuration was used for the calculation.

RuntimeConfiguration

Specifies which configuration was used for the calculation.

name

The name of the configuration that was used for the calculation.

version

The version of the configuration that was used for the calculation. If the `DeplR` element in the request selected a version active on a date or the last version for the calculation, the version that resulted from this selection is mentioned here.

id

The identifier of the configuration that was used for the calculation.

Error

An *Error* contains an error message of an error that occurred during the evaluation of a Feature or Validation. An *Error* has the attributes `nr` and `id` which are the error number and ID as defined in ProductXpress Calculator.

UserError

A *UserError* element contains an error message that is defined by the user in Designer. It is raised during the evaluation of a Feature or Validation. It has the attributes `id`, `ver-nr` and `name`. The ID is a unique identifier of the user error that is assigned by ProductXpress. The `ver-nr` is the version number of the user error and the name the name as is defined by the user.

MessageStack

This element contains the stack of errors that is built up, starting at the location where the user error is raised until the final place (Feature or Validation) that is reported to the calling application.

ValidationMessage

This element contains the message of a validation that has failed.

ValidationError

This element contains a *UserError*, in case a user defined error is raised, or an *Error* in case a ProductXpress error is raised.

Message

The message of a Validation or *UserError*. The message is defined by the user in Designer.

MessageParameters

This contains the parameters of a Message together with their values. The parameter values also appear inside the Message (in case the user has referenced the parameters inside the message).

XML Elements for Static and Dynamic Deployments

This section describes the XML elements for static and dynamic deployments.

Data Tree

`CalculationData` and `Calculation` elements contain an XML representation of an instantiation of either a Deployment Environment model or a Financial Product model. In this section, we will refer to these models as the Calculation Data Model. The XML representation of this Model is referred to as the XML Data Tree.

The Calculation Data Model contains a number of Structure Element trees, each starting in a so-called Root Element.

The Root Elements of the XML Data Tree correspond with the Root Elements in the Calculation Data Model. The descendant elements of the Root Elements in this Data Tree correspond with Links, linked Structure Elements, Features and Validations of Structure Elements in the Calculation Data Model.

ref

Reference Links to instances (amongst which other Root Elements) in the Calculation Data Model are represented as Reference Elements in the XML Data Tree. The `ref` attribute identifies the instance of the linked instance.

Requested Calculations Specifics

The `Calculation` element should only contain an XML Data Tree if it has not set attribute `calculate-all` to `true`. If `calculate-all` is set to `true`, all calculable Features of instances in `CalculationData` (and selected `Scenarios`), will be calculated by the Calculator.

If the `Calculation` element does contain an XML Data Tree, all Structure Element instances in this tree must already have been defined in the `CalculationData` section. The instances in the `Calculation` element reference instances that are specified in the `CalculationData` and `Scenario` sections and specify which features should be calculated for these instances.

Structure Element Instances and References

An instance of a Structure Element has the following XML Element and attributes:

- ▶ XML Element name: identifies the Structure Element.
- ▶ Attribute **id**: identifies a specific instance of a Structure Element inside the `CalculationData` or `Scenario` section. Its value is user-defined, but must be unique within the Request to the Calculator.
- ▶ Attribute **ext-id**: Optional identifier as used in the client system.
- ▶ Attribute **ref**: references the instance within `CalculationData` or `Scenario` section. Its value is the value of the **ID** attribute of the instance to which it refers. Note that the instance to which is referred must have an attribute **id**.

Feature

A *Feature* can be used:

- ▶ In the `CalculationData` section, it contains the values of a Feature.
When the Feature is time-dependent each value on a date is represented by a `Value` child element.
When the Feature has multiple Value Source Options, it contains a Value Source Option attribute that specifies the value source option.
For empty/null values the Feature is either omitted or the Feature has the attribute

`xsi:nil="true".`

In case a Feature (or any part of its value definition) is specified without the attribute `xsi:nil` and no value, its value is undefined, except in the case its datatype is String or Set of <some value definition>. In this case, the value of the Feature is an empty string or an empty set respectively.

- ▶ In the input `Calculation` section it specifies that the Feature is to be included in the Result.
- ▶ In the output `Calculation` section, it contains the calculated value of the requested Feature, or it contains an `Error` child element.
For empty/null values the attribute `xsi:nil="true".`
In case a Feature (or any part of its value definition) is of type String or Set of <some value definition> and it does not contain a value nor the `xsi:nil` attribute, the value is an empty string or empty set respectively.

Value Source Option

A *Value Source Option* contains the **name** of the Value Source Option of the feature it belongs to.

It indicates that the Value Source Option has been selected.

It is specified in the `CalculationData` section.

It does not need to be specified:

- ▶ when the Feature has only one Value Source Option (this value source option will be selected).
- ▶ when the Feature has multiple Value Source Options, and a *default* Value Source Option (the default value source option will be selected). (This is only applicable in static Deployments).

Value

A *Value* contains a value of a time-dependent Feature on a certain date. It can have either been specified by the third party application - this applies only to attributes - or calculated by ProductXpress Calculator. A *Value* has a mandatory *date* attribute. In case the *Value* was provided by a third party, it is the date from which onwards the *Value* is valid for the time-dependent Feature. In case the *Value* was calculated, it is the Calculation Date of the (result) value of the time-dependent Feature.

When a Parameter of an Operation references a Request Parameter in the input, the Result of the Operation will have one or more *Value* elements. Each Value element contains an attribute that represents the request parameter with a value. The content of the *Value* element is the result of the operation for that request parameter value.

When more than one Parameter of the same Operation references a request parameter, the Result of the Operation will contain multiple *Value* elements, each for every combination of values of the referenced request parameters.

When multiple operation parameters of the same Operation reference the same request parameter, the request parameter will appear once as attribute of the *Value* element. Its value represents the value of those operation parameters that referenced that (same) request parameter.

Validation

A *Validation* can be used:

- ▶ In the input `Calculation` section it specifies that the Validation is to be evaluated.

- In the output `Calculation` section. When the validation has failed it contains a message. The message can either be a validation message, user defined error message or a ProductXpress error message. When the validation has succeeded, it does not contain anything.

ValidationMessage

The `ValidationMessage` element contains the message explaining why the validation failed, as defined by the user.

UserError

The `UserError` element contains a message that is raised while a validation or feature is evaluated. The message is defined by the user.

Translation of Validation/User Error Messages to Other Languages

In order to translate the message of a validation or user error to other languages, the integrating party must make a mapping between the message and the translation. The validation/error messages must therefore be uniquely identifiable. Both error messages and validation messages are uniquely identified by an ID and version number and/or by a name. The ID and version number are generated by ProductXpress. The name is defined by the user in Designer. If the third party decides to use just the name to uniquely identify a message (and its versions), it is responsible for creating unique names for all messages that are deployed to the runtime.

Message parameters are uniquely identifiable by the identifier of the message it belongs to and their name.

Error

The `Error` element contains a ProductXpress message in case exceptions have occurred during the evaluation of a validation or feature.

CalculationDates

The `CalculationDates` section contains one or more `At` elements each specifying a datetime on which the requested calculations are performed. It can also contain a range of datetimes that is specified by the `Range` element.

The time part of each datetime can be omitted. In this case the time "00:00:00" is used.

The elements contained by `CalculationDates` are described in the next sections. See [Appendix B - Specification of Calculator Input-Output](#) (see "[Appendix B - Specification of Calculator Input-Output XML](#)" on page [285](#)) for the possible combinations of these elements.

All these elements (except `Range`) contain either a constant value or a reference to a Feature. In the latter case, the element gets the value of the Feature (either calculated or retrieved from `<CalculationData>`).

Range

The `Range` element specifies a range of datetimes by using the `From`, `To`, `NumberOfSteps` and `StepSize` elements. These elements are described in the following sections.

The `Range` element has an attribute **LastDayOfMonth**, which indicates whether or not the last day of each month in the range of dates is to be used. The default is **false**.

At

An `At` element contains a date that specifies the datetime for which the calculations have to be performed. Note that there can be multiple `At` elements.

From

The `From` element contains a datetime that specifies the first datetime ('from date') of a range of dates.

The `From` element has an attribute *inclusive*, which specifies whether or not the 'from date' itself is part of the interval. The default for this attribute is 'true'.

To

The `To` element contains a datetime that specifies the last datetime ('to date') of a range of datetimes.

The `To` element has an attribute *inclusive*, which specifies whether or not the 'to date' itself is part of the interval. The default for this attribute is 'true'.

StepSize

The `StepSize` element contains a number that specifies what the step size is between consecutive datetimes in a range of datetimes. The `StepSize` element has an attribute *unit*, which specifies the unit of the step size.

NumberOfSteps

The `NumberOfSteps` element contains a number that specifies the number of times a step is to be taken, starting either at `From` and going 'upwards' (towards infinity) or at `To` and going 'downwards' (towards minus infinity). The size of the steps are determined by `StepSize`.

Exceptions

When determining the range of dates that are specified by, say a `From` and a `To` element, it may be the case that certain dates within the range do not exist. This can happen when the day of the month is 29 or 30 or 31: e.g., *31 February*. In these cases, the last possible day for such a month will be chosen. This is not the same as the *LastDayOfMonth* being 'true'.

Result order

The Calculator will perform the requested calculations for all the calculation datetimes resulting from the specified range.

Each calculated feature in the result will contain a list of values, ordered from the oldest to the newest calculation datetime.

Calculator Input Example

The following is an example of a calculation input for a static or dynamic deployment. In total it is a complete request, however it is broken up into sections to allow for explanation of the syntax. For more examples (including Calculator output) refer to the guide *Example Code for the ProductXpress Service Calculator*.

In this example, a simple structure of one Root Element contains the data required for one calculation. In the `Calculation` element *first years* the features 'Premium' and 'Main Coverage Premium' are requested to be calculated.

The first section is the header of the input file.

```
<?xml version="1.0" encoding="UTF-8"?>
<clc:CalculationInput
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:clc="http://www.solcorp.com/ns/ProductXpress/
    CalculationInputOutput/CalculatorElement"
  xmlns="http://www.my.name.space.com/ProductXpress/Calculator">
```

```
request-id="Example from Specs">
```

This is followed by the deployment reference (`Dep1R`) and the date information. In the case the deployment name is 'SOLCORP VUL Product in INGENIUM' and the version of the deployment should be the one active on date 2000-01-13.

```
<clc:Dep1R dep-name="SOLCORP VUL Product in INGENIUM"
act-date="2000-01-13" ver-sel="active on date"/>
```

The next section is the Calculation Data section. It contains a number of Structure Elements trees, each one beginning with the Root Element. In this case the Root Element is *SOLCORPVULProduct*. The entered features of this product are *PolicyIssueDate*, *MaturityDate* and *FaceAmount*. The Root Element has a child element called *MainModule* which in turn has multiple *MainCoverage* children. In this example there are two instances of *MainCoverage*, each containing an entered feature *Premium* that has a quantity value and is time dependent.

```
<clc:CalculationData>
  <SOLCORPVULProduct id="ID1234" ext-id="1234">
    <Features>
      <PolicyIssueDate>2002-01-01</PolicyIssueDate>
      <MaturityDate>2050-12-31</MaturityDate>
      <FaceAmount>150000</FaceAmount>
    </Features>
    <Links>
      <LinkToMainModule>
        <MainModule id="ID5678" ext-id="5678">
          <Links>
            <LinkToMainCoverage>
              <MainCoverage id="ID9012"
ext-id="2">
                <Features>
                  <Premium>
                    <Value
date="2000-03-
1" unit="euro">100</Value>
                  </Premium>
                </Features>
              </MainCoverage>
            <MainCoverage id="ID0123"
ext-id="4">
                <Features>
                  <Premium>
                    <Value
date="2002-03-01">200</Value>
                  </Premium>
                </Features>
              </MainCoverage>
            </LinkToMainCoverage>
          </Links>
        </MainModule>
      </LinkToMainModule>
    </Links>
  </SOLCORPVULProduct>
</clc:CalculationData>
```

The next section is the Calculation section that contains the requested features. The Calculation section is labeled *first years*. As part of this Calculation there will be calculations made for the

TotalPremium of the Root Element *SOLCORPVULProduct* as well as the *MainCoveragePremium* for each instance of *MainCoverage*.

Note how the structure in the Calculation section is identical to the structure in the CalculationData section above. The instance of the structure element to be recognized is specified as <Element name> ref="id". In this case, the Element name is *SOLCORPVULProduct* and the ref is *ID1234*. This is the instance of the product specified in the CalculationData section above.

```
<clc:Calculation name="first years">
  <SOLCORPVULProduct ref="ID1234">
    <Features>
      <TotalPremium/>
    </Features>
    <Links>
      <LinkToMainModule>
        <MainModule ref="ID5678">
          <Links>
            <LinkToMainCoverage>
              <MainCoverage>
                <!--
                  omitted ref means all instances of all links to
                  MainCoverage
                -->
                <Features>
                  <MainCoveragePremium/>
                </Features>
              </MainCoverage>
            </LinkToMainCoverage>
          </Links>
        </MainModule>
      </LinkToMainModule>
    </Links>
  </SOLCORPVULProduct>
</clc:Calculation>
```

Within the Calculation section there is the Calculation Dates. In this example it is set as a range from the *PolicyIssueDate* to the *MaturityDate* of the instance of *SOLCORPVULProduct* with id *ID1234* (i.e. the instance specified in the CalculationData section above). By looking at the referenced StructureElement we can see that the *PolicyIssueDate* is 2002-01-01 and the *MaturityDate* is 2050-12-31.

```
<clc:CalculationDates>
  <clc:Range>
    <clc:From>
      <SOLCORPVULProduct ref="ID1234">
        <clc:Features>
          <PolicyIssueDate/>
        </clc:Features>
      </SOLCORPVULProduct>
    </clc:From>
    <clc:To>
      <SOLCORPVULProduct ref="ID1234">
        <Features>
          <MaturityDate/>
        </Features>
      </SOLCORPVULProduct>
    </clc:To>
  </clc:Range>
</clc:CalculationDates>
```



```

    </Features>
  </SOLCORPVULProduct>
</clc:To>
</clc:Range>
</clc:CalculationDates>

```

And finally we close our root xml element with:

```
</clc:CalculationInput>
```

Multiple Requested Calculations

It is possible to repeat the `Calculation` element that groups together requested calculations and the `CalculationDates` on which they need to be calculated. The `Calculation` element optionally has a name that will also be returned in the output.

This enables, among other things, the calculation of different groups of calculations on different dates.

When multiple `Calculation` elements are specified in the input, they will also appear in the output with the results for the requested calculations and calculation dates as specified in the `Calculation` element in the input.

Example Input:

```

<clc:Calculation name="first years">
  <SOLCORPVULProduct ref="ID1234">
    <Features>
      <TotalPremium/>
    </Features>
    <Links>
      <LinkToMainCoverage>
        <MainCoverage>
          <Features>
            <MainCoveragePremium/>
          </Features>
        </MainCoverage>
      </LinkToMainCoverage>
    </Links>
  </SOLCORPVULProduct>
  <clc:CalculationDates>
    <clc:Range>
      <clc:From>2001-02-01</clc:From>
      <clc:To inclusive="false">2005-02-01</clc:To>
      <clc:StepSize unit="year">1</clc:StepSize>
    </clc:Range>
  </clc:CalculationDates>
</clc:Calculation>
<clc:Calculation name="last years">
  <SOLCORPVULProduct ref="ID1234">
    <Features>
      <TotalPremium/>
    </Features>
  </SOLCORPVULProduct>
  <clc:CalculationDates>
    <clc:At>2005-02-01</clc:At>
    <clc:At>2010-02-01</clc:At>
    <clc:At>2020-02-01</clc:At>
  </clc:CalculationDates>
</clc:Calculation>

```

Scenarios

In cases where variations in the `CalculationData` are needed with which calculations need to be performed, one or more `Scenario` elements can be defined. A `Scenario` element has a name and contains the same structure as `CalculationData`. Within a `Scenario` new instances and new attribute values can be defined in relation to the `CalculationData` and existing attribute values in `CalculationData` can be overridden. It is also possible to override references to instances; e.g., the reference to the *person* who plays the role *insured* in the `CalculationData` can be a different *person* within a `Scenario`.

The `CalculationData` itself is regarded as a scenario as well, with a reserved name: *main*. Evidently, this name cannot be used in user defined `Scenario` elements.

In the `Calculation` section one or more `Scenario` elements can be selected using one or more `InputData` elements. An `InputData` element refers via its *scenario* attribute to the name of a `Scenario`. The base scenario is selected by referring to the reserved name *main* in `InputData`. If no `Scenario` is selected, all `Scenario` elements (including the main scenario) will be calculated for.

When a calculation is performed for a certain `Scenario`, the input data used for the calculation is the main input (`Calculation Data`) overridden and/or merged with the `Scenario` data.

Example:

```
<clc:CalculationData>
  <Prod id="ID1234" ext-id="1234">
    <Features>
      <Policy_Issue_Date>2002-01-01</Policy_Issue_Date>
      <Interest_Rate>3</Interest_Rate>
    </Features>
  </Prod>
</clc:CalculationData>
<clc:Scenario name="Interest@5">
  <Prod ref="ID1234">
    <Features>
      <Interest_Rate>5</Interest_Rate>
    </Features>
  </Prod>
</clc:Scenario>
<clc:Scenario name="Interest@12">
  <Prod ref="ID1234">
    <Features>
      <Interest_Rate>12</Interest_Rate>
    </Features>
  </Prod>
</clc:Scenario>
<clc:Calculation>
  <Prod ref="ID1234">
    <Features>
      <premium/>
    </Features>
  </Prod>
  <clc:InputData Scenario="main"/>
  <clc:InputData Scenario="Interest@5"/>
  <clc:InputData Scenario="Interest@12"/>
</clc:Calculation>
```

In this example we have two scenarios (+ the main scenario). In both scenarios a different value for the *Interest_Rate* is defined. In the *Calculation* section both scenarios and the main scenario is selected. In short all scenarios are selected. Note that this can also be achieved by not specifying any *InputData*.

The *premium* will thus be calculated with *Interest_Rate* 3, 5 and 12.

XML Elements for Function Deployments

An input request for a Function deployment consists of the root element *CalculationInput*, just like for static and dynamic deployments. *CalculationInput* consists of three main sections: *DeplR*, *TestData* and *RequestedFunctions*.

An output result for a Function Deployment from Calculator consists of the root element *CalculationOutput* (as with static and dynamic deployments). *CalculationOutput* consists of either an *Error* element (see the section *Error* (on page [204](#))), in case an error occurred, or zero or more *TestSet* elements.

XML Elements in Function Deployments

XML element	Description
<any element>	The <any element> is the <i>navigation name</i> (see Navigation Names, Short Names and Mangled Names (on page 216)) of the component it represents. This can be a Variable, Function Variable, Function, Function Parameter or Tuple Member depending on where it appears in the XML.
TestData	Contains one or more TestSet elements.
TestSet	Contains values for Variables (<any element>) and Function Variables (<any element>).
RequestedFunctions	Contains the Functions (<any element>) that need to be calculated. The Function elements contain one or more Call elements.
Call	Identifies a Call to the Function. If the Function has Parameters the Call contains values for these Parameters (<any element>).
CalculationDate	Contains the value for the reserved variable <i>Calculation Date</i> .
Self	Contains the value for the reserved variable <i>Self</i> .
FunctionName	The name of the Function that is part of the deployment package and is selected as the value of a Function Variable.
ExternalFunctionName	The name of the External Function that is selected as the value of a Function Variable.
Identity	Contains the value for the <i>identity</i> of a Tuple.

XML element	Description
SetElement	Contains the value for an element in a set.

TestData

The `TestData` element contains **one** or more `TestSet` elements.

TestSet

A `TestSet` element in the input contains values for Variables and Function Variables (<any element>). It has an `ID` attribute with which it can be identified in the output.

Variables and Function Variables are represented by XML elements. The names of the XML elements are the names of the Variables/Function Variables. The Variables contain zero or more `Value` elements. Only when a Variable is *time dependent* it can contain more than one `Value` element, with each a *date* attribute.

A Function Variable contains zero or one `Value` element.

A value of a Variable or Function Variable can be permuted by specifying more than one value. See Appendix B of this document and section *Entering Values* in Designer Online Help for the syntax of permutations.

Each permutation will result in a separate `TestSet` element in the output. Note that when the value of a Function Variable is the name of a Function or External Function, it cannot be permuted.

Variables and Function Variables have an `xsi:nil` attribute with which can be specified that the value is *null* or empty. By default this attribute is false.

-
- ❗ The reserved Variables *Self* and *CalculationDate* are also considered Variables with the difference that they are represented by a predefined XML element.

When a string value should not be permuted (i.e., the semi-colon not to be regarded as a permutation separator) it can be indicated by setting the *permute-stringValue* attribute in the `Value` element to *false* (default it is *true*).

A `TestSet` element in the output contains values of Functions. These values are the results of the Functions that are requested in the input in the `RequestedFunctions` element and are calculated with the data provided in the `TestSet`.

A `TestSet` element in the output references a `TestSet` in the input via the *ref* attribute. If a `TestSet` in the input contains permutations, each `TestSet` in the output that references that `TestSet` will contain one of the permuted values of each Variable that was permuted.

RequestedFunctions

The `RequestFunctions` element contains *Function* elements (<any element>). A Function is represented by an XML element with the name of the Function. This element contains one or more `Call` elements.

Call

A `Call` element represents a call to a Function and is identified with an `ID` attribute. When the Function has Parameters, the `Call` contains values for these Parameters (<any element>). A

Parameter is represented by an XML element whose name is the name of the Parameter. A Parameter contains a value.

Elements that contain values

The following elements can contain a value:

Variable, Function Variable, Function Parameter, Tuple Member, SetElement and Value

All of these elements can have the following attributes: `xsi:nil`, `permute-stringValue` and `unit`. These attributes will be explained later in further detail.

Time Dependent Values

When a Variable, Tuple Member or Set Element is time dependent, multiple values (over time) are defined through multiple child `Value` elements that each represents a value on a certain date. Hence, the `Value` element has a mandatory `date` attribute with which is specified that the value is valid from this date onwards (until the earliest date (after this date) of a sibling `Value` element).

Null Values

All of the elements mentioned in the beginning of this section (including `Value`) have an `xsi:nil` attribute with which can be specified that the value is *null* or empty. By default this attribute is false.

String Permutations

For string values, it can be useful to disable permutations when the semi-colon in the value should not be regarded as a permutation separator (which it normally is). This is achieved by setting the `permute-stringValue` attribute to `false`. By default this attribute is `true`.

Quantity Values

When the value is a quantity, the unit of the value can be specified with the `unit` attribute.

Table Values

When the value is a table, the name of a Table can be specified as the value. Note that the Table must be part of the value domain of the value definition of the element for which the value is specified.

CalculationDate

The element `CalculationDate` represents the reserved variable *Calculation Date*. It contains a date (or a permutation of dates).

Self

The element `Self` represents the reserved variable *Self*. It contains a string (or a permutation of strings).

ExternalFunctionName

The element `ExternalFunctionName` contains the name of an External Function (that is known to Calculator via *ExternalFunctions.xml*). It is used as the value of a Function Variable.

FunctionName

The element `FunctionName` contains the name of a Function that is part of the deployment package for which the request applies. It is used as the value of a Function Variable.

Identity

The `Identity` element holds the identity of a tuple. The value is a string.

SetElement

The `SetElement` element contains the value of one of the elements in a Set. When the Set is time dependent, the attributes *start-date* and *end-date* are used to indicate the validity of the `SetElement`.

Navigation Names, Short Names and Mangled Names

Some components, e.g. Structure Elements or Features, in the Calculator Input and Output XML are represented by `<any element>`, which stands for any XML element. XML elements must follow certain naming rules; please refer to <http://www.w3.org/TR/2004/REC-xml11-20040204> (<http://www.w3.org/tr/2004/rec-xml11-20040204>) for these rules.

Components that are modeled in Designer have very little to no naming rules. Therefore, their names cannot be used as is in XML elements. In order to represent components in Calculator Input and Output, their so-called *navigation names* are used instead of their regular names within XML elements.

The *navigation name* of a component is derived from its *name* and its *short name*. The *short name* is an alternative name of a component that the user can specify in Designer and that stands for the XML element name of the component. A short name must therefore comply with the naming rules of an XML element.

The derivation of the navigation name is as follows:

- ▶ If a short name has been defined, the navigation name is the short name; otherwise the navigation name is the *Mangled Name* of the component.

The *Mangled Name* of a component is defined as follows:

- ▶ any character in the regular name of the component that is not allowed in an XML element name is replaced by `'_'` (an underscore)
- ▶ if the regular name of a component starts with a character that is allowed in an XML element name, but not as a start character, an underscore is prefixed
- ▶ If the component is a Link and there is no name specified for the Link (which is allowed), then the navigation name of the Link is the navigation name of the Structure Element that the Link refers to.

Note that only components that are used in Calculator Input and Output need a navigation name. Sections XML Elements of Deployment Environment and Financial Product Instances (on page [199](#)) and XML Elements in Function Deployments (see "[XML Elements for Function Deployments](#)" on page [213](#)) list these components.

Output Attributes

Output attributes are specified on the `CalculationInput` element, which is the root element of the Calculator input. They control the content of the Calculator Output.

Attribute	Description
timer	<p>When the optional timer attribute is set to the value 'true', the CalculationOutput contains some Calculator internally measured times. These times can be used for tuning and performance analysis.</p> <p>Note that these times are not reliable if scenarios are used in the calculation. Scenarios are split into separate calculation tasks that will be executed concurrently in a multi CPU/Core environment. The times in the output will be the sum of the times of each calculation task.</p> <p>Time spent to process the CalculationInput XML:</p> <p>queue-time Time the request spent in the Calculator queue</p> <p>calculation-time Time spent to calculate the requested calculations. Note that the external-function-time is part of this time.</p> <p>external-function-time Time spent in external functions during the calculation.</p> <p>An example XML input:</p> <pre><CalculationInput timer="true"/></pre> <p>Example xml output:</p> <pre><CalculationOutput receive-time='0.02' queue-time='0.01' calculation-time='0.12' external-function-time='0.3'></pre>
request-id	The value of this attribute can be any character string. It will be added to the output, as attribute of the CalculatorOutput element.
input-in-output	When this attribute has the value 'true', the content of the CalculationData and Scenario elements in the input will be repeated in the in the CalculatorInput element in the Calculator output.

Error behaviour attributes

In the `CalculationInput` element, which is the root element of the Calculator Input, each type of error is represented by an optional XML attribute. The defaults for error behaviour are taken from the corresponding section. The specific types of error behaviour are:

Attribute	Description
break-on-error	If break-on-error is set to true, the calculator will stop processing the current request on the first

Attribute	Description
(default: false)	error that occurs. When it is set to false, the Calculator will include the error-message in the output-data and continue with the next calculation. Note that this only applies to errors occurring during the calculation/processing of individual features, not to errors that occur while handling the request as a whole.
break-on-severity (default: 4)	When a validation request results in a message that has a severity that is equal or higher than <i>break-on-severity</i> , Calculator will not perform any of the requested calculations in the request where the validation was defined (or in other words, none of the calculation results appear in the output).
ignore-input-structure-errors (default: false)	<p>If this option is set to true, the input parser will ignore errors in the <code>CalculationData</code> and <code>Scenario</code> sections of the xml-input. This allows for deployment environments that are not fully compatible, to share the same integration software.</p> <p>The following 'errors' are ignored:</p> <p>Links present in the input that do not exist in the FPM and all instances connected to it are ignored.</p> <p>Instances that have a type that is not a known sub-type of the target-type of the link they are connected to, are handled as if they are of the target-type. If the target-type is an abstract type, the instance is ignored completely.</p> <p>Features that are present in the input and which are not defined in the FPM are ignored.</p>
ignore-output-structure-errors (default: false)	This option works the same as the ignore-input-structure-errors option described in the previous section, only for the data specified in the <code>Calculation</code> section.
ignore-input-mapping-errors (default: false)	When this option is active, errors that occur during resolving of input-mappings are ignored, i.e. the variable for which the mapping was requested will get a null value.
ignore-output-mapping-errors (default: false)	When this option is active, errors that occur during resolving of output-mappings are ignored, i.e. the feature for which the mapping was requested will get a null value.
check-link-multiplicity (default: false)	When this option is active, it is checked to determine whether the number of instances under a link violates the multiplicity of that link.

Attribute	Description
check-input-values (default: false)	When this option is active, ValueDomain checking is done on the input values of features.
check-output-values (default: false)	When this option is active, ValueDomain checking is done on the output values of features.

These options can also be set globally in *Calculator.Settings* (standalone Calculator) or in *DSCalc.Settings* (Calculator in Designer). The local options (inside *CalculationInput*) overrule the global options.

Error Handling

Possible errors:

- ▶ XML that is not well-formed (does not conform to XML syntax). When using Soap or COM, an exception will be thrown. When using TCP, the connection will be shut down.
- ▶ Error during the transformation of the input Deployment Environment instances to the internal ProductXpress structures. An error in CalculationOutput will be returned. An example is XML input that does not conform to input specification.
- ▶ Error during calculation of a Feature. The Feature will contain an error element in the output. An example is a missing input attribute.

Error Conversion

- ▶ The error numbers in the Result can be converted to error numbers as desired by the client.
- ▶ The Connector will lookup the conversion for the error in the ErrorConversion.Settings file.
- ▶ An ErrorConversion.Settings file optionally can have an element default, which specifies the default error conversion value. If this element is not used, only the specified errors will be converted.
- ▶ A client can enhance the ErrorConversion.Settings file, but the first 100 error numbers are reserved for standard messages. Usage of numbers in this range may be overruled when a new release is installed. This range will be defined by the Ingenium and ProductXpress teams jointly.
- ▶ If the client does not make use of the default value element, all error numbers from 1000 upwards are also reserved for Calculator errors.
- ▶ The converted Error element contains the desired error number and the full error message of Calculator.
- ▶ Further parameterization of the errors, or displaying a different error message to the user, is the responsibility of the client.

To configure the numeric ids of errors reported through the Calculator Result, a file named **ErrorConversion.Settings**, has to be created in the <install>\Etc directory.

Each entry in this file should have the following elements and attributes:

Element	Attribute
ErrorConversion This is the root element. It contains zero or more <i>Convert</i> elements.	None
convert Each <i>convert</i> element specifies the conversion for a specific error number	the first is either from or id , and the second is to
	from The Calculator error number
	id The Calculator error identifier. This has the format Xxx.YYY. The names are determined by the error message files in the <install>\Etc\Messages directory of the Calculator installation. The first part is the file name minus the msg extension, the second part is the value of the id element in that file
default The optional <i>default</i> element specifies the desired error number for all other errors.	to The desired error number
	value

Example :

```
<?xml version="1.0"?>
<ErrorConversion>
  <convert id="ResMan.CALC_ERROR" to="1"/>
  <convert from="1124" to="1"/>
  <convert from="2231" to="1"/>
  <default value="2"/>
</ErrorConversion>
```

In this example error numbers 1124, 2231 and the error with id=' CALC_ERROR' in ResMan.msg are all converted to 1, other errors are converted to 2.

-
- ❗ When more than one Calculator error is converted to the same error number, their error messages will generally differ, so that in the Calculator output the same error number will not always have the same error message.

As the error conversion is not client specific, all clients of a specific Calculator installation will get the same conversion for their errors.

Calculator Input and Output Syntax Specification

The RELAX NG compact notation (<http://relaxng.org/compact-20021121.html> (<http://relaxng.org/compact-20021121.html>)) is used to specify the Calculator Input and Output syntax.

Static and Dynamic Deployments Definitions

This section describes the calculator input and output XML used in both static and dynamic deployments.

CalculationInput

```

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

namespace dp = "http://www.example.org/myDeployment"
# the deployment namespace (dp in this case) URI is user defined
# as each deployment has its own schema

include "structureCommon.rnc"
include "calculatorCommon.rnc"
include "deploymentReference.rnc"

start = calculationInput

calculationInput = element clc:CalculationInput
{
    calculationInputAttributes    # see calculatorCommon.rnc
    , deploymentReference         # see deploymentReference.rnc
    , calculationData
    , scenario *
    , calculation *
}

calculationData = element clc:CalculationData
{
    structureElementInput *
}

scenario = element clc:Scenario
{
    attribute name {xsd:string}
    , structureElementInput *
}

calculation = element clc:Calculation
{
    attribute name { xsd:string } ?
    , attribute calculate-all { xsd:boolean } ?
    , attribute validate-all { xsd:boolean } ?
    # when either validate-all or calculate-all or both are
    # true, clc:Calculation is empty
    ,
    (
        structureElementRequest *
        & requestedScenario *
        & calculationDates ?
        & calculationParameters ?
    )
}

```

```

requestedScenario = element clc:InputData
{
    attribute scenario { xsd:string }
}

calculationDates = element clc:CalculationDates
{
    (at | dateRange) *
}

calculationParameters = element clc:CalculationParameters
{
    calculationParameter +
}

calculationParameter = element clc:Parameter
{
    attribute name {xsd:string - ("unit" | "date" | "xsi:nil")}
    # the name "date" is reserved for calcdates
    # the name "unit" is reserved for# quantityValues in output
    # xsi:nil is reserved for nilValue in output
    , attribute type { calculationParameterType }
    , simpleTypeMultiValue      # see calculatorCommon.rnc
}

calculationParameterType =
(
    "xsd:boolean" | "xsd:date" | "xsd:dateTime" | "xsd:integer" |
    "xsd:float" | "xsd:string"
)

at = element clc:At
{
    dateOrDateTime
    | structureElementRequest
}

dateRange = element clc:Range
{
    lastDayOfMonth ?
    ,
    (
        (
            from
            ,
            (
                (
                    to
                    , stepSize
                )
                |
                (
                    stepSize
                    , numberOfSteps
                )
            )
        )
    )
}

```

```

        )
        |
        (
            to
            , stepSize
            , numberOfSteps
        )
    )
}

lastDayOfMonth =
(
    attribute last-dom { xsd:boolean }
)

from = element clc:From
{
    attribute inclusive { xsd:boolean } ?
    ,
    (
        dateOrDateTime
        | structureElementRequest
    )
}

to = element clc:To
{
    attribute inclusive { xsd:boolean } ?
    ,
    (
        dateOrDateTime
        | structureElementRequest
    )
}

stepSize = element clc:StepSize
{
    attribute unit { "second" | "minute" | "hour" | "day" |
"week" | "month" | "year" }
    ,
    (
        xsd:integer
        | structureElementRequest
    )
}

numberOfSteps = element clc:NumberOfSteps
{
    (
        xsd:integer
        | structureElementRequest
    )
}

```

calculationOutput

```

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

namespace dp = "http://www.example.org/myDeployment"
namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"

include "structureCommon.rnc"
include "calculatorCommon.rnc"
include "runtimeConfiguration.rnc"

start = calculationOutput

calculationOutput = element clc:CalculationOutput
{
  calculationOutputAttributes # see calculatorCommon.rnc
  ,
  (
    error # see structureCommon.rnc
    |
    (
      runtimeConfiguration
      # see runtimeConfiguration.rnc
    ,
    calculationInput ? # present if input-in-output=true
    ,
    (
      calculationResults
      | invalidInput
      # in case of break-on-error
      |
      (
        invalidInput
        , calculationResults
      )
    )
    )
  )
}

calculationInput = element clc:CalculationInput
{
  calculationData      # see calculation input
  ,
  scenario *           # see calculation input
}

invalidInput = element clc:InvalidInput
# when check-input-value = 'true'
{
  error +
  # If a feature has an invalid value, the Error specifies the
  # value, the instance id, the feature name and
  # the value definition name. If such feature is in a Scenario # the
  Error also mentions the Scenario name.

```

```

# If an operation argument has an invalid value, the Error
# specifies the value, the Operation Call id,
# the operation parameter name, the value definition name and
# (if specified) the Calculation name. If the operation
# parameter has a tuple member, the tuple member name is also
# mentioned.
}

```

```

calculationResults =
(
  simpleResults
  | scenarioResults
  |
  (
    simpleResults
    , aggregationInfo
  )
  | aggregationInfo
# usually the case for aggregated resources as they do not
# request calculations
)

```

```

simpleResults =
(
  calculation +
)

```

```

scenarioResults =
(
  scenario +
)

```

```

aggregationInfo = element clc:Aggregation
{
  attribute name          { xsd:string }
  , attribute status      { "ok" | "error" }
  , attribute instances   { xsd:decimal }
# number of successful instances included in the Aggregation
  , attribute errors      { xsd:decimal }
# number of failed instances discarded from the Aggregation
  , excludedInstance ?
# only appears in the output of an excluded request
}

```

```

excludedInstance = element clc:ExcludedInstance
{
  attribute ref { xsd:string }
  , error
}

```

```

calculation = element clc:Calculation
{
  attribute name { xsd:string } ?
# If there is a name specified for this calculation in
# CalculationInput, the name must be present in
# CalculationOutput
}

```

```

, structureElementResult *
}

scenario = element clc:Scenario
{
  attribute name { xsd:string }
  , calculation +
}

structureElementResult = element dp:* # name of structure element
{
  structureElementIdentifiers # see structureCommon.rnc
  ,
  (
    featureResults ?
    & validationResults ?
  )
  , structureElementResultLinks ?
}

featureResults = element dp:Features
{
  (
    attributeResult
    | operationResult
  ) *
}

validationResults = element dp:Validations
{
  validationResult *
}

validationResult = element dp:*
# name of the validation or validationgroup that is requested in
# the input
{
  (
    validationMessage
    | validationError
  ) *
  # when there is no message or error, the validation has
  # succeeded or all validations inside the validation group
  # have succeeded
}

validationMessage = element dp:ValidationMessage
{
  (
    validationCommonAttributes
    & attribute prerequisite { "true" } ?
    # the prerequisite flag is NOT true, if and only if it
    # is raised by a requested Validation or by the
    # main-part of a Validation Group that is either a
    # requested Validation Group, or a descendant of a
    # requested Validation Group of which all its
    # ancestor's prerequisite-parts are true.
  )
}

```



```

# In all other cases the prerequisite flag is true.
)
'
(
element clc:Message { text }
& element dp:MessageParameters
{
element dp:*
{
attribute unit { xsd:string}?
# in case the parameter is a quantity
, xsd:string # the value of the parameter
} *
} ?
)
}

validationError = element dp:ValidationError
{
validationCommonAttributes
, error
}

validationCommonAttributes =
(
attribute id {
xsd:string {
pattern =
"ID:[0-9a-fA-F]{8}\{4}\
F]{4}\{4}\{12}"
}
} # lid of the validation, which is a guid
& attribute ver-nr {
xsd:string {
pattern =
"[0-9]+\.[0-9]+(\/.*\/[0-9]+\.[0-9]+)*\*?"
}
} # the version number of the validation
& attribute name { xsd:string }
# the navigation name of the validation message
& attribute instance-ref { xsd:IDREF }
# the id of the structure element instance this message is
# raised in
& attribute severity { "1" | "2" | "3" | "4" }
# the severity of the validation
)

attributeResult = element dp:* # name of attribute
{
simpleResultValue
}

operationResult = element dp:* # name of operation
{
value
# when Call is omitted or when option calculate-all is set
# in Input only applicable at operations without parameters

```

```

| call *
}

call = element dp:Call
{
  attribute ref { xsd:IDREF } ?
  # references the call of the operation from calculation
  # input
  , callParameters
  , callResult
}

callResult = element dp:Result
{
  value # see structureCommon.rnc
}

callParameters = element dp:Parameters
{
  operationArgument * # see structureCommon.rnc
}

structureElementResultLinks = element dp:Links
{
  structureElementResultLink +
}

structureElementResultLink = element dp:* # name of link
{
  (
    structureElementResult
    | instanceReference
  ) *
}

instanceReference = element clc:Reference
{
  attribute ref { xsd:IDREF }
  # references the id of an instance when specified in the
  # input
}

simpleResultValue =
(
  simpleValue # see structureCommon.rnc
)

```

structureCommon

```

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
namespace xsi = "http://www.w3.org/2001/XMLSchema-instance"
namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"
namespace dp = "http://www.example.org/myDeployment"

```

```

structureElementInput = element dp:*
# name of structure element
{
    structureElementIdentifiers
    , structureElementInputFeatures ?
    , structureElementInputLinks ?
}

structureElementInputFeatures = element dp:Features
{
    (
        structureElementInputAttribute
        | structureElementInputOperationValueSource
    ) +
}

structureElementInputLinks = element dp:Links
{
    structureElementInputLink +
}

structureElementInputLink = element dp:* # name of link
{
    attribute aggregate { xsd:boolean }?
    # default false;
    # When true, the resources under this link will be sent
    # separately through the output aggregation interface of
    # Calculator. In this case the link does not have any
    # children.
    , attribute id { xsd:string }?
    # the id used in the output aggregation interface to
    # identify a corresponding resource that is sent
    ,
    (
        structureElementInput *
        | instanceReferenceInput *
    )
}

structureElementInputAttribute = element dp:*
# name of attribute
{
    valueSourceOption ?
    ,
    (
        nilValue
        | simpleValue
    )
}

structureElementInputOperationValueSource = element dp:*
# name of operation
{
    valueSourceOption
}

```

```

instanceReferenceInput = element clc:Reference
{
    attribute ref { xsd:IDREF }
# references an id of an instance (structure element input)
# in CalculationData or Scenario
}

structureElementRequest = element dp:*
# name of structure element
{
    attribute ref { xsd:IDREF } ?
# references an id in CalculationData; when omitted all
# instances of this structure element are referenced.
    ,
    (
        structureElementRequestValidations ?
        & structureElementRequestFeatures ?
    )
    , structureElementRequestLinks ?
}

structureElementRequestValidations = element dp:Validations
{
    structureElementRequestValidation *
}

structureElementRequestFeatures = element dp:Features
{
    (
        structureElementRequestAttribute
        | structureElementRequestOperation
    ) *
}

structureElementRequestValidation = element dp:*
# name of validation or validation group
{
    empty
}

structureElementRequestAttribute = element dp:*
# name of attribute
{
    empty
}

structureElementRequestOperation = element dp:*
# name of operation
{
    operationCall *
}

structureElementRequestLinks = element dp:Links
{
    structureElementRequestLink +
}

```

```

}

structureElementRequestLink = element dp:*
# name of link
{
    (
        structureElementRequest
        | instanceReferenceRequest
    ) *
}

instanceReferenceRequest = element clc:Reference
{
    attribute ref { xsd:IDREF } ?
# References an id of an instance (structure element input)
# in CalculationData or Scenario. When omitted, all
# corresponding clc:References in the CalculationData or
# Scenario will be selected
}

operationCall = element dp:Call
{
    attribute id { xsd:ID } ?
# identifies the call of the operation
    , operationArgument *
}

structureElementIdentifiers =
(
    (
        attribute id { xsd:ID }
# identifier of a new instance
        | attribute ref { xsd:IDREF }
# references the identifier of an existing instance
    ) ?
    & attribute ext-id { xsd:string } ?
)

valueSourceOption =
(
    attribute value-source-option { xsd:string }
# name of the value source option
)

operationArgument = element dp:*      # name of operation parameter
{
    value
    | calculationParameterReference
# references a requestParameter from the
# calculationParameters element
# the operationArgument will receive the values that result
# from this calculationParameter
# When the same parameter is referenced more than once
# within an operation call the referring arguments will
# share the same value defined by the referenced request
# parameter.
}

```

```

# This request parameter will appear once in the calculator
# output. In other words, a request parameter is permuted
# only once within the call.
}

calculationParameterReference =
(
    attribute param-ref { xsd:string - ("unit" | "xsi:nil") } ?
# references a calculationParameter as specified in the
# calculation element to which this argument belongs
# the param-ref "date" is used to reference the calculation
# date param-ref cannot be "unit" or "xsi:nil" as those are
# reserved for quantityValue and nilValue respectively
)

value =
(
    simpleValue
    | tupleValue
    | setValue
)

simpleValue =
(
    indexedSimpleValue +
    | nonIndexedSimpleValue
)

indexedSimpleValue = element dp:Value
{
    calculationParameterValue +
    , nonIndexedSimpleValue
}

calculationParameterValue =
(
    attribute *-(xsi:nil | unit) { simpleValueType }
# calculationParameter with its value
# the name "date" means calcdatetime
# the name "unit" is reserved for quantityValue
# xsi:nil is reserved for nilValue
)

nonIndexedSimpleValue =
(
    simpleValueType
    | instanceReferenceValue
    | quantityValue
    | nilValue
    | error
)

simpleValueType =
(
    xsd:string          # also for tables
    | dateOrDateTime

```

```

        | float
        | xsd:integer
        | boolean
    )

instanceReferenceValue = element clc:Reference
{
    attribute ref { xsd:IDREF }
# references an id of an instance (structure element input)
# in CalculationData or Scenario
}

quantityValue =
(
    valueUnit
    , float
)

tupleValue =
(
    tupleMemberValue *
)

tupleMemberValue = element dp:*           # name of tuple member
{
    value
}

setValue =
(
    tdSetValue +
    | notTDSetValue
)

tdSetValue = element dp:Value
{
    dateTime
    ,
    (
        setElement *
        | error
        | nilValue
    )
}

notTDSetValue =
(
    setElement *
)

setElement = element dp:SetElement
{
    attribute id { xsd:ID } ?
    , value
}

error =

```

```

(
    pxError
    | userError
)

pxError = element clc:Error
{
    (
        attribute nr { xsd:integer }
        & attribute id { xsd:string }
        & dateTime ?
    )
    , xsd:string
}

userError = element dp:UserError
{
    (
        attribute id {
            xsd:string {
                pattern =
"ID:[0-9a-fA-F]{8}\{4}\
fA-F]{4}\{4}\{12}"
            }
        } # the lid of the error message, which is a guid
        & attribute ver-nr {
            xsd:string {
                pattern =
"[0-9]+\.[0-9]+(\/.*\/[0-9]+\.[0-9]+)*\*?"
            }
        } # the version number of the error message
        & attribute name { xsd:NCName }
        # the navigation name of the error message
        & attribute date {
            xsd:date
            | xsd:dateTime { pattern = ".*T[^Z+-.]*" }
        }? # when calculation date dependent
    ),
    (
        (
            element clc:Message { text }
            , element dp:MessageParameters {
                element dp:* {
                    xsd:string
                    # the value of the parameter
                }*
            }?
        )
        & element clc:MessageStack { text }
        # the message history of the user defined error
    )
}

nilValue =
(
    attribute xsi:nil { "true" }
)

```



```

)

valueUnit =
(
    attribute unit { xsd:string }
)

dateTime =
(
    attribute date { dateOrDateTime }
)

dateOrDateTime =
(
    xsd:date
    | xsd:dateTime { pattern = ".*T[^Z+-.]*" }
)

float =
(
    #we don't support NaN or INF values for reals
    xsd:float { pattern="^[^IN].*" }
)

boolean =
(
    #we don't support 0 or 1 for booleans
    xsd:boolean { pattern="^[01].*" }
)

```

Function Deployment Definitions

This section describes the calculator input and output XML used in function deployments.

functionInput

```

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/CalculatorElement"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
namespace dp = "http://www.example.org/myDeployment"

```

See <http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/#regexs>
 # for the regular expression language used by RelaxNG

```

include "functionCommon.rnc"
include "deploymentReference.rnc"
include "calculatorCommon.rnc"

```

```

start = functionInput

```

```

functionInput = element clc:CalculationInput
{
    functionInputAttributes,
    functionInputChildren
}

```

```

}

functionInputAttributes =
(
    calculationInputAttributes
# note that from errorLevels only break-on-error, ignore-
# input-domain-errors and # ignore-output-domain-errors are
# applicable for functionInput
)

functionInputChildren =
(
    deploymentReference
# references the function deployment package - from
# deploymentReference.rnc
    , testData ?
    , requestedFunctions
)
testData = element clc:TestData
{
    testSet *
}

requestedFunctions = element clc:RequestedFunctions
{
    function *
}

testSet = element clc:TestSet
{
    id
    ,
    (
        calculationDate ?
        & self ?
    )
    ,
    (
        variable
        | functionVariable
    ) *
}

function = element dp:*
# The element represents the function (short) name from the
# function deployment package. The function deployment package
# contains a list of functions that can be invoked
{
testSetReference *
    # if no TestSet is specified, this function will be
    # calculated with all TestSets
    , functionCall +
}

testSetReference = element clc:TestSet
{
    ref      # references the TestSet id

```

```

    }

    functionCall = element dp:Call
    {
        id
        , functionArgument *
    }

```

functionOutput

```

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
namespace dp = "http://www.example.org/myDeployment"

include "functionCommon.rnc"
include "calculatorCommon.rnc"
include "runtimeConfiguration.rnc"

start = functionOutput

functionOutput = element clc:CalculationOutput
{
    functionOutputAttributes
    , functionOutputChildren
}

functionOutputAttributes =
(
    calculationOutputAttributes
)

functionOutputChildren =
(
    error
    |
    (
        runtimeConfiguration
        , testSetResult *
    )
)

testSetResult = element clc:TestSet
{
    ref      # references the TestSet in functionInput
    ,
    (
        simpleResults
        | permutationResults
    )
}

simpleResults =
(
    functionResult *

```

```

)

permutationResults =
(
    permutationResult *
)

functionResult = element dp:*          # name of calculated function
{
    functionCallResult *
}

permutationResult = element dp:Permutation
{
    variablePermutation
    , functionResult *
}

functionCallResult = element dp:Call
{
    ref          # references the Call in functionInput
    ,
    (
        functionCallSimpleResult
        | functionCallPermutationResult +
    )
}

functionCallSimpleResult =
(
    callParameters
    , callResult
)

callParameters = element dp:Parameters
{
    functionArgument *
}

callResult = element dp:Result
{
    value
}

functionCallPermutationResult = element dp:Permutation
{
    functionCallSimpleResult
}

variablePermutation =
(
    (
        calculationDate ?
        & self ?
    )
    ,
    (

```

```

        variable
        | functionVariable
    ) *
)

```

functionCommon

```

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
namespace xsi = "http://www.w3.org/2001/XMLSchema-instance"
namespace dp = "http://www.example.org/myDeployment "

functionArgument = element dp:*
# name of the function parameter
{
    value
}

calculationDate = element clc:CalculationDate
{
    dateOrDateTimeMultiValue
}

self = element clc:Self
{
    identityMultiValue
}

variable = element dp:*          # (short) name of the variable
{
    value
    | scalarValue
}

functionVariable = element dp:*   # (short) name of the variable
{
    value
# constant value
    | functionName
# name of a function (or more functions in case of
# permutations) that exists in the Function Deployment
# Package
    | externalFunctionName
# name of an external function that exists in
# ExternalFunctions.Settings
}

functionName = element dp:FunctionName
{
    nameValue
}

```

```

externalFunctionName = element dp:ExternalFunctionName
{
    xsd:string
}

value =
(
    scalarValue
    | tupleValue
    | setValue
)

scalarValue =
(
    tdScalarValue +
    | notTDScalarValue
)

tdScalarValue = element dp:Value
{
    attribute date { dateOrDateTimeMultiValue }
    ,
    (
        quantityValue
        | simpleTypeMultiValue
        | nilValue
        | error
    )
}

notTDScalarValue =
(
    simpleTypeMultiValue           # see calculatorCommon.rnc
    | quantityValue
    | nilValue
    | error
)

quantityValue =
(
    valueUnit
    , numberMultiValue
)

tupleValue =
(
    identity ?
    , tupleMemberValue *
)

tupleMemberValue = element dp:*      # name of Tuple Member
{
    value
}

identity = element clc:Identity
{

```

```

        identityMultiValue
    }

    setValue =
    (
        setElement *
    )

    setElement = element dp:SetElement
    {
        (
            id ?
            & attribute start-date { dateOrDateTimeMultiValue }
            ?          # in case of a time dependent set
            & attribute end-date { dateOrDateTimeMultiValue }
            ?          # in case of a time dependent set
        )
        , value
    }

    nilValue =
    (
        attribute xsi:nil { "true" }
    )

    valueUnit =
    (
        attribute unit { xsd:string }
    )

    nameValue =
    (
        xsd:string { pattern = "(\i)([\c])*(;(\i)([\c])*)*" }
        # \i is the first character in an XML identifier (as defined
        # by w3c): any (unicode) letter, the character '_', or the
        # character ':'
        # \c is the set of characters matched by NameChar (as
        # defined by w3c)
    )

    error = element clc:Error
    {
        (
            attribute nr { xsd:integer }
            & attribute id { xsd:string }
            & attribute date { dateOrDateTimeMultiValue } ?
        )
        , xsd:string
    }

    id =
    (
        attribute id { xsd:ID }
    )

    ref =

```

```
(
    attribute ref { xsd:IDREF }
)
```

Common Definitions

This section provides common XML definitions used in all types of deployments.

deploymentReference

```
namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

deploymentReference = element clc:DeplR
{
    deploymentName
    & versionSelector
    &
    (
        activeDate
        | versionNumber
    ) ?
}

deploymentName =
(
    attribute dep-name { xsd:string }
)

activeDate =
(
    attribute act-date { xsd:date }
)

versionNumber =
(
    attribute ver-nr { xsd:string }
)

versionSelector =
(
    attribute ver-sel { "specific" | "last" | "active on date" }
)
```

RuntimeConfiguration

```
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/
CalculatorElement"
```



```

runtimeConfiguration = element clc:RuntimeConfiguration
# this is the runtime configuration that was selected in the input
# with DeplR
{
    runtimeConfigurationAttributes
    , empty
}

runtimeConfigurationAttributes =
(
    attribute name { xsd:string }
    # the name of the selected configuration
    , attribute version { xsd:string }
    # the version of the selected configuration
    , attribute id { guid }
    # the id of the selected configuration
)

guid =
(
    xsd:string
    {
        pattern="ID: ([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}) | (\{ [0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12} \}) "
    }

    # readable format of the regexp
    #
    # (
    #     [0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-
    #     [0-9a-fA-F]{4}-[0-9a-fA-F]{12}
    # )
    # |
    # (
    #     \{ [0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-
    #     [0-9a-fA-F]{4}-[0-9a-fA-F]{12} \}
    # )
)

```

CalculatorCommon

```

namespace clc =
"http://www.solcorp.com/ns/ProductXpress/CalculationInputOutput/"
CalculatorElement"

datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"

calculationInputAttributes =
(
    attribute timer { xsd:boolean } ?
    & attribute request-id { xsd:string } ?
    # id of the request
    & errorLevels

```

```

)

calculationOutputAttributes =
(
    attribute request-id { xsd:string } ?
    # the id of the request provided in CalculationInput
    &
    # if timer is true in CalculationInput the following
    # attributes are not optional
    (
        attribute queue-time { xsd:decimal }
        & attribute receive-time { xsd:decimal }
        & attribute external-function-time { xsd:decimal }
        & attribute calculation-time { xsd:decimal }
    ) ?
)

errorLevels =
(
    attribute break-on-error { xsd:boolean } ?
    # default false
    & attribute break-on-severity { "1" | "2" | "3" | "4" }
    # default 4
    & attribute ignore-input-structure-errors { xsd:boolean } ?
    # default false
    & attribute ignore-output-structure-errors { xsd:boolean } ?
    # default false
    & attribute ignore-input-mapping-errors { xsd:boolean } ?
    # default false
    & attribute ignore-output-mapping-errors { xsd:boolean } ?
    # default false
    & attribute check-link-multiplicity { xsd:boolean } ?
    # default false
    & attribute check-input-values { xsd:boolean } ?
    # default false
    & attribute check-output-values { xsd:boolean } ?
    # default false
)

simpleTypeMultiValue =
(
    dateOrDateTimeMultiValue
    | numberMultiValue
    | booleanMultiValue
    | identityMultiValue
    | stringMultiValue
    # as stringValue can be any string, static checking of the
    # value is not possible
    | tableNameMultiValue
)
dateOrDateTimeMultiValue =
(
    dateMultiValue
    |
    dateTimeMultiValue
)

```

```
# In the following definitions we use regular expression to
# specify the format of various values.
# See for details about regular expressions in XML:
# http://www.xmlschema-reference.com/regularExpression.html
```

```
numberMultiValue =
# reals and integers
(
    xsd:string { pattern="(\s*)((-?(0|([1-
9]\d*))(\.\d*))|(feature:\i\c*@i\c*))|((<|\[)((-?(0|([1-
9]\d*))(\.\d*))|(feature:\i\c*@i\c*))(>|\])@((0\.\d*[1-
9])|([1-9]\d*(\.\d*))|(feature:\i\c*@i\c*))?)");(((-
?(0|([1-9]\d*))(\.\d*))|(feature:\i\c*@i\c*))|((<|\[)((-
?(0|([1-9]\d*))(\.\d*))|(feature:\i\c*@i\c*))(>|\])@((0\.\d*[1-
9])|([1-9]\d*(\.\d*))|(feature:\i\c*@i\c*))?)"))*(\s*)" }
# readable format of the regexp
```

```
# readable format of the regexp
#
#      (\s*)
#      (
#          (
#              (-?(0|([1-9]\d*))(\.\d*))?
#              |
#              (feature:\i\c*@i\c*)
#          )
#          |
#          (
#              (<|\[)
#              (
#                  (-?(0|([1-9]\d*))(\.\d*))?
#                  |
#                  (feature:\i\c*@i\c*)
#              )
#              ;
#              (
#                  (-?(0|([1-9]\d*))(\.\d*))?
#                  |
#                  (feature:\i\c*@i\c*)
#              )
#              (>|\])
#              (
#                  @
#                  (
#                      (0\.\d*[1-9])
#                      |
#                      ([1-9]\d*(\.\d*))?
#                      |
#                      (feature:\i\c*@i\c*)
#                  )
#              )?
#          )
#      )
#      (
#          ;
```

```

#           (
#           (
#               (-?(0|([1-9]\d*))(\.\d*)?)
#               |
#               (feature:\i\c*@i\c*)
#           )
#           |
#           (
#               (<|\[)
#               (
#                   (-?(0|([1-9]\d*))(\.\d*)?)
#                   |
#                   (feature:\i\c*@i\c*)
#               )
#               ;
#               (
#                   (-?(0|([1-9]\d*))(\.\d*)?)
#                   |
#                   (feature:\i\c*@i\c*)
#               )
#               (>|\])
#               (
#                   @
#                   (
#                       (0\.\d*[1-9])
#                       |
#                       ([1-9]\d*(\.\d*)?)
#                       |
#                       (feature:\i\c*@i\c*)
#                   )
#               )?
#           )
#       )
#   ) *
#   (\s*)
#
booleanMultiValue =
(
    xsd:string { pattern =
"(\s*)(true|false|(feature:\i\c*@i\c*)) (;(true|false|
(feature:\i\c*@i\c*)))?(\s*)"
    }
# readable format of the regexp
#
#   (\s*)
#   (
#       true
#       |
#       false
#       |
#       (feature:\i\c*@i\c*)
#   )
#   (
#       ;
#       (

```

```

#           true
#           |
#           false
#           |
#           (feature:\i\c*\i\c*)
#           )
#       )?
#       (\s*)
#
)

identityMultiValue =
(
    stringMultiValue
)

stringMultiValue =
(
    # escaped characters and sequences (not able to define in
    # regexps):
    # - \\ (literal '\')
    # - \; (literal ';')
    # - \" (literal '"')
    # - \feature:\i\c*\i\c* (to reference a feature)
    xsd:string
    # ; means multiple values
    # \ is used to escape characters or sequences
    # " is used to allow whitespaces before and after non
    # whitespace characters in strings; the " itself is not part
    # of the string value whitespaces (space, tab, newline,
    # return) before the first and after the last non whitespace
    # character are ignored
)

tableNameMultiValue =
(
    stringMultiValue
)

dateMultiValue =
(
    xsd:string
    {
        pattern =
        "(\s*)((\d{4}-(0[1-9]|1[012])-(0[1-9]|[12][0-9]|3[01]))|(feature:
        \i\c*\i\c*))|((<|\[)((\d{4}-(0[1-9]|1[012])-(0[1-9]|[12][0-9]|3[0
        1]))|(feature:\i\c*\i\c*));((\d{4}-(0[1-9]|1[012])-(0[1-9]|[12][0
        -9]|3[01]))|(feature:\i\c*\i\c*))(>|\])@((\d+)|(feature:\i\c*\i
        \c*)) (day|week|month|year|(feature:\i\c*\i\c*)))?(
        endofmonth?))((\d{4}-(0[1-9]|1[012])-(0[1-9]|[12][0-9]|3[01]))
        |(feature:\i\c*\i\c*))|((<|\[)((\d{4}-(0[1-9]|1[012])-(0[1-9]|[12
        ][0-9]|3[01]))|(feature:\i\c*\i\c*));((\d{4}-(0[1-9]|1[012])-(0[1
        -9]|12][0-9]|3[01]))|(feature:\i\c*\i\c*))(>|\])@((\d+)|(featur
        e:\i\c*\i\c*)) (day|week|month|year|(feature:\i\c*\i\c*)))?(
        endofmonth?)))*(\s*)"
    }
)
# readable format of the regexp

```

```

#
#      (\s*)
#      (
#          (
#              (\d{4})-(0[1-9]|1[012])-(0[1-9]|[12][0-9]|3[01]))
#              |
#              (feature:\i\c*@\i\c*)
#          )
#          |
#          (
#              (<|\[)
#              (
#                  (\d{4})-(0[1-9]|1[012])-(
#                      0[1-9]|[12][0-9]|3[01]))
#                  |
#                  (feature:\i\c*@\i\c*)
#              )
#              ;
#              (
#                  (\d{4})-(0[1-9]|1[012])-(
#                      0[1-9]|[12][0-9]|3[01]))
#                  |
#                  (feature:\i\c*@\i\c*)
#              )
#              (>|\])
#              (@((\d+)|(feature:\i\c*@\i\c*))
#              ( (day|week|month|year|(feature:\i\c*@\i\c*)) )?
#              ( endofmonth) )?
#          )
#      )
#      (
#          ;
#          (
#              (
#                  (\d{4})-(0[1-9]|1[012])-(
#                      0[1-9]|[12][0-9]|3[01]))
#                  |
#                  (feature:\i\c*@\i\c*)
#              )
#              |
#              (
#                  (<|\[)
#                  (
#                      (\d{4})-(0[1-9]|1[012])-(
#                          0[1-9]|[12][0-9]|3[01]))
#                      |
#                      (feature:\i\c*@\i\c*)
#                  )
#                  ;
#                  (
#                      (\d{4})-(0[1-9]|1[012])-(
#                          0[1-9]|[12][0-9]|3[01]))
#                      |
#                      (feature:\i\c*@\i\c*)
#                  )
#                  (>|\])
#                  (@((\d+)|(feature:\i\c*@\i\c*))

```

```

#           ( (day|week|month|year|
#               (feature:\i\c*@i\c*))
#           )
#       )?
#       ( endofmonth)?
#   )
#   )
#   ) *
#   (\s*)
#
dateTimeMultiValue =
(
    xsd:string
    {
        pattern =
"(\s*)((\d{4})-(0[1-9]|1[012])-(0[1-9]|1[2][0-9]|3[01]) (T([01]\d|2
[0-3]):[0-5]\d:[0-5]\d))|(feature:\i\c*@i\c*))|((<|\[) ((\d{4})-(0[
1-9]|1[012])-(0[1-9]|1[2][0-9]|3[01]) (T([01]\d|2[0-3]):[0-5]\d:[0-
5]\d))|(feature:\i\c*@i\c*));((\d{4})-(0[1-9]|1[012])-(0[1-9]|1[2]
[0-9]|3[01]) (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))|(feature:\i\c*@i\
c*))(>|\]) (@((\d+)|(feature:\i\c*@i\c*)))
(second|minute|hour|day|week|month|year|(feature:\i\c*@i\c*))?(
endofmonth?)) ; (((\d{4})-(0[1-9]|1[012])-(0[1-9]|1[2][0-9]|3[01]) (
T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))|(feature:\i\c*@i\c*))|((<|\[) (
\d{4})-(0[1-9]|1[012])-(0[1-9]|1[2][0-9]|3[01]) (T([01]\d|2[0-3]):[
0-5]\d:[0-5]\d))|(feature:\i\c*@i\c*));((\d{4})-(0[1-9]|1[012])-(0
[1-9]|1[2][0-9]|3[01]) (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))|(feature
:\i\c*@i\c*))(>|\]) (@((\d+)|(feature:\i\c*@i\c*)))
(second|minute|hour|day|week|month|year|(feature:\i\c*@i\c*))?(
endofmonth?))) *(\s*)"
    }
    (\s*)
    (
        (
            (\d{4})-(0[1-9]|1[012])-(
                0[1-9]|1[2][0-9]|3[01])
                (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))
            |
            (feature:\i\c*@i\c*)
        )
        |
        (
            (<|\[)
            (
                (\d{4})-(0[1-9]|1[012])-(
                    0[1-9]|1[2][0-9]|3[01])
                    (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))
                |
                (feature:\i\c*@i\c*)
            )
        )
        ;
        (
            (\d{4})-(0[1-9]|1[012])-(
                0[1-9]|1[2][0-9]|3[01])
                (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d))

```

```

#           |
#           (feature:\i\c*@i\c*)
#       )
#       (>|\])
#       (@((\d+)|(feature:\i\c*@i\c*))
#           ( (second|minute|hour|day|week|month|year|
#               (feature:\i\c*@i\c*))
#           )
#       )?
#       ( endofmonth)?
#   )
# )
# (
#     ;
#     (
#         (
#             (\d{4}-(0[1-9]|1[012]))-
#             (0[1-9]|[12][0-9]|3[01])
#             (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d)
#             |
#             (feature:\i\c*@i\c*))
#         |
#         (
#             (<|\[)
#             (
#                 (\d{4}-(0[1-9]|1[012]))-
#                 (0[1-9]|[12][0-9]|3[01])
#                 (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d)
#                 |
#                 (feature:\i\c*@i\c*))
#             )
#         ;
#         (
#             (\d{4}-(0[1-9]|1[012]))-
#             (0[1-9]|[12][0-9]|3[01])
#             (T([01]\d|2[0-3]):[0-5]\d:[0-5]\d)
#             |
#             (feature:\i\c*@i\c*))
#         )
#         (>|\])
#         (@((\d+)|(feature:\i\c*@i\c*))
#             ( (second|minute|hour|day|week
#                 |month|year|
#                 (feature:\i\c*@i\c*))
#             )
#         )?
#         ( endofmonth)?
#     )
# )
# ) *
# (\s*)
# )

```


XSLT Extension Functions

This section describes XSLT extension functions.

Introduction

The ProductXpress XSTL extension functions are available for XSLT transformations through the transformation API (see Perl Interface (on page 97)). These extension functions allow you to obtain information from the run-time configuration. The selection of the run-time configuration is the same as for calculation requests (See Calculator Input-Output Specification for the Push Calculator (on page 194)).

The namespace for the XSLT extension functions is
<http://www.solcorp.com/ns/ProductXpress/Repository/ExtensionFunction>
[\(<http://www.solcorp.com/ns/productxpress/repositoryapi/extensionfunction>\)](http://www.solcorp.com/ns/productxpress/repositoryapi/extensionfunction).

In general, all extension functions have a string-in string/boolean-out interface.

API

The following sections describe the API functions in alphabetic order.

We will use certain categories of component types in the following sections to describe the component(s) a function applies to:

Category	Component Type
all components	all component types
versioned components	attribute, dynamic deployment, dynamic deployment environment, error message, external function, function, function variable, link, operation, static deployment, static deployment environment, stereo type, structure element, table, tag, text block, validation, validation group, value definition, variable
non versioned components	dimension, function parameter, mapping group, mapping slice, message parameter, table property, table return type, tuple member, value domain, value mapping, value source

- ❗ Some of the component types themselves are groups (such as value mapping). In the bottom table of *Appendix E - Transition Matrix* in the *ProductXpress Path Language Reference Guide* it is explained which component types these groups represent.

Some functions only apply to one or more specific component types or not on all component types in a category. In those cases, the specific component types and/or exceptions will be mentioned explicitly.

In general, when an API results in an exception, an error message is returned. The main exceptions are:

- ▶ when a reference cannot be found within a run-time configuration
- ▶ when the component (that is mentioned in the description of the API) does not support the API

allowsSubtypes

applicable on: link

Input	Output	Description
Reference	Boolean	Returns whether or not the link that is referenced allows subtypes of the structure element it refers to.

componentID

applicable on: all components

Input	Output	Description
Reference	String	<p>Returns the <i>component ID</i> of the component that is referenced. The <i>component ID</i> can be used as a NameTest in a path expression in <i>evaluatePath</i> (see the <i>ProductXpress Path Language Reference Guide</i> for the NameTest in a path). The <i>component ID</i> is identical for all versions of the same component.</p> <p>Note: In case of a non versioned component, such as a function parameter or tuple member, the <i>component ID</i> is the concatenation of the component ID of the versioned parent component (e.g. function or tuple value definition) and the <i>component ID</i> of the non versioned component that is referenced.</p> <p>The regular expression of the format of the <i>component ID</i> is as follows: <code>ID_[0-9a-fA-F]{8}\{(\{4\}\{4\}\{4\}\{4\}\{3\}\{12\}(\{([0-9a-fA-F]{8}\{4\}\{4\}\{4\}\{4\}\{(\{4\}\{3\}\{12\})\})?)</code></p> <p>Example versioned component: <code>ID_1f118dfc-42cd-4e08-803d-71629098fde5</code></p> <p>Example non versioned component: <code>ID_5432c725-cb1d-4fdf-975c-99799e54716e_cb4fd058-02de-4612-86d4-8f5cca00f313</code></p>

componentType

applicable on: versioned components

Input	Output	Description
Reference	("attribute" "dynamic deployment environment" "error message" "external function" "function" "function variable" "link" "operation" "static deployment environment" "stereotype" "structure element" "table" "tag" "text block " "validation" "validation group" "value definition" "variable")	Returns the <i>type</i> of the component that is referenced.

creationDate

applicable on: versioned components except links and features.

Input	Output	Description
Reference	String	Returns the <i>creation date</i> of the component that is referenced; the format is yyyy-mm-dd.

dataType

applicable on: value definition

Input	Output	Description
Reference	("boolean" "date" "datetime" "identity" "integer" "quantity" "real" "string" "tableId")	Returns the <i>data type</i> of the value definition that is referenced.

defaultValue

applicable on: feature, function variable and variable

Input	Output	Description
Reference	String	Returns the <i>default value</i> of a feature or variable that is referenced.

documentation

applicable on: all components

Input	Output	Description
Reference	String	Returns the <i>documentation</i> of the component that is referenced.

entryPoint

applicable on: n.a.

Input	Output	Description
n.a.	String	<p>In case of a dynamic deployment: returns the financial product entry point of the deployment.</p> <p>In case of a static deployment: returns the deployment structure element on which the financial product entry point of the deployment is mapped.</p> <p>Note: in case of multiple deployment structure elements are mapped on the financial product entry point, only one is returned (order is undefined).</p>

evaluatePath

applicable on: the optional reference can be any of the versioned components

Input	Output		Description
Reference	PXPath	nodeSet	Returns zero or more <i>references to components</i> that result from the evaluation where the optional Reference parameter is used as the starting point. The references are returned as a nodeSet (See <i>ProductXpress Path Language Reference Guide</i>) of the path.

When no reference is passed as the starting point, one can navigate to the following components from the root:

- ▶ **Deployment:** this is either the static or dynamic entry point deployment of the configuration. From a Deployment we can navigate to:
 - ♦ **Structure Element:** this is the *financial product entry point* referenced by the deployment (both for dynamic and static deployments)
 - ♦ **Deployment Environment:** this is the deployment environment referenced by the deployment.
- ▶ **Structure Element:**
 - ♦ in case of a **dynamic deployment**, this returns the entry point referenced by the deployment and all *resource roots* the entry point references;
 - ♦ in case of a **static deployment**, this returns all resource roots in the static deployment environment and all their descendant deployment structure elements.

instantiables

applicable on: n.a.

Input	Output		Description
NavigationName	String	String	<p>If a NavigationName is provided, returns zero or more <i>references to components</i> with navigation name <i>NavigationName</i> that occur in the list of instantiables in either a static or dynamic deployment.</p> <p>If a NavigationName is not provided, returns zero or more <i>references to components</i> that are instantiable in either a dynamic or static deployment.</p> <p>The references in the list are separated by a semi-colon.</p>

isAbstract

applicable on: structure element

Input	Output	Description
Reference	Boolean	Returns whether or not the structure element that is referenced is abstract.

isDeferred

applicable on: value mapping

Input	Output	Description
Reference	Boolean	Returns whether or not the mapping that is referenced is deferred (a.k.a. external).

isEntered

applicable on: value source

Input	Output	Description
Reference	Boolean	Returns whether or not the value source that is referenced is of type "Entered".

isEntryPoint

applicable on: structure element

Input	Output	Description
Reference	Boolean	Returns whether or not the structure element that is referenced is a resource root candidate.

isExternal

applicable on: table

Input	Output	Description
Reference	Boolean	Returns whether or not the table that is referenced is external.

isFinal

applicable on: feature, link and structure element

Input	Output	Description
Reference	Boolean	Returns whether or not the feature, link or structure element that is referenced is final.

isInherited

applicable on: feature and link

Input	Output	Description
Reference	Structure Element	Boolean
		Returns whether or not the Feature or Link (Reference) is inherited in Structure Element.

isInstantiable

applicable on: structure element

Input	Output	Description
Reference	Boolean	Returns whether or not the Structure Element is marked as <i>Instantiable</i> within a dynamic deployment.

isMandatory

applicable on: link

Input	Output	Description
Reference	Boolean	Returns whether or not the link that is referenced is mandatory.

isMappedHere

applicable on: value mapping

Input	Output	Description
Reference	Boolean	Returns whether or not the mapping that is referenced is mapped here.

isPartial

applicable on: tuple value mapping

Input	Output	Description
Reference	Boolean	Returns whether or not the tuple mapping that is referenced is partially mapped.

isPrerequisite

applicable on: validation group member

Input	Output	Description
Reference	Boolean	Returns whether or not the validation group member that is referenced is a prerequisite.

isTimeDependent

applicable on: value definition

Input	Output	Description
Reference	Boolean	Returns whether or not the value definition that is referenced is time dependent.

linkMultiplicity

applicable on: link

Input	Output	Description
Reference	("multiple" "single" "zero")	Returns the multiplicity of the link that is referenced.

linkType

applicable on: link

Input	Output	Description
Reference	("complement part" "navigation" "owning" "resource")	Returns the type of the link that is referenced.

mappingPath

applicable on: value mapping

Input	Output	Description
Reference	String	Returns the <i>mapping path</i> of the mapping or mapping slice that is referenced.

❗ The mapping path contains logical names and not IDs.

messageText

applicable on: Error Message or Validation

Input	Output	Description
Reference	String	Returns the <i>text</i> of the Error or Validation Message that is referenced.

❗ The content is returned as the raw XML string (as in PXML), where the parameter IDs are replaced with their names.

navigationList

applicable on: link

Input	Output	Description
Structure Element Reference	Link Reference nodeSet	Returns zero or more <i>references to links</i> that occur in the navigation list of the referenced Navigation Link within the context of the referenced Structure Element. When the referenced Link is not a Navigation Link or it is a Navigation Link but not of the Structure Element (or of its supertypes) or it is a Navigation Link of the Structure Element (or of its supertypes) but contains no Links, the returned list of Link References is empty.

navigationName

applicable on: versioned components, function and operation parameters and tuple members

Input	Output	Description
Reference	String	Returns the <i>navigation name</i> of the component that is referenced.

name

applicable on: all components

Input	Output	Description
Reference	String	Returns the <i>name</i> of the component that is referenced.

overrides

applicable on: link, feature and value source

Input	Output	Description
Reference	Boolean	Returns whether or not the component (link, feature or value source) that is referenced overrides its base type.

remoteMappingMode

applicable on: value mapping

Input	Output	Description
Reference	("in composition ancestor" "in subtype")	Returns the <i>mapping mode</i> of the mapping that is referenced.

severity

applicable on: validation

Input	Output	Description
Reference	String	Returns the <i>severity</i> of the validation that is referenced.

shortName

applicable on: versioned components, function and operation parameters and tuple members

Input	Output	Description
Reference	String	Returns the <i>short name</i> of the component that is referenced.

subtypesOf

applicable on: structure element

Input	Output	Description
Reference	String	Returns the available <i>subtypes</i> of the component that is

		referenced.
--	--	-------------

symbol

applicable on: enumeration

Input	Output	Description
Reference	String	Returns the <i>symbol</i> of the enumeration that is referenced.

table

applicable on: table

Input	Output	Description
Reference	nodeSet	Returns the contents of the internal table that is referenced as a nodeSet.

tablePropertyValue

applicable on: table

Input		Output	Description
Reference	Property	String	Returns the <i>value</i> of the table property of the table that is referenced.

tagValue

applicable on: tag

Input	Output	Description
Reference	String	Returns the <i>value</i> of the tagged value that is referenced.

text

applicable on: textblock

Input	Output	Description
Reference	String	Returns the <i>text</i> of a textblock that is referenced.

typeExtension

applicable on: document

Input	Output	Description
Reference	String	Returns the <i>type extension</i> of a document that is referenced.

validationGroupMemberPath

applicable on: validation group member

Input	Output	Description
Reference	String	Returns the <i>structure path</i> of a validation group member that is referenced.

valueDomain

applicable on: value definition

Input	Output	Description
Reference	String	Returns the <i>value domain</i> of a value definition that is referenced.

Output examples:

<1;10];[20;30>

a;b;c

versionNumber

applicable on: versioned components

Input	Output	Description
Reference	String	Returns the <i>version number</i> of the component that is referenced; when the referenced component does not have a version number, an empty string is returned.

visibility

applicable on: features and validations

Input	Output	Description
Reference	String	Returns the <i>visibility</i> of the component that is referenced.

Table XML Syntax

This section describes the XML representation of a Table that is returned by the table extension function.

```
start = tableData

tableData = element TableData
{
    column +
    , dimension +
    , tableValue +
}

column = element Column
{
    columnDataType
    , name
}
```

```

dimension = element Dimension
{
    dimensionDataType
    , name
    , dimensionValue *
}

tableValue = element TV
{
    string
}

columnDataType =
(
    attribute data-type
    {
        dataType | tableID
    }
)

name = element Name
{
    string
}

dimensionDataType =
(
    attribute data-type
    {
        dataType
    }
)

dimensionValue = element CV
{
    string
}

dataType =
(
    "boolean" | "date" | "integer" | "real" | "string" | "time"
)

tableID =
(
    "guid"
)

```

Description

The root of the XML that is returned by the table extension function is `TableData`. `TableData` has one or more `Column` elements (return values), one or more `Dimension` elements and one or more `TV` (table value) elements. Each `Dimension` contains `CV` elements (dimension values). The number of `TV` elements is in fact related to the dimension values: the number of `TV` elements is the number permutations of the dimension values of all dimensions.

We will describe the relation between dimension values and table values in more detail in the following section.

How to read multi-dimensional tables

As can be seen in the XML representation of a Table, a Table has a flat list of table values. These table values however have a specific order. The order corresponds with the dimension values. When there is only one dimension the order is evident: the first table value corresponds with the first dimension value, the second with the second, etcetera. However in the case of multiple dimensions, the order is more complicated though well-defined.

In multi-dimensional tables the first table value corresponds with the first dimension value of all dimensions. The second table value corresponds with the first dimension value of all dimensions but the last dimension whose second dimension value is taken. This continues until all dimension values of the last dimension are taken in order of appearance in the XML. In general, the order of the table values is related to the permutation of all dimension values, starting with the values of the last dimension and working backwards to the values of the first dimension.

Example

Suppose we have a four dimensional table with each two dimension values 1 and 2. The XML table looks as follows:

```
<TableData>
  <Column data-type="integer">
    <Name>r</Name>
  </Column>
  <Dimension data-type="integer">
    <Name>d1</Name>
    <CV>1</CV>
    <CV>2</CV>
  </Dimension>
  <Dimension data-type="integer">
    <Name>d2</Name>
    <CV>1</CV>
    <CV>2</CV>
  </Dimension>
  <Dimension data-type="integer">
    <Name>d3</Name>
    <CV>1</CV>
    <CV>2</CV>
  </Dimension>
  <Dimension data-type="integer">
    <Name>d4</Name>
    <CV>1</CV>
    <CV>2</CV>
  </Dimension>
  <!-- 16 TV elements (left out in this example) -->
</TableData>
```

We will have in total $2 \times 2 \times 2 \times 2 = 16$ table values (TV elements). The corresponding dimension values for these table values in ascending order is the "backwards permutation" of all dimension values (CV elements) of all dimensions (Dimension elements):

(1,1,1,1)

(1,1,1,2)

(1,1,2,1)

```
( 1 , 1 , 2 , 2 )
( 1 , 2 , 1 , 1 )
( 1 , 2 , 1 , 2 )
( 1 , 2 , 2 , 1 )
( 1 , 2 , 2 , 2 )
( 2 , 1 , 1 , 1 )
( 2 , 1 , 1 , 2 )
( 2 , 1 , 2 , 1 )
( 2 , 1 , 2 , 2 )
( 2 , 2 , 1 , 1 )
( 2 , 2 , 1 , 2 )
( 2 , 2 , 2 , 1 )
( 2 , 2 , 2 , 2 )
```

TableID as return type

When a table has `tableID` (guid) as its return type (`Column`), the `tableValues` contain a string that represents the ID (guid) of a table. The table (on page [259](#)) extension function can be invoked again with these IDs.

External Functions

ProductXpress Calculator provides facilities to invoke External Functions and use the results of External Functions in any calculation. External Functions are implemented as an entry point in a Dynamic Link Library (DLL) on a Microsoft Windows platform, or as an entry point in a Shared Library on IBM AIX and SUN Solaris platforms. Administration/External System Providers will develop these functions that contain logic to access the external systems.

The time spent in external functions can be obtained through a `CalcServer` setting.

External functions should be implemented as functions in a Dynamic Link Library (DLL) on the Microsoft Windows platform, or as functions in a shared library on the UNIX platforms.

Interface

The interface of an external function must comply with the following C-language function definition:

```
void ExtFuncName( PX_XFC xfc );
```

Where:

`ExtFuncName` is the name of the external function's entry point in the dll.

`xfc` is the External Function Context.

PX_XFC is a *typedef* defined in `PxXFC.h`. Programmers of External Functions should include this header file in their source code. The External Functions should be linked to the ProductXpress Library `PxXFC.lib` (on Microsoft Windows) or `libPxXFC.so` (on IBM AIX, SUN Solaris platforms).

The name of the function that is used in ProductXpress to call the External Function can be obtained with the following API call:

```
PX_CHAR* PxGetFunctionName(PX_XFC xfc);
```

It will return a zero terminated Unicode character string.

External Function Context

All communication between the ProductXpress Calculator and the External Function is done via the External Function Context. The External Function Context is passed to the External Function as a parameter when it is called and is passed back to the Calculator in calls of the ProductXpress External Function API.

An External Function Context is valid only for one call. When the External Function returns, the **xfc** becomes invalid. The same applies to all data retrieved via the External Function API.

Function Arguments

An External Function can have an arbitrary number of arguments. The actual number of arguments can be retrieved by the **PxGetArgCount** function:

```
int PxGetArgCount(( PX_XFC xfc );
```

The argument values can be obtained with the **PxGetArgValue** call:

```
PX_VAR PxGetArgValue(PX_XFC xfc, int n);
```

n is the argument number.

Arguments are numbered from zero onwards: ($n \leq 0 < \text{PxGetArgCount}(\text{xfc})$)

The values returned by **PxGetArgValue** are of the type PX_VAR. This is a handle to an oblique type used by the External Function API to exchange data from and to the Calculator (see Handling External Function Data (on page [265](#))).

-
- ❗ The actual number and data types of the arguments depends on the signature of the ProductXpress Function mapped to the External Function.
-

External Function Properties

An external function can have properties. Properties are configurable constants which can be defined in the ExternalFunctions.Settings file.

The implementation of the External Function can use Properties for any purpose. Properties are not used by the Calculator itself.

Property values can be obtained with the following call:

```
PX_CHAR* PxGetProperty(PX_XFC xfc, PX_CHAR* name);
```

name is the name of the property of which the value is to be obtained.

The function returns a zero terminated character string. If the property is not defined NULL is returned.

-
- ❗ It is possible to configure the same entry-point in a dll or shared object with different Properties as distinct External Functions for ProductXpress.
-

Returning Results from an External Function

An External Function should call the following function to set the return value. If it does not, the function value in ProductXpress will be Null.

```
void PxSetReturnValue(PX_XFC xfc, PX_VAR value);
```

value is the new return value of the External Function. If the function is called more than once, the **value** of the last call will be used. Passing NULL for **value** will result in a return value of Null in ProductXpress.

The values passed to **PxSetReturnValue** are of the type PX_VAR. This is a handle to an oblique type used by the External Function API to exchange data from and to the Calculator.

-
- ❗ The structure and type of the data returned to the ProductXpress Calculator must comply with the ValueDefinition of the Function mapped to it.
-

Information regarding how to create and manipulate this data is described in the next section.

Handling External Function Data

Input (arguments) and output (return value) data to and from External Functions in ProductXpress are passed using the type PX_VAR.

The data can have the following types: (*defined in PxTypes.h*)

Integer (<i>px_integer</i>)	Time Dependent Integer (<i>px_tdinteger</i>)
String (<i>px_string</i>)	Time Dependent String (<i>px_tdstring</i>)
Real (<i>px_real</i>)	Time Dependent Real (<i>px_tdreal</i>)
Time (<i>px_time</i>)	Time Dependent Time (<i>px_tdtype</i>)
Quantity (<i>px_quantity</i>)	Time Dependent Quantity (<i>px_tdquantity</i>)
Boolean (<i>px_bool</i>)	Time Dependent Boolean (<i>px_tdbool</i>)
Set (<i>px_set</i>)	Time Dependent Set (<i>px_tdset</i>)
Tuple (<i>px_tuple</i>)	

The data type of a PX_VAR (e.g. an argument obtained with **PxGetArgValue**) can be obtained with the following function:

```
PX_TYPE PxGetType(PX_VAR var);
```

Data in ProductXpress can also be Null. This is represented in the External Function API by a variable of type PX_VAR with value NULL.

-
- ❗ For arguments defined as type Real, the Calculator could also provide values of type integer if the value contains no fraction.
-

Plain Data Types

Integer, String, Real, Time and Boolean are so called plain data types.

The value of PX_VARS with those data types can be obtained with the following Calls:

```
PX_CHAR* PxGetValue(PX_VAR var);
int PxGetIntegerValue(PX_VAR var);           (for integer and boolean types only)
PX_BOOL PxGetBooleanValue(PX_VAR var); (for booleans only)
```

PxGetValue returns a pointer to a zero terminated string of Unicode characters. The format of the string is specified in the table below. **PxGetIntegerValue** returns an integer. For Boolean data that is 0 for *false* and 1 for *true*.

ProductXpress Data-Type	Unicode String Representation
Integer	String representation of an integer. [+]?[0-9]+
Real	String representation of a real. [-+]?[0-9]+[.]?[0-9]*
Time	Represented in format YYYY-MM-DDThh:mm:ss
boolean	True value is represented by string "true". False value is represented by string "false"
string	No conversion necessary for the value.

```
PX_VAR PxCreateInteger(PX_XFC xfc, int value);
PX_VAR PxCreateString(PX_XFC xfc, PX_CHAR* value);
PX_VAR PxCreateReal(PX_XFC xfc, PX_CHAR* value);
PX_VAR PxCreateTime(PX_XFC xfc, PX_CHAR* value);
PX_VAR PxCreateBoolean(PX_XFC xfc, PX_BOOL value);
```

The format for **PxCreateReal** and **PxCreateTime** is the same as previously described.

Quantity Data

Quantities exist of a Real Value and a Unit (e.g. Money with unit currency).

The Real value can be obtained with:

```
PX_CHAR* PxGetValue(PX_VAR var);
```

The Unit can be obtained with:

```
PX_CHAR* PxGetUnit(PX_VAR var);
```

Quantities can be created with the following function:

```
PX_VAR PxCreateQuantity(PX_XFC xfc, PX_CHAR* value, PX_CHAR* unit);
```

value is the value of type Real in the format as specified in the previous section.

unit is the unit value of type string. If the unit type is actually an integer type, a string representation of an integer should be passed.

Tuples

A tuple exists of a number of named values which are called members and is comparable with a struct in C. Each value itself is a PX_VAR again and has its own type. The values of the tuple members can be obtained by their names or through a tuple iterator. With a tuple iterator all members of a tuple can be accessed in sequence.

Function to retrieve tuple member values:

```
PX_VAR PxGetMember(PX_VAR tuple, PX_CHAR* member_name);
PX_TUPLEITER PxGetTupleIter(PX_VAR tuple);
PX_BOOL PxTupleIterValid(PX_TUPLEITER tupleiter);
PX_BOOL PxTupleIterNext(PX_TUPLEITER tupleiter);
```



```
PX_CHAR* PxTupleIterGetName(PX_TUPLEITER tupleiter);
PX_VAR PxTupleIterGetValue(PX_TUPLEITER tupleiter);
```

PxGetMember gets the value of a tuple member by name. **tuple** must be a PX_VAR of type `px_tuple`.

PxGetTupleIter can be used to obtain a tuple iterator. The iterator will be pointing at the first tuple member, or be invalid if there are no tuple members set. **tuple** must be a PX_VAR of type `px_tuple`.

PxTupleIterValid returns *true* if the iterator is pointing at a valid tuple member, *false* otherwise.

PxTupleIterNext moves the iterator to the next tuple member. It returns *false* if it was already at the last member. In that case the iterator becomes invalid and subsequent calls to **PxTupleIterValid** will return *false*.

PxTupleIterGetName can be used to obtain the name of the tuple member where the iterator is pointing.

PxTupleIterGetValue returns the value of the tuple member where the iterator is pointing.

Functions to create a tuple data structure:

```
PX_VAR PxCreateTuple(PX_XFC xfc);
int PxAddMember(PX_VAR tuple, PX_CHAR* name, PX_VAR value);
```

tuple is a PX_VAR handle that was returned by a call to **PxCreateTuple**

name is the name of the tuple member. This name has to be unique.

value is the value of the tuple member.

PxAddMember returns 0 in case of success, -1 in case of an error.

Sets

A set is a sequence of PX_VARS, called elements. All elements of a set must be of the same data type. The contents of a set can be retrieved through a set iterator. With a set iterator all elements of a set can be accessed in sequence.

Functions to access the elements of a set:

```
PX_SETITER PxGetSetIter(PX_VAR setvar);
PX_BOOL PxSetIterValid(PX_SETITER setiter);
PX_BOOL PxSetIterNext(PX_SETITER setiter);
PX_VAR PxSetIterGetValue(PX_SETITER setiter);
```

PxGetSetIter returns a new set iterator. The set iterator will be pointing at the first element of the set, or be invalid in case of an empty set. **setvar** should be a PX_VAR of type `set`.

PxSetIterValid returns *true* if the set iterator is pointing at a set element or *false* otherwise.

PxSetIterNext moves the set iterator to the next element in the set. If it was already at the last element, the set iterator will become invalid and **PxSetIterNext** will return *false*. Subsequent calls to **PxSetIterValid** will then also return *false*.

PxSetIterGetValue returns the value of the set element where the set iterator is pointing at.

Functions to create a set data structure:

```
PX_VAR PxCreateSet(PX_XFC xfc);
int PxInsert(PX_VAR setvar, PX_VAR value);
```

PxCreateSet creates an empty set.

PxInsert adds an element to a set. **setvar** should be a PX_VAR of type `set`. **PxInsert** will return 0 if successful or -1 otherwise.

Time Dependent Data Types

Time dependent data types model data that varies over time. They consists of a set of times and the actual values at each specific time.

To retrieve the dates and the values of a time dependent PX_VAR a Time Dependency iterator should be used.

Functions to access Time Dependent Data:

```
PX_TDITER PxGetTDIter(PX_VAR tdvar);
PX_BOOL PxTDIterValid(PX_TDITER tditer);
PX_BOOL PxTDIterNext(PX_TDITER tditer);
PX_CHAR* PxTDIterGetTime(PX_TDITER tditer);
PX_VAR PxTDIterGetValue(PX_TDITER tditer);
```

PxGetTDIter returns a Time Dependency iterator. It will be pointing at the first time value pair, **tdvar** should be a PX_VAR with a Time Dependent data type.

PxTDIterValid returns *true* if the iterator is pointing at a valid time value pair, *false* otherwise.

PxTDIterNext moves the iterator to the next time value pair. If there are no more time value pairs, it will return *false* and the iterator will become invalid

PxTDIterGetTime returns the time of the time value pair where the iterator is pointing at. The time is a zero terminated character string in the format :
YYYY-MM-DDThh:mm:ss

PxTDIterGetValue returns the value of the of the current time value pair. The type of the returned TD_VAR will be the time *independent* variant of the containing type.

Functions to create Time Dependent data structures:

```
PX_VAR PxCreateTDInteger(PX_XFC xfc);
PX_VAR PxCreateTDReal(PX_XFC xfc);
PX_VAR PxCreateTDString(PX_XFC xfc);
PX_VAR PxCreateTDTime(PX_XFC xfc);
PX_VAR PxCreateTDBoolean(PX_XFC xfc);
PX_VAR PxCreateTDQuantity(PX_XFC xfc);
PX_VAR PxCreateTDSet(PX_XFC xfc);
int PxSetValueAtTime(PX_VAR tdvar, PX_CHAR* time, PX_VAR value);
```

The **PxCreateTD*** functions create a Time Dependent PX_VAR of the specific type.

With the **PxSetValueAtTime** a value can be set at a specific time. The format of the **time** is the same as that returned by **PxTDIterGetTime** (see above). **tdvar** should be a PX_VAR with a time dependent type.

PxSetValueAtTime returns 0 on success and -1 on error.

Error Handling

External Functions can pass errors on to the ProductXpress Calculator by setting the error flag. If the error flag is set, this will cause the calling function in ProductXpress to fail. It is also possible to pass an error message to ProductXpress that explains the problem.

Functions of the External Function API can also set the error flag. They will do this when errors are made in the usage of the API. The implementation of the External Function can test for these errors. If it doesn't, the errors will slip through to ProductXpress and the calling function will fail.

In addition to setting the error flag functions of the External Function API will also return a NULL pointer instead of a valid handle or -1 when ever possible.

Functions to manipulate the error flag:

```
PX_BOOL PxHasErrors(PX_XFC xfc);
PX_CHAR* PxGetErrors(PX_XFC xfc);
void PxSetError(PX_XFC xfc, PX_CHAR* error);
void PxClearErrors(PX_XFC xfc);
```

PxHasErrors returns the error flag.

PxGetErrors returns the messages of all errors.

PxSetError sets the error flag and adds an error message.

-
- ❶ The method described here is the only way to pass errors to the ProductXpress Calculator. Other error mechanisms like C++ exceptions are not supported. Exiting an external function by means of a C++ exception will lead to termination of the Calculator process.
-

Java Interface

It is also possible to program External Functions in Java. For this purpose a special adaptor is provided consisting of a dll (or shared object file) and a Java jar file.

To use this adaptor, configure the dll PxXFC_jni.dll (or LibPxXFC_jni.so) file as the implementation in the ExternalFunctions.Settings file and add the following jar files to your CLASSPATH:

```
FiaJNI.jar
PxVal.jar
PxXFC.jar
```

The classpath can be set in the XFJava.Settings file:

```
<ClassPath value="your class path"/>
```

-
- ❶ Setting the classpath via the XFJava.Settings file will only work if the Calculator creates the Java virtual machine. This is not the case when using the Embedded Calculator in a Java Environment.

Make sure that the Java Runtime Environment is installed properly and that the jvm can be loaded by the Calculator; e.g. on Windows this requires the path '`<jre install dir>\bin\client`' to be part of the PATH environment variable.

The Java code should also load the PxXFC_jni.dll with System.loadLibrary function.

The External Function should be implemented in a static function of a class and should be configured using the following Properties:

ClassName	Name of the java class in which the external function is implemented.	Required e.g. my/package/MyClass
MethodName	Name of the static method that implements the external function.	Optional default is the name of the Function in ProductXpress

The static function that implements the External Function should have the following interface:

```
public static com.solcorp.productxpress.val.PxCalcValue
    myFunction(com.solcorp.productxpress.xfc.XFC xfc)
```

The class `com.solcorp.productxpress.val.PxCalcValue` is used for all data values in the External Function API and is described in detail in the section about the Embedded Calculator.

The class `com.solcorp.productxpress.xfc.XFC` is the External Function Context and has the following interface:

```
public String getFunctionName();
public String getProperty(String name);
public int    getArgCount();
public PxCalcValue getArg(int argno);
```

`getFunctionName` returns the name of the External Function within `ProductXpress`.

`getProperty` returns the value of the property with the given `name`.

`getArgCount` returns the number of arguments passed to the External Function.

`getArg` returns the value of the argument with the given `argno`. ($0 \leq \text{argno} < \text{getArgCount}()$)

Example:

```
import com.solcorp.productxpress.xfc.*;
import com.solcorp.productxpress.val.*;

public class MyClass
{
    static {
        System.loadLibrary("PxXFC_jni");
    }
    public static PxCalcValue myFunction(XFC xfc)
    {
        int argc = xfc.getArgCount();
        int result = 0;

        for (int i = 0; i < argc; ++i)
        {
            result +=
            xfc.getArg(i).getIntegerValue().intValue();
        }
        return PxCalcValue.createIntegerValue(result);
    }
}
```

❗ To return so called "null value" return Java null instead of a `PxCalcValue` object reference.

.NET Interface

On the Microsoft Windows platform, it is also possible to program the external functions for the .NET environment. For this purpose, a special adaptor is provided consisting of a dll.

To use this adaptor, configure the dll `PxXFC_net.dll` file as the implementation in the `ExternalFunctions.Settings`.

The External Function should be implemented in a function of a class that has a default constructor and should be configured using the following Properties:

ClassName	Name of the .NET class in which the external function is implemented.	Required e.g. my.Namespace.MyClass
-----------	-----------------------------------------------------------------------	------------------------------------

MethodName	Name of the method that implements the external function.	Optional default is the name of the Function in ProductXpress
Assembly	Name of the .NET assembly that contains the external function.	Required e.g. MyAssembly

The function that implements the External Function should have the following interface:

```
public ProductXpress.Val.PxCalcValue
    myFunction(ProductXpress.XFC.XFC xfc)
```

The class `ProductXpress.Val.PxCalcValue` (PxValNet.dll) is used for all data values in the External Function API and has the following interface:

```
public PxCalcValue();
public bool IsNull();
public PxType Type;
public int GetIntegerValue();
public decimal GetRealValue();
public string GetStringValue();
public bool GetBooleanValue();
public DateTime GetTimeValue();
public List<PxCalcValue> GetSetValue();
public int GetSetValueCount();
public PxCalcValue GetSetValueItem(int index);
public PxCalcValue GetTupleMemberValue(string
    tupleMemberName);
public PxCalcValue GetTupleMemberValue(PxTupleMember
    tupleMember);
public void AddTupleMember(string tupleMemberName, PxCalcValue
    tupleMemberValue);
public Dictionary<DateTime, PxCalcValue> GetValueHistory();
public PxValueHistoryEnumerator GetValueHistoryEnumerator();
public string GetUnit();
public string GetInstanceId();
public static PxCalcValue CreateIntegerValue(int value);
public static PxCalcValue CreateRealValue(decimal value);
public static PxCalcValue CreateStringValue(string value);
public static PxCalcValue CreateBooleanValue(bool value);
public static PxCalcValue CreateTimeValue(DateTime value);
public static PxCalcValue CreateQuantityValue(decimal value,
    string unit);
public static PxCalcValue CreateSetValue(PxType containedType,
    List<PxCalcValue> values);
public static PxCalcValue CreateTupleValue(PxType tupleType,
    List<PxCalcValue> tupleValues);
public static PxCalcValue CreateTupleValue(string tupleName);
public static PxCalcValue CreateTableIdValue(string value);
public static PxCalcValue CreateInstanceReferenceValue(string
    value);
public static PxCalcValue CreateTdIntegerValue(
    Dictionary<DateTime, PxCalcValue> valueHistory);
public static PxCalcValue
    CreateTdRealValue(Dictionary<DateTime, PxCalcValue>
```

```

        valueHistory);
    public static PxCalcValue CreateTdStringValue(
        Dictionary<DateTime, PxCalcValue> valueHistory);
    public static PxCalcValue CreateTdBooleanValue(
        Dictionary<DateTime, PxCalcValue> valueHistory);
    public static PxCalcValue CreateTdTimeValue(
        Dictionary<DateTime, PxCalcValue> valueHistory);
    public static PxCalcValue CreateTdQuantityValue(
        Dictionary<DateTime, PxCalcValue> valueHistory);
    public static PxCalcValue CreateTdSetValue(PxType
        containedType, Dictionary<DateTime, PxCalcValue>
        valueHistory);
    public static PxCalcValue CreateTdTableIdValue(
        Dictionary<DateTime, PxCalcValue> valueHistory);
    public override string ToString();

```

The class `ProductXPress.Val.PxType (PxValNet.dll)` is used to identify the type of value contained in a `PxCalcValue` object. It has the following interface:

```

    public static PxType IntegerType();
    public static PxType RealType();
    public static PxType StringType();
    public static PxType BooleanType();
    public static PxType TimeType();
    public static PxType QuantityType();
    public static PxType SetType(PxType containedType);
    public static PxType TupleType(string name,
        List<PxTupleMember> tupleMembers);
    public static PxType TableIdType();
    public static PxType InstanceReferenceType();
    public static PxType TdIntegerType();
    public static PxType TdRealType();
    public static PxType TdStringType();
    public static PxType TdBooleanType();
    public static PxType TdTimeType();
    public static PxType TdQuantityType();
    public static PxType TdSetType(PxType containedType);
    public static PxType TdTableIdType();
    public bool IsIntegerType();
    public bool IsRealType();
    public bool IsNumericType();
    public bool IsStringType();
    public bool IsBooleanType();
    public bool IsTimeType();
    public bool IsQuantityType();
    public bool IsSetType();
    public bool IsTupleType();
    public bool IsTableIdType();
    public bool IsInstanceReferenceType();
    public bool IsTdIntegerType();
    public bool IsTdRealType();
    public bool IsTdNumericType();
    public bool IsTdStringType();
    public bool IsTdBooleanType();
    public bool IsTdTimeType();
    public bool IsTdQuantityType();
    public bool IsTdSetType();
    public bool IsTdTupleType();

```

```

public bool IsTdTableIdType();
public bool IsTimeDependent();
public PxType ContainedType();
public string TupleTypeName();
public List<PxTupleMember> TupleMembers();
public PxTupleMember TupleMember(int index);
public int TupleMembersCount();
public override string ToString();

```

The class `ProductXPress.Val.PxTupleMember` (`PxValNet.dll`) is used to define a tuple member of a tuple type. It has the following interface:

```

public PxTupleMember(PxType type, string name, int index);
public PxType Type;
public string Name;
public int Index;

```

The class `ProductXPress.XFC.XFC` (`XFC_net.dll`) is the External Function Context and has the following interface:

```

public string GetFunctionName ();
public string GetProperty(string name);
public int GetArgCount();
public PxCalcValue GetArg(int argno);

```

`GetFunctionName` returns the name of the External Function within ProductXpress.

`GetProperty` returns the value of the property with the given name.

`GetArgCount` returns the number of arguments passed to the External Function.

`GetArg` returns the value of the argument with the given argno. ($0 \leq \text{argno} < \text{GetArgCount}()$)

Example:

```

using System;
using System.Runtime.InteropServices;
using System.Collections.Generic;
using ProductXPress.Val;
using ProductXPress.XFC;

namespace MyNameSpace{
    public class MyClass
    {
        public PxCalcValue myFunction(XFC xfc)
        {
            int argCount = xfc.GetArgCount();
            int result = 0;
            for(int i=0; i<argCount; i++)
            {
                result += (int)xfc.GetArg(i).GetIntegerValue();
            }
            return PxCalcValue.CreateIntegerValue(result);
        }
    }
}

```

i To return so called "null value" return .NET null instead of a `PxCalcValue` object reference.

Compatibility Mode

Prior to this release, a different interface with External Functions existed. External Functions written for this old interface can still be used when using the XFCompat dll or shared object file.

This XFCompat implements an External Function that will translate the new Arguments to old Arguments and the old Return Value to the new Return Value.

The extra info needed for these translations is passed through by the following Properties:

dll	the name of the dll or shared object file
entry-point	the name of the entry point within the dll
return-type	the data type of the return value
return-type- <i>n</i> (where <i>n</i> is a sequence number starting from 1)	data types for multiple return values.
max-result-length	the size of the output buffer
arg- <i>n</i> (where <i>n</i> is a sequence number starting from 1)	constant value for specified argument

Configuring External Functions

The ProductXpress Calculator uses the external function definitions from the ExternalFunctions.Settings file to call an external function; the ExternalFunctions.Settings file may be found and edited accordingly in the "C:\Program Files\Hewlett-Packard\ProductXpress\Calculator\Etc" directory of Calculator (where C is the operating system drive of your computer). XML is used in this file to define an external function.

The structure of this file is:

Element	Attribute Description		Child elements
ExternalFunctions			Function *
Function	name	Name of the function	Target* NOTE: multiple targets are only possible in a Designer context.
Target	name	Name of the target may be empty if only one target exists	Property*
	file	The name of the dll (Windows) or shared library (AIX, Solaris or Linux) that contains the external function NOTE: Make sure that the dll or shared library	

Element	Attribute Description		Child elements
		can be found by either specifying the complete path or setting the specific environment variable that is used by the operating system to locate a dll or shared library.	
	entry-point	The name of the function in the dll (Windows) or shared library (AIX, Solaris or Linux) that should be called. default value = "execute"	
Property	name	Name of the property	
	value	Value of the property	

* : zero or more

Example of an **ExternalFunctions.Settings** file content:

```
<ExternalFunctions>
  <Function name="Lookup1" >
    <Target file="MyExtFunc.dll" entry-point="lookup">
      <Property name="Database" value="MyDatabase"/>
      <Property name="Table" value="Table1"/>
    </Target>
  </Function>
  <Function name="Lookup2" >
    <Target file="MyExtFunc.dll" entry-point="lookup">
      <Property name="Database" value="MyDatabase"/>
      <Property name="Table" value="Table2"/>
    </Target>
  </Function>
  <Function name="oldFunction">
    <Target file="XFCompat"> <!-- compatibility mode -->
      <Property name="dll" value="MyOldDll.dll"/>
      <Property name="entry-point" value="MyOldFunction"/>
      <Property name="return-type" value="integer"/>
      <Property name="max-result-length" value="1024"/>
    </Target>
  </Function>
  <Function name="javaFunction">
    <Target file="PxXFC_jni"> <!--Java adaptor -->
      <Property name="ClassName"
value="my/package/MyClass"/>
      <Property name="MethodName" value="MyJavaFunction"/>
    </Target>
  </Function>
</ExternalFunctions>
```

Other Issues

This section describes various other issues with respect to External Functions.

Security

When implementing an External Function, be aware that this function will be executed in the process context of the ProductXpress Calculator. This means that when signals or exceptions like *access violations* occur in an External Function, this will terminate the Calculator process. It may also be possible to access and/or disturb the internal data structures of the Calculator.

Always use defensive programming practices and test well in a test environment in which crashes are not catastrophic.

Multi Threading

External functions must be implemented in a reentrant and multi-thread safe manner. This is necessary because the Calculator itself is a multi-threaded application and therefore can issue concurrent calls to an external function.

Immutability

External functions must be immutable. This means that they must return the same value for the same set of parameters every time they are called.

The calculator caches external function results. When an external function is used with the same parameters for the second time, the calculator will use the cached value.

Therefore when the data in the external system changes in such way that the External function should return a different value, the global caches of the calculator must be cleared (using the `clearGlobalCalculatorCache` API).

Performance/Scalability

For performance sensitive calculations using external functions, special care must be taken to ensure that overall response-time, or processing is not hampered by the external function call. A few guidelines to improve performance include:

- ▶ Use of index, and database tuning for database intense operations.
- ▶ Implement caching mechanism in the DLL (Microsoft Windows) or shared library (AIX, Solaris or Linux) such that requests for data on a non-ProductXpress Calculator platform are made less often.
- ▶ Use of a suitable middleware, or asynchronous call mechanism for scalability.

Common Mistakes

-
- ❶ Make sure that when using a C++ compiler to compile an external function, the function is declared as `extern "C"`. For details, refer to a C++ reference manual.

On the Microsoft Windows platform, make sure that external function is preceded with `__declspec(dllexport)`, to export this function from the DLL (Make it an externally visible function). For further details, refer to MSDN.

Common Configuration and Maintenance

This section provides common configuration and maintenance information.

Logging

The Calculator logs errors and important information.

Each logging entry has the following parts:

- ▶ Time information
- ▶ Host information: machine, process number and thread number of the Calculator
- ▶ Context information: the Method and Class in the code where the logging took place
- ▶ Logged message: the actual information logged

Internal log file

The Calculator creates an internal log file which receives all log data from the Calculator. This log file can be accessed through the Calculator Console. Note that this log file, just as any other log file, can become very large.

Logging on the Windows Platform

On the Windows platform, logging uses the Windows application event log with source SOLCORP ProductXpress Calculator.

The time information is displayed in the list of events. The host information is omitted. The context information is placed between brackets after the logged message.

Logging on the UNIX Platform

Log messages are written onto the system log maintained by the syslogd command.

Each entry is preceded by +++ and followed by ---- as in the following example:

```
++++ information 2003-06-02 09:26:06
      host = WS-1022 process = 00000720 thread = 000001e4
      context = Server::listen(), module FiaSrv
      Server CalcServer: listening on port 30001 for requests.
----
```

Logging when running Calculator Interactively

Logging is formatted as on the UNIX platform and shown in the Command Window where the Calculator is running.

Tracing

The actions performed by the Calculator can be logged using the tracing mechanism.

Tracing can be configured by specifying the Trace Setting with a Trace Spec. Tracing output can be redirected through the TraceFile setting. When the TraceFile setting is used an existing trace file from the previous run is backed-up, this will overwrite an existing old backup.

-
- ❶ In order to achieve optimal performance in your live production environment, we strongly recommend that tracing is turned off.
-

Trace Spec

The Trace Spec has the following format:

<module-name>:<class-name>:<function-name>

If the function-name is omitted, traces issued within all functions of the specified class will be generated.

If the class-name is also omitted, traces issued within all classes of that module will be generated.

Multiple trace specs should be separated by commas.

For example:

```
<Trace value="Clc,Ctx:ExternalFunctionTarget,
Conn:ConnectorRequestMethod:prepare" />
```

This trace spec traces the Clc library, the ExternalFunctionTarget class of the Ctx library and the prepare method of the ConnectorRequestMethod class of the Conn library.

Tracing Request and Result

To trace the request and the result, use the following trace spec:

```
Conn:ConnectorRequestMethod
```

To trace only the request as received by the Calculator, use the following trace spec:

```
Conn:ConnectorRequestMethod:prepare
```

To trace only the result as returned by the Calculator, use the following trace spec:

```
Conn:ConnectorRequestMethod:reply
```

Tracing External functions

Use the following trace spec to view what is sent and received:

```
ExtFunction,Ctx:ExternalFunctionCollector
```

Tracing Embedded Calculator JNI interface

Use the following trace spec to view all interface calls for the JNI interface of the Embedded Calculator:

```
CalculatorJNI
```

Tracing Embedded Calculator C interface

Use the following trace spec to view all interface calls for the C interface of the Embedded Calculator:

```
Calculator
```

Tuning

Cache

In order to improve speed, the Calculator caches calculated values of Basic Expressions in order to reuse them in subsequent calculations.

The Calculator interface or the Calculator.Settings configuration file allows the configuration of the size of the caches in order to prevent them from becoming so big that they would exceed the available memory.

i If the cache becomes too large, performance will degrade because of disk swapping.

Due to the fact that some calculated values have little chance of being reused within calculations, in other policies, the following two cache types exist:

- ▶ The Policy cache. Each request (or subrequest, see note below) has its own Policy cache, which is used within a single call of a single client. It contains results of expressions that contain variables, which are Policy and date dependent, and therefore are not suitable for reuse by different clients. The size of a Policy cache is checked each time a value is added. The values that have the least reuse are removed when the size exceeds the configured maximum. A Policy Cache is emptied after each policy calculation.
- ▶ The Global cache is used by all clients. It contains results of all other expressions.

The size of the Global cache is checked at a configurable interval. The values, which are removed when the cache exceeds the configurable maximum, are determined by the reuse count and how recently a value has been reused (the ProductCacheAgingFactor setting). This last criterion prevents the values, which were reused in the first few calculations and never again afterwards, from remaining endlessly in the cache.

i A request containing scenarios is split within Calculator into subrequests, with one subrequest for each scenario. Each subrequest has its own policy cache.

A batch request is split within Calculator into requests. Each of these requests has its own policy cache (Or policy caches if the request contains scenarios).

Redistribution of Calculator Components

This section provides information regarding the redistribution of Calculator components.

Windows Installer Merge Modules

For easy and correct redistribution of the necessary components needed for integrating the Calculator into ISV (Independent Software Vendor) developed applications, Windows Installer merge modules are provided. During installation, these merge modules will be installed in the MergeModules subdirectory.

Most applications used for developing setup applications for Windows platforms include a way to use merge modules, and merge the information contained in such module into the main setup project during compilation.

The following merge modules, which are described below, are installed in the MergeModules subdirectory under the root of the Calculator installation directory.

- ▶ CalcServerCOM.msm
- ▶ CalculatorC.msm
- ▶ CalculatorCOM.msm
- ▶ CalculatorJNI.msm
- ▶ CalculatorCore.msm

When using these modules, it may be necessary to copy them to a specific location, if required by your setup development application.

When using the merge module(s) in your own setup project, the contents of the merge modules will always be installed in certain subdirectories of your installation root directory. Some examples of such subdirectories are \Bin and \Etc.

When using these merge modules, no configuration files will be stored in the \Etc subdirectory. Templates for all configuration files are installed in the \Etc\Templates directory.

Calculator COM Interface Components

The components necessary for the integration with the Server Calculator via COM are available in the CalcServerCOM.msm merge module. Include this merge module in your setup project if your application uses the Calculator COM interface.

Embedded Calculator for C Components

When using C integration in your application, you can include the CalculatorC.msm merge module in your setup project. This merge module contains the files and logic necessary to install an embedded Calculator for C integration on the system, with its default settings.

Embedded Calculator for Microsoft .NET Components

When using Microsoft .NET integration in your application, you can include the CalculatorCOM.msm merge module in your setup project. This merge module contains the files and logic necessary to install an embedded Calculator for .NET integration on the system, with its default settings.

Embedded Calculator for Java components

When using Java integration in your application, you can include the CalculatorJNI.msm merge module in your setup project. This merge module contains the files and logic necessary to install an embedded Calculator for Java Native Interface integration on the system, with its default settings.

Embedded Calculator Core Components

Always include the CalculatorCore.msm in your project, as this merge module is a dependency for all merge modules mentioned above

i These redistributables can only be used to install on Windows 2000 platforms SP4 or higher.

Troubleshooting Guide

This section provides troubleshooting information.

Troubleshooting Steps

The actions performed by the Calculator can be logged using the tracing mechanism.

The tracing mechanism can be turned on in order to trace specific modules in which errors may be reported.

See the Tracing (on page [277](#)) section for explanation on how to turn on specific tracing options. Tracing information may be viewed more easily by running the Calculator interactively.

When troubleshooting check:

- ▶ Is the Calculator running?
Check if the CalcServer.exe (or CalcServer for AIX, Solarisplatforms) process is running.
- ▶ Does the standard test succeed?
See the *ProductXpress Calculator User Guide*.
- ▶ Has the Calculator reported any errors?
Check the Logging output of the Calculator, see section Logging (on page [277](#)).

- ▶ If the Calculator shows no sign of receiving a request that was sent.
Was a complete request sent? Otherwise the Calculator will do nothing but wait for the rest of the request to be sent.
- ▶ If the Calculator closes connection after sending result of request, or when sending the next request.
Did the sent request have extra characters after the last xml element?
If so, these will be regarded as beginning of the header of a new request, or as the complete header of a new request. The Calculator will not recognize the header, log an error, and close the connection.
- ▶ Has the requested deployment been loaded by the Calculator?
Check the logging output of the Calculator.
- ▶ Is the request received correctly by the Calculator?
Trace the Request received by the Calculator.
- ▶ Which result is sent by the Calculator?
Trace the result sent by the Calculator.
- ▶ What is the Calculator doing (only on AIX platform)?
The signal SIGUSR1 (on AIX platform this has value 30) can be sent to the Calculator with the kill command (example: kill -30 <pid calculator>) the Calculator will print statistics about the number of worker threads, number of idle worker threads and the number of requests queued.

Running Calculator Interactively

When run in interactive mode, the Calculator displays its logging information on the command screen where the Calculator is started, which on a Windows platform is easier to view than the event viewer.

The Calculator can be run interactively by entering the command in a Command Window in the <InstallDir>\Bin directory.

```
CalcServer -interactive
```

-
- i** In the Windows command window used to start the Calculator, make sure word wrap is turned on, otherwise logging information may be lost by truncation.
-

Appendix A - Examples for the Service Calculator

The Examples for the Push Calculator can be found in an external document, *ProductXpress Calculator Example Code for the Service Calculator*. This can be found in the Documentation Folder of the ProductXpress CD or can be found in <CalInstallDir>\Documentation.

Appendix B - Specification of Calculator Input-Output XML

The Specification of the Calculator Input and Output can be found here in the *ProductXpress Calculator Reference Guide*.

Appendix C - Examples for the Embedded Calculator

The Examples for the Embedded Calculator can be found in an external document, *ProductXpress Calculator Example Code for the Embedded Calculator*. This can be found in the Documentation Folder of the of the ProductXpress CD or can be found in <CalcInstallDir>\Documentation.

Appendix D - Transition Matrix

The Transition Matrix can be found in an external document, *ProductXpress Path Language Reference Guide*. This can be found in the Documentation Folder of the of the ProductXpress CD or can be found in <CalInstallDir>\DocumentationReferences.

Appendix E - W3C XML Schema Datatypes

Paul V. Biron, Ashok Malhotra, editors. XML Schema Part 2: Datatypes
(<http://www.w3.org/TR/xmlschema-2/>). W3C (World Wide Web Consortium), 2001.