ProductXpress

User Guide

ProductXpress Calculator v3.7

April 2013

Table of Contents

About This Guide

Read this section before progressing to the remainder of the document. It contains important information describing the *Calculator User Guide* content and structure.

The goal of this guide is to serve as a tutorial to perform all the activities necessary for a successful operation of ProductXpress Calculator within a complex architecture.

This guide is intended as an introduction to the complete breadth of using and implementing the ProductXpress Calculator. It touches on the most common actions required but does not go into detail, since these are specified in other documentation. After each section, a reference to another document (usually the *Calculator Reference Guide*) is given. Before completing any of the described actions, it is advised that the corresponding reference is also read and understood.

Audience

This guide is meant for:

- System Designers, to help them decide how to integrate the Calculator with their architecture
- System engineers, to explain how to install and maintain the Calculator application
- Software engineers, to explain how to integrate the Calculator with other systems, and how to test this integration

Terminology

In this guide the following terminology is used:

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

Contents

This Guide starts with an overview of the Runtime Architecture of ProductXpress and its key features. It describes the applications ProductXpress consists of and addresses architectural decisions that have to be made when setting up ProductXpress Calculator.

The subsequent sections of this guide describe the key activities needed for a successful deployment of the ProductXpress Calculator.

The order of these sections is more or less chronological. However as each project is unique, the actual order might vary, or activities might be revisited due to the development cycle.

- ProductXpress Runtime Architecture - provides an overview of the applications ProductXpress consists of and how they work together. It also covers the design decisions that need to be made during the integration project.
- Product Development and Testing - covers the configuration of the Calculator and the possibilities to run standalone tests on the Calculator.
- Integration – is spread over three sections and describes the generic steps of integrating into the deployment environment as well as specific steps for the Embedded and the Service Calculator. Combined, these sections cover the integration software and end-to-end testing.

- Developing External Functions describes how to develop and use external functions.
- Developing External Tables describes how to develop and maintain external table data.
- Operational Management - covers the maintenance issues.
- Performance Optimization - covers the measures to analyze and improve the performance of the Calculator.
- Appendix A - This appendix provides handy tips to help improve performance.

Related Documents

Calculator Console Online Help

ProductXpress Calculator Installation Guide

ProductXpress Calculator Reference Guide

ProductXpress Designer Online Help

ProductXpress Runtime Architecture

ProductXpress is a fully integrated software suite comprised of a Designer and a Calculator that assists organizations to dramatically reduce time to market and overall product development, implementation and maintenance costs.

HP ProductXpress Designer (Designer): Designer offers product design professionals an innovative tool for facilitating rapid product design of (complex) rules and calculations. Based on a building block approach, the Designer applies a "build it once, use it often" philosophy toward product development. With advanced calculation and archive capabilities, this tool lets you build many product prototypes rapidly, provides smooth product implementation and system testing, and shortens the time to market.

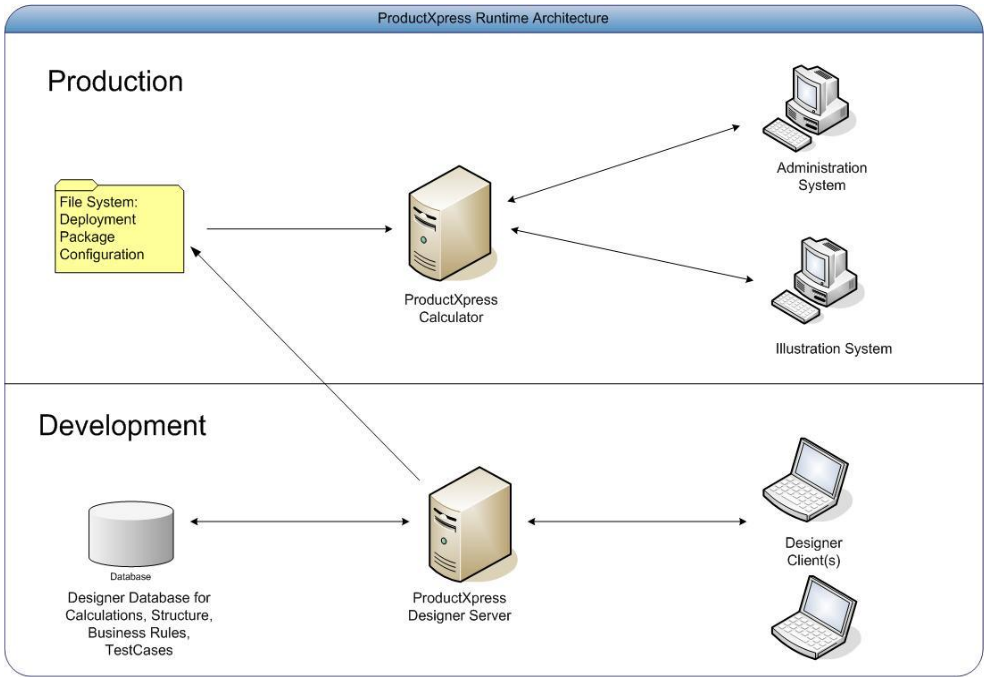
Designer is positioned in the development environment. In this environment you design and test product rules and calculations. Once products have been approved, they can become part of the production environment of the company. The production environment consists of additional software applications, such as policy administration, relations administration, and claims administration.

HP ProductXpress Calculator (Calculator): Calculator is a powerful mathematical calculator that can be executed from multiple production systems - ensuring exactly the same calculation results in any place the calculation is required. A multi-functional deployment of Calculator creates a single point of truth. Calculations are performed centrally and universally, so that they always have identical results. Calculator combines the highest level of consistency with superb performance and scalability. As well as a standalone product, it is also embedded within the ProductXpress product range and is used to perform calculations in Designer, RADIENCE Administration and INGENIUM. This ensures that calculations have exactly the same results, whether they are made during the product development process, or within the illustration or administration systems.

As well as a standalone product, it is also embedded within the ProductXpress product range and is used to perform calculations in Designer. This ensures that calculations have exactly the same results, whether they are made during the product development process, or within the illustration or administration systems like RADIENCE Administration and INGENIUM, which are also integrated with Calculator.

The Calculator can either be used as a standalone service (this is referred to as the S*ervice Calculator*) or it can be embedded into a third party application.

The following graphic shows an overview of the applications in the ProductXpress suite during development and production. In this diagram, a Service Calculator is used by two client applications (an Administration System and an Illustration System).


ProductXpress Runtime Architecture
Production
File System: Deployment Package Configuration
ProductXpress Calculator
Administration System
Illustration System
Development
Database
Designer Database for Calculations, Structure, Business Rules, TestCases
ProductXpress Designer Server
Designer Client(s)

Key Features and Integration Considerations

The ProductXpress Calculator application is used by or is integrated into client applications for performing calculations and to evaluate business rules.

When making architectural decisions as to how the Calculator should be integrated into the target environment, it is important that the key features are understood and considered. Questions such as, "On which platform should Calculator be installed? Will calculations be sent in batches to the Calculator and will the calculations be performed in parallel?" should be satisfied before implementation begins. The following is an introduction to the key features of the Calculator – all of which are discussed in greater detail in this guide and the *Calculator Reference Guide*.

Platform

The Calculator can run on a number of different platforms (Windows and UNIX). It does not require a database management system.

There are various considerations to make when choosing a platform for the Calculator including costs and calculation speed of the underlying architecture (the Calculator performs best when running on a system that is optimized for calculation speed).

Calculator Types and Communication Mechanisms

The way in which a client application such as an illustration system communicates with the Calculator depends on the decision whether Calculator will be used as a standalone service ('Service Calculator') or will be embedded into the client application ('Embedded Calculator').

Service Calculator

Client applications of the Service Calculator communicate with it using TCP/IP requests over sockets, using either a native format or using the SOAP standard.

The contents of this request consist of an XML document containing the requested rules/calculations and the data on which these rules/calculations should be executed. This method of data transfer is also referred to as a 'push model'.

To simplify interfacing with the Service Calculator, the CalcClient library (CalcClient.dll on Windows platform, libCalcClient.so on UNIX platforms) is provided. This library uses a simple character, string-based interface. As a result, no low level socket programming is required for the client application. It also implements other features like asynchronous calculations.

On the Windows platform, the Calculator can also be called through a COM interface.

A single Calculator application can handle simultaneous requests of multiple client applications that use different communication mechanisms.

For details on the protocols used by the Service Calculator, refer to the *Calculator Reference Guide*.

Embedded Calculator

The integration with the Embedded Calculator can be done by using one of the following interfaces: .NET, Java or C.

The Embedded Calculator supports both the 'push model' (see Service Calculator (on page 5)) and the 'pull model'. In the pull model, only the calculation requests are sent to the Calculator. The necessary inputs to calculate the requested calculations are obtained by Calculator from the application in which it is embedded through a call-back mechanism.

Function Calculator

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

Scalability

If large batches of calculations need to be performed by Calculator, extra hardware in the form of multiple CPUs or multiple machines may be a valuable and affordable option to ensure optimum performance. The large batch can then be split up in multiple smaller batches that can be processed by multiple pieces of hardware.

Multiple CPUs

When running on multiple CPU machines, concurrent calculation requests to Calculator will be distributed across multiple execution threads. Each calculation request will be handled by one execution thread and thus allows for parallel processing.

Multiple Machines

Calculator can run on multiple machines under cluster management software. If each machine is configured identically, calculation requests can be spread across these machines. Since information is not shared between multiple Calculators, the scaling is virtually linear (i.e. if twice as many machines are used, performance will be twice as fast).

An example of such cluster management software is ‘IBM SecureWay Network Dispatcher’ that sits in front of a cluster of Calculators each running on a Windows or UNIX server.

Aggregation

When calculating across large volumes of data, for example to calculate averages across a portfolio of assets or policies, embedded values, etc., a calculation can be split into multiple smaller requests and distributed across multiple CPUs and machines using the ProductXpress aggregation mechanism.

Multiple Instances

It is possible to install more than one instance of Calculator on a single machine. This capability is particularly useful in large scale system development where the development, test and acceptance environments may each require their own Calculator; each with its own set of run-time configuration and specific set of loaded products. These instances can be monitored from both a local and a remote Calculator Console. Management can be done from the Calculator Console.

ⓘ Changes to settings files, startup/shutdown, and CSV external tables files would have to be managed by the local Calculator Console.

When installing multiple Calculator instances on a single Windows Server, all the Calculator instances will share the hardware resources. It is recommended that only one Calculator instance is installed on Production system Windows servers. This will provide the best access to available machine resources without conflict with other applications, therefore deriving optimal response time. If multiple production instances of Calculator are installed on a single Windows server, the available hardware resources must be sufficient for the peak workload.

Precision and Rounding

Calculator allows the configuration of the precision accuracy and rounding method used in calculations to suit individual needs. Both are controlled by specific settings in the Calculator.Settings file.

Application Failover

Calculator does not support failover directly. It uses the application management capabilities of the operating system for failover support.

Calculator runs as a service on various UNIX and Windows platforms. These platforms can easily be custom configured for error recovery actions like restarting the service, running a batch file that restarts the Calculator on another machine, and so on.

External Functions and External Tables

Calculator provides facilities to invoke External Functions and use their results in any calculation. External functions are functions that contain logic to access data that resides on external systems. They are implemented as an entry point in a Dynamic Link Library (DLL) on the Microsoft Windows platform, or as an entry point in a Shared Library on UNIX platforms.

Another way to gather external data from calculations is through the use of External Tables. External Tables are referenced from within calculations but are maintained outside of the ProductXpress product definitions.

Carefully consider upfront whether external data sources are going to be used. The sections on External Functions and External Tables in this guide provide guidance as to when to make use of these.

Calculator Console

The Calculator Console can be run under both a UNIX and a Windows environment. It can be used to monitor the internal activity of a running Calculator through a number of performance indicators and to manage deployment packages and external tables of the local and remote Calculator.

The Calculator Console provides the functionality to:

- Add one or more Calculators to monitor
- Connect to a selected Calculator
- View the different performance indicators of the selected Calculator in a graphical and numerical way
- Configure the Calculator settings file
- View the Log file of a Calculator
- Manage and view Deployment Packages
- Manage and view External Tables
- Manage Encryption
- Manage User Authorization

Calculator Web Portal

The Calculator Web Portal hosts all Calculator-related information including end-user guides and samples of integration code so that all collateral is easily accessible to local and remote stakeholders. This also includes the Calculator Web Console, with all of the capabilities of the "Calculator Console" for operational management, testing and other related operations directly from the portal.

For further details, see Calculator Web Portal (on page 41).

SNMP Support

It is possible to monitor the Service Calculator via any SNMP enabled management tool. For this purpose, the Calculator contains an embedded SNMP Server, which is disabled by default. Please refer to the *Calculator Reference Guide* for details.

Publishing

The publishing infrastructure of ProductXpress allows customers to easily design and produce high-quality documents concerning products that they have created. Amongst others, it allows the publication of illustration documents (both inforce and new business), and policies. In the Operational Management (on page 49) section, the publishing process is described in more detail.

Deployment Overview

The ProductXpress Calculator is a generic calculation engine that uses product definitions that were defined in a generic definition language. It is designed to integrate with any type of client application.

The integration code to the Calculator from the client application is thus independent of the product definition, and the product definition is independent of the client application. This means that a product (with its associated rules and calculations) only needs to be created once, and a client application only needs to be integrated with the Calculator once. This is achieved though the concept of the "Deployment Environment".

When no product structure is needed in the client application, *functions* that are developed in Designer can be made available without such a structure to the application through function deployments.

Deployment Environment

Since each client application will have its own logical (and physical) model in which policies are stored and processed, there needs to be a method of bridging the generic model of the ProductXpress Product Definition language to the specific model of each client application with which the ProductXpress Calculator is to be integrated.

For this purpose, ProductXpress Designer provides the ability to model the logical structure of a client application. In ProductXpress, such a client application model is called a deployment environment. In the deployment environment, the model of the client application is defined in such a way that the ProductXpress Calculator can understand it.

Deployment

When a product is to be deployed into a client application (deployment environment), a mapping needs to be defined between this product and the deployment environment model of this application. This is called a deployment. A deployment defines how the structure of the product relates to the structure of the client application.

A single product can be deployed into many deployment environments (for example, both an illustration system and a policy administration system). There needs to be a deployment for each combination of product and target deployment environment.

ⓘ The process of configuring the Calculator with a product is also referred to as "deployment of a product" to the Calculator.

Function Deployment

When no product structure is needed, *functions* that are developed in Designer can be deployed separately without being part of a product structure. This is referred to as a function deployment. A function deployment consists of a list of *functions*.

Connectors

In order to integrate the ProductXpress Calculator with a production system, a layer of integration code (a connector) needs to be developed. This integration layer needs to gather all the policy information from the production system, which is needed for calculations, and pass it on to the Calculator. The structure in which this data is to be passed to the Calculator is the structure as defined in the deployment environment.

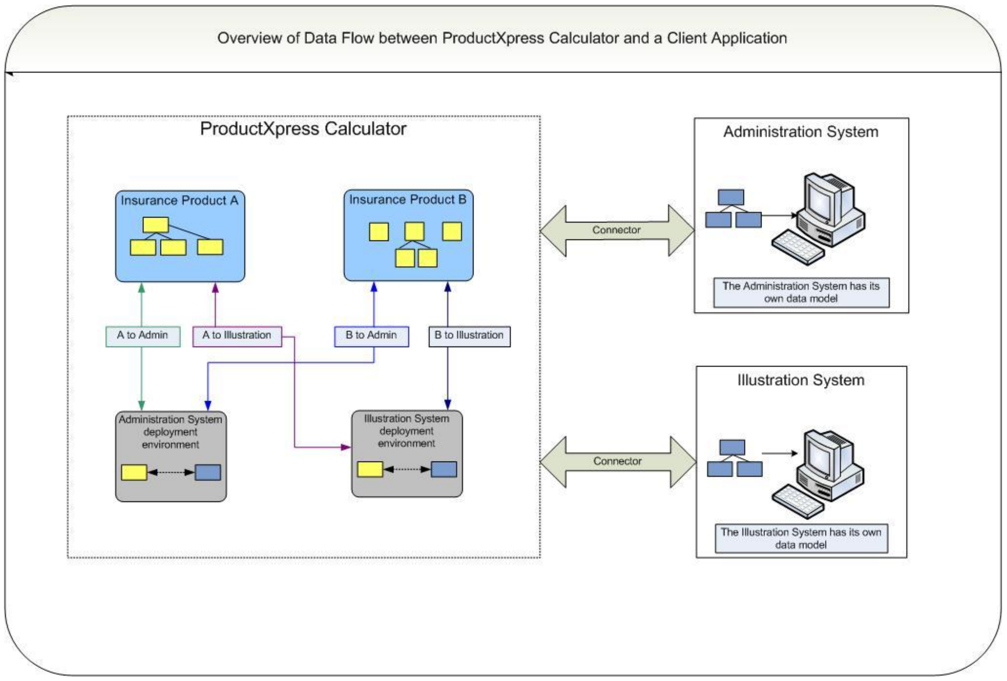
When the Calculator receives this data, it uses the deployment mappings to translate the data into the structure that is required by the calculation (i.e. the structure of the product that is being calculated).

The Calculator passes the results back to the client application in the structure that is defined in the deployment environment.

With function deployments, data (e.g. variable and parameter values) needs to be provided as well in order for Calculator to calculate the requested functions. This data however has a much simpler and looser structure in comparison with a deployment environment or product structure.

ProductXpress Runtime Architecture Example

The following diagram provides an example of the way in which data flows from the client applications to the ProductXpress Calculator, and the way in which this data is transformed into a ProductXpress product definition.


Overview of Data Flow between ProductXpress Calculator and a Client Application
ProductXpress Calculator
Insurance Product A
Insurance Product B
A to Admin
A to Illustration
B to Admin
B to Illustration
Administration System deployment environment
Illustration System deployment environment
Connector
Connector
Administration System
The Administration System has its own data model
Illustration System
The Illustration System has its own data model

The diagram shows two product definitions: Product A and Product B. These two products are used from two client applications: An administration system and an illustration system.

The two products are deployed into two deployment environments: Each one into the administration system and each one in to the illustration system. This results in four deployments – A to Admin, B to Admin, A to illustration, B to Illustration - one for each combination of deployment environment and product definition. The administration system and the illustration system each have their own connector, which sends data to the ProductXpress Calculator and receives results that can be used in the client application.

This means that Product A and Product B are only defined once (instead of once for each client application, as is common in traditional systems). In addition, the connector for the illustration system and the connector for the administration system are shared between both Product A and Product B, so the need to write integration code is minimized.

Reference

For more information about defining deployment environments, performing deployment mappings and creating function deployments, refer to the online help of ProductXpress Designer.

Product Development and Testing

This section describes the tasks involved from taking a product that has been designed in Designer to testing it in Calculator.

The steps involved are:

1. Create a deployment package.
2. Deploy the Deployment Package in Calculator.
3. Create Calculation Requests.
4. Test Calculator using PXTest.exe
5. Encrypt the deployment package, if needed.

Each of the steps is defined in the remainder of this section.

Creating a Deployment Package in Designer

Once a deployment is created for a product or a set of functions, a deployment package (also referred to as package) needs to be created from it so that it can be loaded into Calculator. The deployment package for a product is an XML file, which contains all the definitions of the product (such as the calculations and structure) that are defined in Designer. The Calculator needs this information in order to process a calculation request. This is why the Calculator is also referred to as a data and model driven application.

The deployment package of a function deployment consists of a list of functions. It also defines which (function) variables are needed to calculate the functions.

To create a deployment package in Designer:

1. Within Designer, select the deployment and invoke the *Deployment Package* menu item.

   A dialog will appear.
2. Select the location where the package will be saved.

Considerations when Creating a Deployment Package in Designer

This section provides information when creating a deployment package in Designer.

Encryption

The content of the package is XML and hence readable. Therefore, the package that is created can be encrypted to ensure that the contents of the package cannot be read.

During the creation of the package, in the dialog mentioned in the previous section, the following encryption options are available: *No encryption* (default), *Standard level encryption* or a *High level encryption.*

The Standard level encryption encrypts the package with a key that is built into the software. The advantage is that any Calculator installation can read these encrypted packages. The disadvantage is that it is less secure than the High level encryption.

The High level encryption will encrypt the package with a custom key. Custom keys can be generated from within Designer through the ENCRYPTION KEY MAINTENANCE menu item that can be found in the TOOLS menu. A generated custom key must also be used for decryption at the Calculator site where the package is to be loaded.

Activation Date

The activation date of a package is the starting date on which the package is active. Requests that are sent before this date will result in an error.

The activation date can also be used to distinguish between multiple versions of a package (thus product) in the Calculator. A version of a package is inactive before its own activation date and after the activation date of another version of the package. Depending on the activation date that is selected in the Calculator request, the package that is active on that date will be used to calculate the request.

The activation date can be set during the creation of the package in the *Deployment Package* dialog. It acts as the default value for this date. During the actual deployment, this date can be overridden.

Expiration Date

Along with the activation date, another method to make a package invalid is the expiration date.

The expiration date of a package is the date on which the package expires. Requests sent after this date will result in an error.

The expiration date can only be set during the creation of the package in the *Deployment Package* dialog. This date cannot be overridden during the actual deployment of the package.

Dynamic versus Static Deployments

Depending on the target system to which a product is going to be deployed, the choice must be made between a static and a dynamic deployment. A static deployment is used when the target system already has a fixed (data) model in which products are administered. Typically, when integrating with legacy systems this will be the case. A dynamic deployment will be used when the target system does not have a model in which the products are administered. Typically, this is the case when integrating with illustration systems.

The Embedded Calculator supports dynamic and static deployments.

Function Deployments

When there is no product structure needed, it is possible to only deploy a set of functions. Typically, this will be the case in Web quotation systems or lightweight illustration systems where no complex data structures (e.g. sets of sets) are needed in the calculations.

Function deployments can only be used in a push-model, in both the Embedded and the Service Calculator.

Reference

- *ProductXpress Designer Online Help*, section Export a Deployment Package.

Deploying a Deployment Package in Calculator

As the Calculator is data driven, it needs the definition of a product, such as its calculations and data structure, in order to process calculation requests for it. As a deployment package contains all this and other necessary information that is defined in Designer, it needs to be loaded in Calculator, so that Calculator can perform calculations.

The loading of packages differs between a Service Calculator and an Embedded Calculator. This is described in the following section.

Deploying a Package in the Embedded Calculator

The Embedded Calculator exposes a function called `loadDeploymentPackage` with which packages can be loaded into memory. See the *Calculator Reference Guide* for details.

Deploying a Package in the Service Calculator

A package can be loaded in the service Calculator via a command line utility, called `DepImp`, or via the Calculator Console.

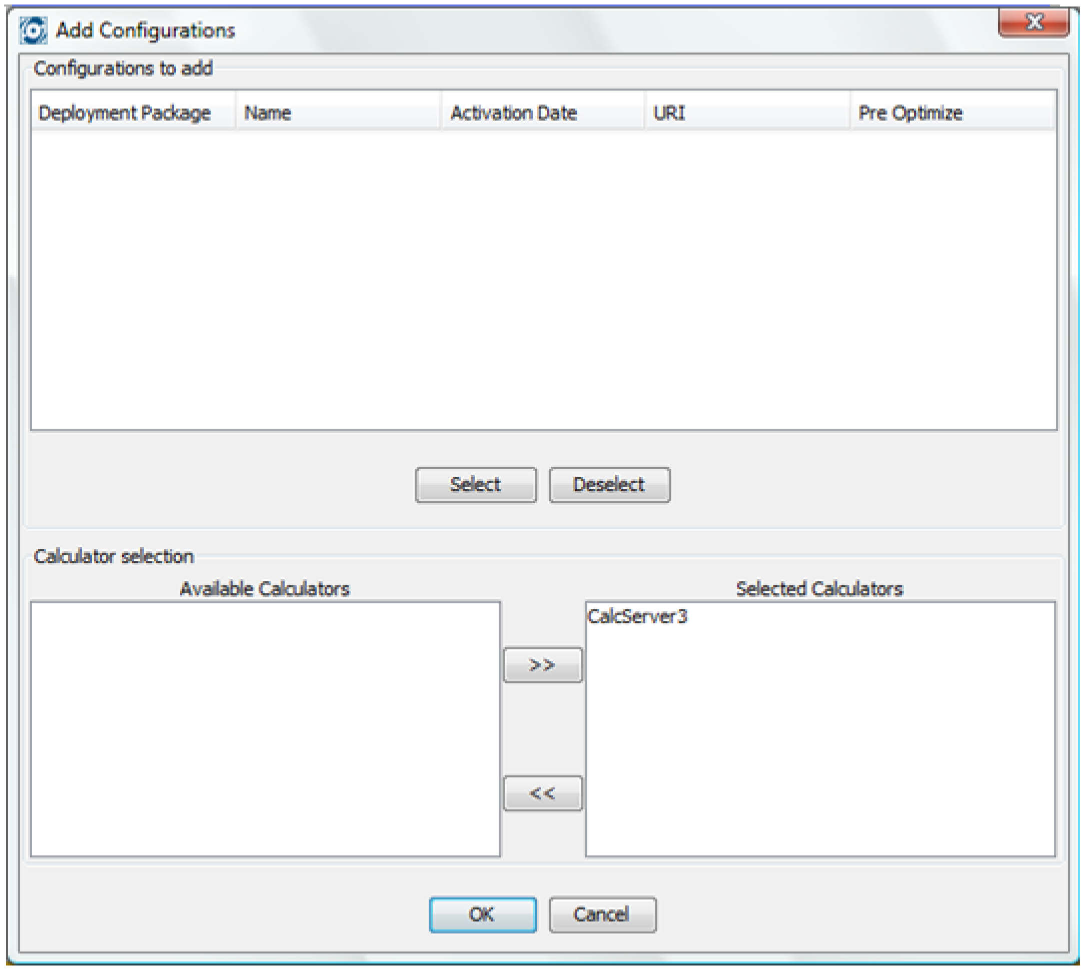
Command Line

To deploy a package using the command line, the utility `DepImp` is used, which is installed during the Calculator installation. Usage:

```
DepImp -add=<deployment package file>
```

Calculator Console

To deploy a package in the Calculator Console, the menu item ADD is selected from the CONFIGURATION menu.



Considerations when Deploying a Deployment Package in Calculator

This section provides information when deploying a deployment package in Calculator.

DepImp Usage with Multiple Instances on Windows platforms

If you operate more than one instance of Calculator, and you use the `DepImp` command line utility, you should be aware that this utility will work on the default instance, if called from a random directory on your system.

This is caused by the installation directory of the default instance being part of the system path. In order to use `DepImp` on a specific instance, call the utility from the <InstallDir>\Bin directory for this specific instance, or use the Calculator Console to deploy a package for a specific instance.

Checked out deployments

In order to load checked out deployments in the service Calculator, it must run in *Test Mode*. This mode can be set either at the command line or through the Calculator Console.

Command Line

This is achieved by adding the line:

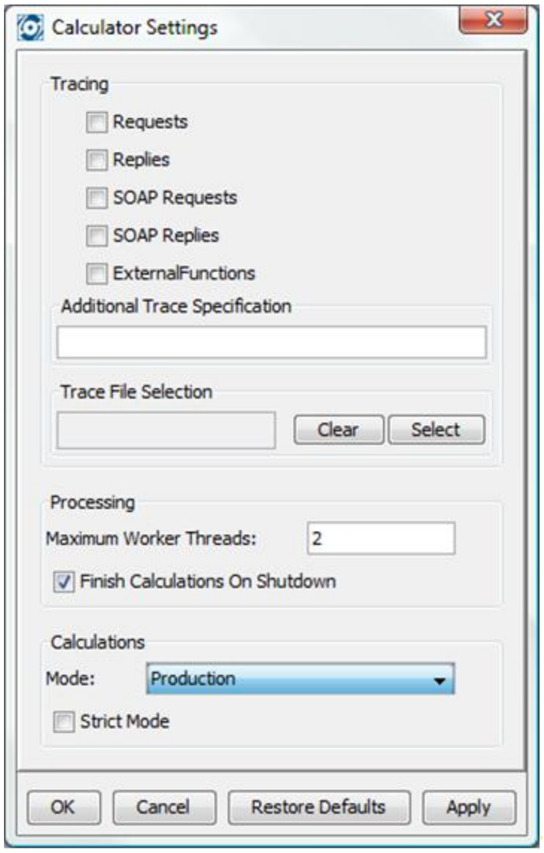
```
<Mode value="test">
```

in `Calculator.Settings` that can be found in `<Calculator Installation Dir>/Etc`.

ⓘ The Embedded Calculator does not support this mode.

Calculator Console

The settings file can optionally be changed through the Calculator Console by switching the Mode between 'Production' and 'Test' after selecting SETTINGS from the CALCULATOR menu.


Calculator Settings

Tracing
- [ ] Requests
- [ ] Replies
- [ ] SOAP Requests
- [ ] SOAP Replies
- [ ] ExternalFunctions

Additional Trace Specification

Trace File Selection

Processing
Maximum Worker Threads: 2
- [x] Finish Calculations On Shutdown

Calculations
Mode: Production
- [ ] Strict Mode

Overriding default activation date in the Service Calculator

The default activation date is the date that is given during the creation of a package. This date can be overridden during the loading of a package in Calculator. This can be done as follows:

Command Line

Use `DepImp` with the `–redate` option

Calculator Console

Specify a date when the package is added in the Calculator Console when adding the deployment. Or, select the deployment under the *Deployment* tab and reenter a new date.

Overriding default activation date in the Embedded Calculator

Pass a date to the `date` parameter of the function `loadDeploymentPackage`.

Expiration Date

When a package is expired due to its expiration date, it cannot be loaded in Calculator after this date.

Unlike the activation date, the expiration date of a package cannot be changed during the load process. The package would have to be recreated with a different expiration date.

Renaming the Deployment in the Service Calculator

The default name of a deployment is specified within Designer. This can be changed during or after the loading of the package in Calculator as follows:

Command Line

- Use `DepImp` with the `–name` option together with the `–add` option.
- Use the `–rename` option if the package is already loaded in Calculator.

Calculator Console

- Specify a name during the addition of the package in the Calculator Console.
- Change the name in the *Runtime Configuration* tab of the Calculator Console if the package is already loaded by selecting the deployment and typing in the new name.

Renaming the Deployment in the Embedded Calculator

Pass a name to the `name` parameter of the function `loadDeploymentPackage`.

Versions and Branches

When one or more versions of a package already exist in Calculator and another version is to be added, either the older versions need to be removed or the version to be added should get a different activation date than the existing versions.

When the new package is created in a branch in Designer and there is a package present in the Service Calculator with the same branch name but from a different repository, the branch name can be renamed during the deployment of the new package to solve naming conflicts. This can be accomplished by using `DepImp` with the option `–branchRename` together with the `–add` option.

ⓘ This is not possible in the Embedded Calculator. It is also not possible through the Calculator Console for the Service Calculator.

DepImp Options

Using 'DepImp' at the command line is not only for adding or renaming deployments. DepImp can also be used for other options such as listing the deployments and renaming the deployment. For a list of the possibilities, type the following on the command line:

```
DepImp  -help
```

Each option is explained in the *Calculator Reference Guide*.

Reference

- *Calculator Console Online Help* – Deployment folders
- *Calculator Reference Guide* - Service Calculator Configuration and Maintenance, sections Deployment Package Management and Test Mode of Calculator
- *Calculator Reference Guide* - Embedded Calculator Configuration and Maintenance, section Interfacing the Embedded Calculator

Creating Calculation Requests

The Service Calculator communicates with third party applications via XML. It receives XML with requested calculations and data necessary to calculate the requested calculations and returns the calculated results back in XML to the calling application.

Irrespective of the status of the actual Calculator integration with the production system, independent tests can be performed for a deployment package utilizing the test tool `PxTest`, which is delivered with Calculator. This tool simulates the client or production system to invoke the Calculator with a request.

This section describes how calculation requests can be created, which are sent to a Service Calculator using the tool `PxTest`.

ⓘ PxTest cannot be used with the Embedded Calculator.

Create Calculation Input Manually

It is possible to create a Calculator Input file manually. It must comply with the syntax required by the Calculator and thus must contain the following:

- The selection of the deployment for which a calculation needs to be performed.
- The data that is needed to perform the requested calculations.
- The requested calculations that need to be calculated.
- The dates on which the calculations have to be performed (not applicable to function deployments).

Reference

- *Calculator Reference Guide* - Calculator Input-Output Specification for the Push Calculator, Section Calculator Input Example.

Create Calculation Input through Designer Testlab

From within Designer Testlab, Calculator input can be generated from a testcase. In Designer, a testcase can be created for Structure Elements (resource root candidates), Dynamic and Static Deployments and Type Configurations.

ⓘ There is no support to generate Calculator Input for function deployments.

Reference

Designer Online Help – Test Cases; Generating Calculator Input

Testing the Deployment Package in Calculator using ProductXpress Testing Tools

There are two testing options available:

- Using the command line
- Using the Calculator Tester of the Calculator Console

Command line

This section describes how the Service Calculator can be tested independently and irrespectively of the state of the integration with Calculator.

There are two command line utilities available:

- PxTest, which provides a simple interface to the most used functionalities of Calculator, and allows easy distribution of requests across multiple Calculators
- CalcServerTest, which provides access to a very wide range of the Calculator APIs

Using PxTest

The command line utility PxTest can be used to send requests to and receive results from the Service Calculator. It basically simulates the production system by making specific calls to Calculator that reside locally or on a remote server.

In order to send a request (that is created in the previous task) to the service Calculator and save the results in an output file, PxTest can be invoked as follows:

```
PxTest calc <request filename>
```

This section provides further information for using `PxTest`.

Automatic distribution of requests

PxTest automatically distributes requests across multiple CPUs and/or multiple Calculator machines. The list of Calculators is configured in the PxTest.Settings file in the \etc folder of the Calculator installation.

Batch of Requests

When PxTest is provided a directory rather than a file, it will automatically calculate all files that have extension .clcin. The `-output-dir` option can be used to specify a directory where the output files are written.

Regression testing

When using the `-verify` and `-expected-dir` options, PxTest compares the output files to expected results, performing a complete regression test.

Timing information

PxTest can output timing information in the console with the `-time` option. This returns the total time elapsed from sending the request to the Calculator until receiving the result from the Calculator. If in the calculation request the attribute timer was set to true, besides the total request time, the more detailed timing information (this is also available in the output file) will be shown in the console.

PxTest Options

Using PXTest at the command line has many more options than those described here. For a list of the possible options, type the following at the command line:

```
PxTest help
```

Each option is explained in the *Calculator Reference Guide*.

Protocol

PxTest uses the SOAP protocol to communicate with Calculator.

Reference

- *Calculator Reference Guide* - Service Calculator Configuration and Maintenance section "The PxTest.exe Testing Tool".

Testing aggregations using PxTest

In addition to simple calculation requests, which can be used for requests that contain a large but limited amount of data, Calculator also supports the ability to perform aggregations across huge (potentially infinite) amounts of data using the aggregation protocol.

PxTest can be used to perform such aggregations. In order to perform an aggregation, the data is split into two parts:

- A single ‘master’ file that contains data that is shared for the entire aggregation, and the calculations that are to be performed for the aggregation

- Many detail files that contain the bulk of the detail. Note that detail files can also be packed into a single compressed zip file.

In order to perform an aggregation using PxTest, it is invoked as follows:

```
PxTest aggr <master-clcin> <link>=<detail-folder>
```

`<link>` is the name of an aggregation link and is specified in the `<master-clcin>`.

Using CalcServerTest

The command line utility `CalcServerTest` provides access to a very wide range of the Calculator APIs, including the ability to transform xml files.

ⓘ For day to day usage, PxTest is recommended.

In order to send a request (that is created in the previous task) to the service Calculator and save the results in an output file, `CalcServerTest` can be invoked as follows:

```
CalcServerTest -r=<request filename> -o=<output filename>
```

CalcServerTest Options

Using 'CalcServerTest' at the command line has many more options than those previously described. For a list of the possible options, type the following at the command line:

```
CalcServerTest –help
```

Each option is explained in the *Calculator Reference Guide*.

Reference

- *Calculator Reference Guide*, - Service Calculator Configuration and Maintenance section "The CalcServerTest.exe Testing Tool".

Calculator Console

It is also possible to test the Calculator Deployment through the Calculator Console. This can be done through the CALCULATOR TESTER that can be found under the TOOLS menu.

Within this interface, it is possible do the following:

- select either the request file, script or aggregation
- set the calculator options
- specify expected results (if required)
- view the output and results.

Reference

Calculator Console Online Help.

Web Portal

It is possible to test calculations through a Web browser. For more information, please see Calculator Web Portal (on page 41).

Key Management

If you decided to encrypt the deployment packages with custom keys, these keys must be managed in both Calculator and Designer.

Key Management in Designer

In order to encrypt packages on a high level, custom keys need to be created. These keys also have to be exported from Designer to import them in Calculators where deployment packages, which are encrypted with these keys, are loaded. Keys can also be exchanged between different Designer repositories via the export/import mechanism.

In Designer, select ENCRYPTION KEY MAINTENANCE from the TOOLS menu in order to generate, remove, export or import custom keys. See the *Designer Online Help*, section *Deployment Key Management* for details about these actions.

Key Management in Calculator

Packages that are encrypted during their creation need to be decrypted by Calculator. When custom keys have been used to encrypt packages, these keys must be present in Calculator for it to decrypt these packages.

Keys can be added to and removed from Calculator either via the command line with the `DepImp` tool or via the Calculator Console.

Command Line

To add: `DepImp -keyimport=<key export file>`

To remove: `DepImp -keyremove=<key label>`

Calculator Console

Select KEY MAINTENANCE from the TOOLS menu to import or remove custom keys. See section *Encryption Key Maintenance* of *Calculator Console Online Help* for details.

Using Keys

This section provides information regarding using keys.

Backup Keys

You should back up custom keys in case they are accidentally removed by the user from within Designer and/or Calculator. To back up a key, simply export the key from Designer. The key will be stored in a file as selected by the user.

ⓘ Keys cannot be exported from Calculator, but the keys that have been imported in Calculator can be found in `<Calculator Installation folder>/Etc/KeyRegistry.xml`.

Reference

- *ProductXpress Calculator Reference Guide* - Runtime Considerations for the Service Calculator
- *ProductXpress Designer Online Help* - section Deployment Package Encryption

Calculator Security

This section provides information regarding Calculator security, including how this should be achieved from the "client" or calling application.

Use SSL Certificate

SSL (Secure Sockets Layer) is a protocol that allows secure data transmission, by using an encryption/decryption system with a public key and a private key (known only by the recipient of the data) to encrypt data.

Calculator supports the use of SSL and you may import an SSL certificate and use it. The **SSLConf** tool is used to activate/deactivate secure communication and configure the keys and certificates. While SSL is enabled, the communication with the Calculator takes place through the

SSL protocol. From this point on, the calling applications (including Calculator Console) must also be configured appropriately to use SSL.

Command Line

```
SSLConf -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>]
```

```
SSLConf -off
```

```
SSLConf -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>
```

Calculator Console

In order to configure the Calculator Console to use SSL for a specific Calculator, select IMPORT SSL CERTIFICATE from the TOOLS menu to import the required SSL Certificate and select SSL SWITCH from the CALCULATOR menu, in order to use the specific SSL certificate. See the section *Encryption* of *Calculator Console Online Help* for details.

Reference

- *ProductXpress Calculator Reference Guide* - Managing Secured Sockets
- *ProductXpress Calculator Console Online Help* - section 'Access Control and Security' and 'Encryption'

Access Control

Access Control allows you to perform actions relevant to User Management and Access Control by allowing you to manage a user's access rights to the various features. The Access Control commands (*Enable Access Control* / *Disable Access Control* available through the ACCESS CONTROL menu of the Calculator Console) allow you to control which users or user groups are allowed to access certain features of the specific Calculator, by requiring the user to log on before he/she can view further details.

Calculator Console

Select the applicable USER MANAGEMENT menu options, i.e. Add/Remove User/User Group, Add/Remove User to User Group and Add/Remove User / User Group to Features from the ACCESS CONTROL menu of the Calculator Console to define the required users/user groups and their access to the required features. See section 'Access Control and Security' and 'Access Control' of the *Calculator Console Online Help* for details.

Reference

- *ProductXpress Calculator Console Online Help* - section 'Access Control and Security' and 'Access Control'.

Integration into the Production (Deployment) Environment

Integration is the process of embedding a product into a production environment by loading the product into the Calculator, connecting the data sources to the Calculator and connecting the client application(s) requesting calculation output to the Calculator.

The integration process involves steps such as:

- Creating the Deployment Environment
- Installing the Calculator and testing the installation
- Testing the deployment locally
- Testing the deployment through a remote machine
- Testing the deployment through the target client
- Fine tuning the Calculator for optimal performance

Since the installation and testing steps are different for both the Embedded and Service Calculator, each has its own section in this guide. However, for both types of Calculator, in order to begin the integration, a deployment environment has to be created. This subject is described in this section.

Deployment Environment

When integrating the Calculator with a client application, the data model used in the integration is the Deployment Environment.

The Deployment Environment is a model of the client system within ProductXpress.

The usage of a Deployment Environment means modelling part of the integration in ProductXpress. It simplifies the integration code for the client system.

ⓘ Both dynamic deployments and function deployments do not use a data model of the client system.

How To Create a Deployment Environment

- Analyze the data model of the client system.
- Determine the data that can be used by the calculations in ProductXpress.
- Design the Deployment Environment Elements so that the data that can be used in the ProductXpress calculations will fit into this structure.

Creating a Deployment Environment

This section provides information regarding creating a deployment environment.

Function Deployment

A function deployment does not need a deployment environment. It basically consists of the list of functions that can be calculated. This can be useful in cases where, for example, there is no client system model available or no complicated data structures (e.g. sets of sets) are needed.

Dynamic Deployment

By using a *Dynamic Deployment*, a Deployment can be made without having to create a data model of the client system. The advantage is not to have to create this data model. The disadvantage is the risk that different products have different interfaces, thus complicating integration code in the client application.

Model a Subset of the Client

When the client system has a large data model, it is not necessary to model the whole data model in the Deployment Environment. The subset that can be used by the calculations is sufficient.

Incremental Design

The Deployment Environment can be designed incrementally, but the structure of the Deployment Environment should fit all possible calculations, not just those in scope for a first phase of the project.

Changes in the Model

Addition of calculations results in changes in the integration code in the client because of the changes in the Deployment Environment. This is a sign that the Deployment Environment has been modeled by just looking at the data needed for particular calculations in ProductXpress and no heed was taken of the data model for the whole of the product.

For example, a particular calculation might make no difference between the owner and the insured. But if owner and insured are modeled as one Deployment Environment element, this has to change when adding calculations that do distinguish between owner and insured.

Reference

- *ProductXpress Designer Online Help*, section *Deployment Environment*

Integrating Embedded Calculator with a Client Application

With the Embedded Calculator, a tight and efficient integration can be established between the hosting application and the Calculator. As there is no middleware involved, the overhead of data exchange is minimal. This is for example especially beneficial for systems where many small calculations need to be performed.

The Embedded Calculator interface gives full control over package loading/unloading. Packages can be loaded on demand, reducing use of system resources. As the application controls the loading of packages, there is no need to use tools such as `DepImp`.

Distributing/installing the Embedded Calculator is as simple as copying the required files to the target machine or including the provided merge module in the application's installer.

The Embedded Calculator is available with three different interfaces: .NET (COM), Java (JNI) and C. The Embedded Calculator supports both the push and the pull model. In the push model dynamic, static and function deployments are supported (like in the Service based Calculator). In the pull model, only static deployments are supported. By being based on the pull model, the hosting applications must provide data to the Calculator on demand. This data consists primarily of information about instances and their definitions and of values of entered attributes on these instances. The push interface is explained in the section Integrating Service Calculator with a Client Application (on page 29).

The hosting application must implement a callback interface (IInstanceStore) that has several methods to return data to the Calculator. In Java and .NET, the interface can be implemented by deriving from a predefined type. With the C-interface, an intermediate shared library is required that exposes the callback methods to the Calculator.

ⓘ Integration of the Embedded Calculator is discussed in detail in the *Calculator Reference Guide*.

Embedded Calculator for .NET

The Embedded Calculator for .NET is implemented as an in-process COM server that is used in the .NET environment through a .NET technology called COM-Interop. This technology allows transparent interoperability to and from code that supports the COM standard. To use the Calculator in any of the .NET languages, developers need to add a reference to the COM type library; "CalculatorCOM 1.0 Type Library" (that describes all the types and interfaces exposed by *CalculatorCOM*). The development environment (e.g. Visual Studio) will then generate an interop-assembly in the .NET project that contains the code and types that make the interface between .NET and COM.

ⓘ `CalculatorCOM.dll` needs to be registered (`regsvr32`) on the target machine.

Perform a Calculation

Performing a calculation involves the following steps:

1. Create a new instance of the *PxCalculatorHome* class.
2. Call *initialize()* passing the path to the root of the Calculator's `Etc` folder.

3. Load a deployment package using the *loadDeploymentPackage()* method passing the name of the deployment package file.

If it is a pull Calculator, the following steps are performed:

1. Create a new Calculator session with a call to *createCalculatorSession()* passing the reference to the *IInstanceStore* implementation.

2. Perform the calculation with a call to *getValues()* passing the list of features (*PxFeatureRef* instances) to calculate.

If it is a push Calculator, the following steps are performed:

1. Create a new Calculator session with a call to *getPushCalculator().*

2. Perform the calculation with a call to *calculate()* passing the input data and the requested calculations (see XML Requests (on page 31)).

The following steps are performed for both a push and a pull Calculator:

1. Close the Calculator session with a call to *closeSession().*

2. Process the calculated results.

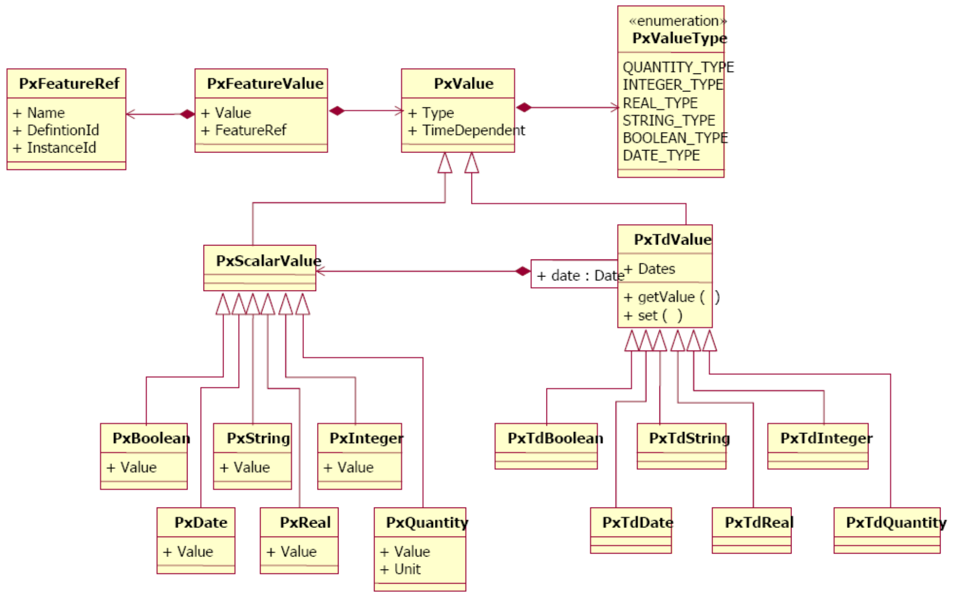
In the pull Calculator, in Calculator will make several calls to the *IInstanceStore* callback interface querying data it needs to perform the calculation. In the push Calculator, this is not necessary as all data that is needed to perform the requested calculations is passed at once.

Processing Results with the Push Interface

Processing Results with the Push interface is as easy as parsing the XML that is returned by the *calculate()* method. Refer to XML Requests (on page 31) for details regarding the XML structure.

Processing Results with the Pull Interface

CalculatorCOM contains a class hierarchy that is shown in the following UML class diagram:



PxFeatureValue is a container class that has two references to other classes accessible through the properties:

- **PxFeatureValue.FeatureRef**, a reference to a *PxFeatureRef* that contains information about the feature that was calculated

- **PxFeatureValue.Value**, a reference to *IPxValue*, a polymorphic reference to the actual feature value.

To find out what type of value is returned, the *IPxValue* interface has two read-only properties, *IPxValue.TimeDependent* and *IPxValue.Type*. By inspecting these two properties, the *IPxValue* interface can be narrowed to the proper type (e.g. *PxInteger*) and the actual value can be accessed.

Implement IInstanceStore with the Pull Interface

To implement the *IInstanceStore* interface in the .NET application, add a class that inherits from *IInstanceStore* (defined in the *CalculatorCOM* type library), for example:

```
using CalculatorCOM;
public class Callback : IInstanceStore
{
}
```

An instance of the *Callback* class can than be passed to the *PxCalculatorHome. createCalculatorSession ()* call.

The *IInstanceStore* interface consists of 8 methods that need to be implemented by the hosting application. See the *Calculator Reference Guide* for details.

To improve performance, the *IInstanceStore.getValues()* method may return all feature data at once. This will reduce the number of callbacks the Calculator needs to make and improves response time.

Embedded Calculator for Java

Calculating with the Java implementation of the embedded interface of Calculator is very similar to that with the .NET implementation (see previous section). The only difference is that instead of the class *PxCalculatorHome*, the class *PxCalculatorHomeJNI* is used. All other functions and steps are the same as those in .NET.

Embedded Calculator for "C"

While the .NET and Java interface use binary objects to exchange data between Calculator and the hosting application, the C-interface uses XML for this purpose. This ensures maximum interoperability with all sorts of environments. The format of the XML is described in the *Calculator Reference Guide*.

In order to use the Calculator for "C" in an application, first the library should be loaded into memory (e.g. by using Win32 *LoadLibrary()* call). Then, access to the different methods exposed by the DLL can be gained using for example *GetProcAddress()* using the function signatures as described in the *Calculator Reference Guide*.

When using the pull interface, the Calculator needs to retrieve data from the hosting application. It does so through the InstanceStore interface. When integrating with the embedded Calculator for "C" the InstanceStore methods need to be implemented in a shared library (e.g. DLL) that will be loaded by the Calculator. The InstanceStore could be implemented in the callback DLL itself or the calls to this shared library could be dispatched back into the application using for example "C" function pointers.

ⓘ As the C-interface uses XML to exchange data between Calculator and the hosting application, it will be slightly less efficient than the Java/COM interface due to the parsing overhead.

Reference

- *Calculator Reference Guide*, - Examples Embedded Calculator contains example code for C, .NET (C#) and Java.

Integrating Service Calculator with a Client Application

The Service Calculator only supports the push interface. In the push interface, the client application creates a connection with the Calculator. This can be done through SOAP, COM, CalcClient library or TCP/IP.

Over this connection, requests are sent to the Calculator in the following sequence:

1. A request contains an XML message that lists the Features or Functions that are to be calculated and all the input data that the Calculator needs to perform these calculations.
2. The Calculator then sends the result back to the client through the connection.
3. A result contains an XML message that contains the values for the Features or Functions that have been calculated.
4. After reading the result, the client can send the next request.
5. The process can be repeated until the client closes the connection.

On the Calculator side, the COM, CalcClient library and the SOAP interface all use the TCP/IP interface. Apart from headers and 'envelopes', the COM, CalcClient library, SOAP and TCP/IP interfaces all use the same request and result formats.

Types of request

Calculator supports the following request types:

- **Calculate**: performs the calculations on the data in the request
- **Transform**: applies an xslt stylesheet to the content of the request
- **Calculate And Transform**: first performs the Calculate request, then applies an xslt stylesheet
- **Transform, Calculate And Transform**: first applies an xslt stylesheet to the content of the request, then performs the Calculate on the resulting request, and finally applies another xslt stylesheet to the result of the Calculate step
- **Runtime Configuration Request**: returns information about the runtime configuration of Calculator
- **Aggregation Start**: starts an aggregation
- **Aggregation Map**: adds a detail to an aggregation (and performs the required calculations on that detail)
- **Aggregation Reduce**: reduces a set of map results to a single result
- **Aggregation Finish**: finishes the aggregation started with the aggregation start request

Interfaces

This section provides information regarding the various interfaces and protocols that Calculator supports to perform the requests mentioned in the previous section.

SOAP

Both the SOAP request sent to the Calculator and the SOAP response returned from Calculator contain a specific SOAP header as indicated in the *Calculator Reference Guide*. The body of the request and response SOAP messages is described in section XML Requests (on page 31).

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 UNIX 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)

(D)COM

The Calculator provides a COM interface, which is located in the Dynamic Link Library `Conn.dll`. Note that the Merge Modules must have been chosen during the Calculator installation.

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 and Output XML (see section XML Requests (on page 31)).

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

TCP/IP

A Request sent to the Calculator consists of a header containing the character sequence **ConnV3.0\n**, followed by a CalculationInput (see section XML Requests (on page 31)). Note that the TCP/IP protocol only supports Calculate requests.

The result consists of a CalculationOutput (see section XML Requests (on page 31)).

Integrating the Service Calculator

This section provides information regarding integrating the Service Calculator.

Changing Client Priority

It is possible to change the priority of requests received on a specific socket connection. By default, the priority is ‘Normal’. Other priorities are ‘Low’ and ‘High’.

To change the priority for a specific socket connection, the client must send a message with a specific format over that socket connection to the Calculator. Requests sent after this message over the same socket connection will be handled with the corresponding priority. See the *Calculator Reference Guide* for details.

Reference

- *ProductXpress Calculator Reference Guide* - Interfacing the Service Calculator.

XML Requests

This section provides information regarding XML requests.

Requests for Static and Dynamic Deployments

An XML request for static and dynamic deployments contains the following sections:

- **DeploymentReference**: Specifies for which Deployment the request is meant.
- **CalculationData**: Provides the data necessary to perform the calculations.
  This section contains an XML representation of Deployment Environment (static) or Product (dynamic) elements.
- **Calculations**: Specifies which Calculations need to be calculated.
  This section contains an XML representation of Deployment Environment (static) or Product (dynamic) elements.
- **CalculationDates**: Specifies for which dates the Calculations have to be calculated.

XML Representation of Deployment Environment and Product Elements

The Deployment Environment or Product determines the structure of the requests and results (of the `<Calculation>` and `<CalculationData>` sections to be precise).

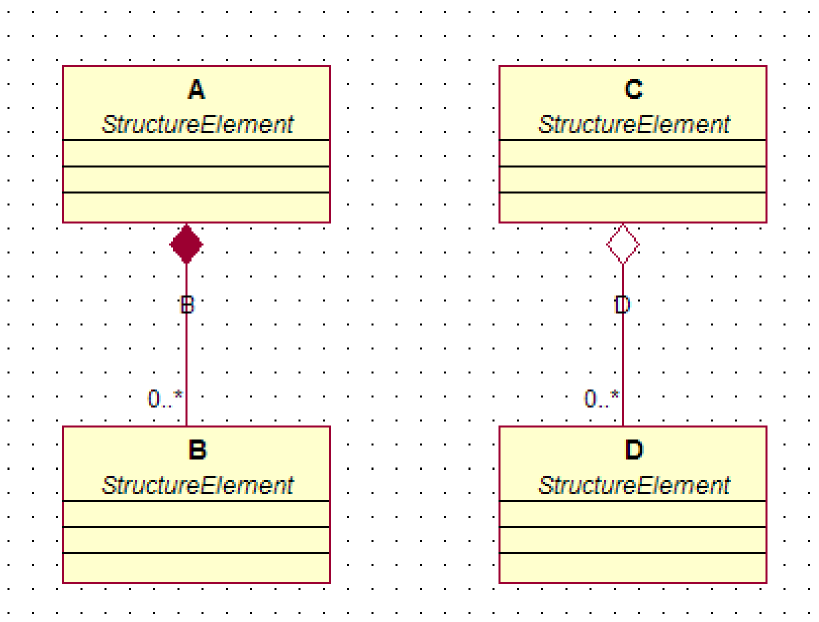
The relation is as follows:

- *Deployment Environment or Product Structure Element*
  Corresponds to an XML element.
  The XML **tag name** is the name of the **Structure Element**.
  The XML element contains an XML attribute **id** that uniquely identifies the element within the request. Elements can refer to other elements through an XML attribute **ref** that contains the id of the referred element.
  This XML element contains a **Features** element that contains an XML element for each Feature of the Structure Element. The name of these XML elements are the names of the Features.
- *Deployment Environment Structure*
  The Deployment Environment or Product tree structure corresponds with the tree structure of the XML.
  A Deployment Environment element named "A" with a **multiple link** to Deployment Environment element named "B" corresponds to an XML element with name "A" that contains an XML element with the name of the link that can contain multiple XML elements with name "B".
  A Deployment Environment element named "A" with an **optional link** to Deployment Environment element named "B" corresponds to an XML element with name "A" that

contains 0 or 1 XML element representing the link; if there is 1 XML element, it will contain one XML element named "B".
A **Resource link** from Deployment Element "A" to Deployment Environment Element "B" corresponds with a special XML element in "A" that refers to "B".

Examples:

Deployment Environment:



Structure Element "A" contains Structure Element "B" with attribute "E" that has value 10

```
<A>
     <Links>
     <B>
          <B id='ID:2'>
               <Features>
                    <E>10</E>
               </Features>
          </B>
     </B>
     </Links>
</A>
Structure Element ‘C’ has a resource link to Structure Element ‘D’
<C id="ID:5">
     <Links>
     <D>
          <clc:Reference ref="ID:4"/>
     </D>
     </Links>
</C>
<D id="ID:4"/>
```

Example of a Calculation Input

Although the *Calculator Reference Guide* should be consulted before creating Calculation requests, the following provides information for a simple Calculation Input XML.

The Calculation Request is an XML replication of the product structure that, includes the input attributes and calculations (features):

The Calculation Input begins with the header:

```
<?xml version='1.0'?>
<clc:CalculationInput
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInpu
tOutput/CalculatorElement' xmlns='Testlab_Deployment'>
```

The URI identifies the XML namespace of the XML elements in Calculator Input and Output XML that are used for the structure elements, features, values and links in this deployment. Initially, a namespace is automatically generated from the deployment. It must be changed when the client wants to use a different namespace for these XML elements. See http://www.w3.org/TR/REC-xml-namess for more information on XML namespaces. Please also view the *Calculator Console Online Help* regarding changing the URI.

It is followed by the deployment reference (name and version):

```
<clc:DeplR dep-name='tc1 deployment' ver-sel='last'/>
```

This is followed by the CalculationData Section. The root structure element is Yearly_time_account. This is linked to Weekly_timesheet (through the link Weekly_timesheet) which is linked to the Structure Element Daily_work (through the link Daily) which is linked to the Structure Element Work_element through the Work_element Link. Work_element has a feature (entered attribute in Designer) called Number_of_hours_worked. This is assigned the value of 10. Account also has a feature called customer_name. in this instance the value is HP.

```
    <clc:CalculationData>
      <Yearly_time_account id='ID1' ext-id='d57f9592-bb4e-45ff-857
a-0f900a697078:1'>
        <Links>
          <Weekly_timesheet>
            <Weekly_timesheet id='ID5' ext-id='bbc0c99c-4f1d-4229-
bd83-5501eac77f88:1'>
              <Links>
                <Daily>
                  <Daily_work id='ID9' ext-id='372dbd83-5b3b-4ccb-
8123-1dc6a4e15dcd:1'>
                    <Links>
                      <Work_element>
                        <Work_element id='ID14' ext-id='c14fa5b5-5
d8e-4cfe-beb3-906acbf067ff:1'>
                          <Features>
                            <Number_of_hours_worked>10</Number_of_
hours_worked>
                          </Features>
                          <Links>
                            <Account>
                              <clc:Reference ref='ID21'/>
                            </Account>
                            <Leaf/>
                          </Links>
                        </Work_element>
                      </Work_element>
                    </Links>
                  </Daily_work>
                </Daily>
              </Links>
            </Weekly_timesheet>
          </Weekly_timesheet>
        </Links>
      </Yearly_time_account>
      <Account id='ID21' ext-id='48a66a4a-78ff-4b27-a52f-9d5da10ca
  620:1'>
        <Features>
          <Customer_name>HP</Customer_name>
        </Features>
      </Account>
    </clc:CalculationData>
```

The next section is the Calculation section. This has exactly the same structure as the CalculationData section. In this section the Calculations are listed. So in this example, Yearly_time_account and Weekly_timesheet and Work_element contain the calculation Total_time_worked.   Daily_work contains the calculation only_24_hours_in_a_day.

```
  <clc:Calculation name='main test'>
    <Yearly_time_account ref='ID1'>
      <Features>
        <Total_time_worked/>
      </Features>
      <Links>
        <Weekly_timesheet>
          <Weekly_timesheet ref='ID5'>
            <Features>
              <Total_time_worked/>
            </Features>
            <Links>
              <Daily>
                <Daily_work ref='ID9'>
                  <Features>
                    <Total_time_worked/>
                    <only_24_hours_in_a_day/>
                  </Features>
                  <Links>
                    <Work_element>
                      <Work_element ref='ID14'>
                        <Features>
                          <Total_time_worked/>
                        </Features>
                      </Work_element>
                    </Work_element>
                  </Links>
                </Daily_work>
              </Daily>
            </Links>
          </Weekly_timesheet>
        </Weekly_timesheet>
      </Links>
    </Yearly_time_account>
```

The Following section contains the CalculationDates. In this example the calculation date is 2006-04-25

```
<clc:CalculationDates><clc:At>2006-04-25</clc:At></clc:Calculation
Dates>
```

Lastly the InputData specifies which Calculations to perform. In this instance all the calculations specified under the calculation name 'main test' (thus all of the above) will be performed. We will be given results for 4x Total_Time_Worked and 1x Only_24_hours_in_a_day.

```
      <clc:InputData scenario='main'/>
    </clc:Calculation>
```

- </clc:CalculationInput>

Building Requests

This section provides information regarding building requests.

Different Calculations on Different Dates

You may want to request different (groups of) calculations on different calculation dates. This can be achieved by using multiple `<Calculation>` elements in the `<CalculationInput>` element, each containing a section with requested calculations and a `<CalculationDates>` element. The `<Calculation>` element has an optional `name` attribute, with which the different groups of calculations can be distinguished in the output.

Performance

The following Performance issues should be considered when designing requests:

- Bundling separate requests into a batch that is sent as a single request to Calculator. This can reduce the total amount of data sent back and forth to the Calculator.
- On multi CPU machines: the Calculator will by default use all the available CPU when it is called concurrently by multiple clients (unless this is explicitly restricted in the Settings file).
- Not sending XML elements and attributes that are unused in the calculation.

Deployment Version Selection Methods

As ProductXpress supports versioning throughout the whole process from development to run-time, there can be multiple versions of a deployment (thus indirectly of a product or set of functions) in Calculator. Calculator supports three methods to select a specific version of a deployment for which a calculation needs to be performed:

- Select a specific version - the version number that is provided in the request will be used to select the specific deployment version
- Select the last version - the last version of a deployment will be selected
- Select the version that is active on a date - the deployment version that is active on the date that is provided in the request will be selected.

This is achieved by the use of the `ver-sel=` in the Calculation Input.

Reference

- *Calculator Reference Guide* - Calculator Input-Output Specification for the Push Calculator.

Scenarios

In cases where calculations have to be performed for a policy with different values for some of the attributes, *Scenario* elements can be defined in the Calculator Input.

A *Scenario* element contains the same structure as *Calculation Data*. Within a *Scenario,* new instances and new attribute values can be defined in relation to the *Calculation Data* and existing attribute values in *Calculation Data* can be overridden.

The *Calculation Data* itself is regarded as a scenario as well, with a reserved name: *main*. In the *Calculation* section one or more *Scenarios* 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 (`<CalculationData>`) is selected by referring to the reserved name *main* in *InputData*. If no *Scenario* is selected, all *Scenarios* (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.

ⓘ This is not applicable to function deployments.

Example:

```
</clc:CalculationData>
    <clc:Scenario name='NumberOfHoursWorked@6'>
      <Yearly_time_account ref='ID1' ext-id='d57f9592-bb4e-45ff-85
7a-0f900a697078:1'>
        <Links>
          <Weekly_timesheet>
            <Weekly_timesheet ref='ID5' ext-id='bbc0c99c-4f1d-4229
-bd83-5501eac77f88:1'>
              <Links>
                <Daily>
                  <Daily_work ref='ID9' ext-id='372dbd83-5b3b-4ccb
-8123-1dc6a4e15dcd:1'>
                    <Links>
                      <Work_element>
                        <Work_element ref='ID14' ext-id='c14fa5b5-
5d8e-4cfe-beb3-906acbf067ff:1'>
                          <Features>
                            <Number_of_hours_worked>6</Number_of_h
ours_worked>
                          </Features>
                        </Work_element>
                      </Work_element>
                    </Links>
                  </Daily_work>
                </Daily>
              </Links>
            </Weekly_timesheet>
          </Weekly_timesheet>
        </Links>
      </Yearly_time_account>
    </clc:Scenario>
    <clc:Calculation name='main test'>
      <Yearly_time_account ref='ID1'>
        <Features>
          <Total_time_worked/>
        </Features>
        <Links>
          <Weekly_timesheet>
            <Weekly_timesheet ref='ID5'>
              <Features>
                <Total_time_worked/>
              </Features>
              <Links>
                <Daily>
                  <Daily_work ref='ID9'>
                    <Features>
                      <Total_time_worked/>
                      <only_24_hours_in_a_day/>
                    </Features>
                    <Links>
                      <Work_element>
                        <Work_element ref='ID14'>
                          <Features>
                            <Total_time_worked/>
                          </Features>
                        </Work_element>
                      </Work_element>
                    </Links>
                  </Daily_work>
                </Daily>
              </Links>
            </Weekly_timesheet>
          </Weekly_timesheet>
        </Links>
      </Yearly_time_account>
      <clc:CalculationDates><clc:At>2006-04-25</clc:At></clc:Calcu
lationDates>
      <clc:InputData scenario='main'/>
    </clc:Calculation>
    <clc:Calculation name='NumberOfHoursWorked@6 test'/>
</clc:CalculationInput>
```

This example is an extension of the above Calculation Input Example where an extra scenario called 'NumberOfHoursWorked@6' has been added. In this example both the main and the NumberOfHoursWorked@6 scenario will be selected. Because of the difference in input (10 and 6 respectively) different Calculation results will be generated.

Feature References in Calculation Dates

The range of dates on which the calculations have to be performed are specified in the `<CalculationDates>` section using several elements (like `<From>` and `<To>`). These elements either contain constant values or references to *Features*. In the latter case, the element gets the value of the *Feature* (either calculated or retrieved from `<CalculationData>`).

ⓘ This is not applicable to function deployments.

Example: Calculate from Policy Issue to Maturity

```
<clc:CalculationDates>
```

```
		<clc:Range>
			<clc:From>
				<Prod ref="ID1234">
					<Features>
						<Policy_Issue_Date/>
					</Features>
				</Prod>
			</clc:From>
			<clc:To>
				<Prod ref="ID1234">
					<Features>
						<Maturity_Date/>
					</Features>
				</Prod>
			</clc:To>
			<clc:StepSize unit="year">1</clc:StepSize>
		</clc:Range>
	</clc:CalculationDates>
```

Timers

The Calculator has the ability to return timing information for a request. To enable this, the attribute `timer` in the calculation request must be set to true. The timing information will be returned in the output from Calculator.

Reference

- *Calculator Reference Guide*,- Calculator Input-Output Specification for the Push Calculator.
- *Designer Online Help* – Test Cases; Generating Calculator Input.

Results for Static and Dynamic Deployments

The result for static and dynamic deployments contains the following sections:

- **Calculation Results**: Similar to the requested *Calculation*s section in the request, but now augmented with the values that were calculated for each requested feature.
- **Error**: When a feature could not be calculated, an error section indicates the cause.

Processing Results

Similarly to how the input data in `<CalculationData>` is built up, the output data `<Calculation>` is read, as it has the exact same structure. The output contains the calculation results of the calculations that have been requested in the `<Calculation>` sections on the dates as specified in the `<CalculationDates>` section.

Handling Results

This section provides information regarding handling results.

Error Handling

If an error has occurred during the calculation, it will be returned in the output as an `<Error>` element. This element contains an error number that represents the type of error. The

`ErrorConversion.Settings` file in `<Calculator Installation folder>/Etc` can be used to let Calculator output a different message corresponding with the error number. See the *Calculator Reference Guide* for details.

Different Calculations on Different Dates

When in the input multiple, `<Calculation>` elements are used to request different calculations on different dates, the output will contain as many `<Calculation>` elements, each containing the results for the requested calculations for that `<Calculation>` element. If for a `<Calculation>` element the `name` attribute is specified in the input, this name will also be present in the corresponding `<Calculation>` element in the output.

Requests for Function Deployments

An XML request for function deployments contains the following sections:

- **DeploymentReference**: This section specifies which Deployment the request is meant for.
- **TestData**: This section contains one or more **TestSet**s.
- **TestSet**: This section provides values for (function) variables that are used in the functions.
- **RequestedFunctions**: This section 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.

Function Deployment Requests

This section provides information regarding function deployment requests.

Permutations

A (function) variable in a TestSet and a parameter of a Function in RequestedFunctions can be permuted just like in a Test Set in Designer. The Functions (that reference such Test Sets) will be calculated for all the permutations of the permuted variables. The permutation of function parameters will result in multiple calls to the Function.

Reserved Variables

Calculation Date and *Self* are reserved variables that are referenced with reserved elements (in the Calculator namespace) in the Test Set.

External Functions

When an external function is to be used as the value of a function variable, simply the name of the external function is provided as the value.

ⓘ Internal functions (developed in Designer and available in the deployment package) can also be used as values for function variables.

Tables

When a variable has a value definition of type TableID, the name of the table is provided as its value. Note that this name is the name of one of the tables that are in the value domain of the value definition. This implies that the value definition must have a value domain with a set of tables.

Results for Function Deployments

The result for function deployments contains the following sections:

- **TestSet (one or more)**: This section contains the results of the Functions that were calculated with the data provided in this TestSet.
- **Error**: When a Function could not be calculated an Error section indicates the cause.

Handling Function Deployment Results

This section provides information regarding handling function deployment results.

Error Handling

See Error Handling (on page 38) of Results for Static and Dynamic Deployments (on page 38).

Reference

- *ProductXpress Calculator Reference Guide*, Service Calculator Configuration and Maintenance

Integration Checklist

The following steps can be used to check the integration. They should also be followed whenever troubleshooting is required.

To perform these checks, it is necessary to trace the requests and results and examine the Calculator logging.

1. The request reaches Calculator.

   Network failures or incorrect TCP/IP or DCOM configurations can prevent the request reaching Calculator.
2. The request contains correct XML.

   Load the traced request into an XML editor/reader.
3. The request contains a correct XML representation of the Deployment Environment or Product Structure.

   Examine error messages in the result.
4. The request contains all data needed for the calculations.

   Examine error messages in the result.

When the first of these stages has been completed, requests that return an error can also be analyzed with `PxTest` by extracting the request from the Calculator log.

When the first three of these stages have been completed, requests that return an error can be also analyzed within Designer by constructing a corresponding test in TestLab or TestSet editor.

An XML request can be imported directly in Testlab.

Calculator Web Portal

The Calculator Web Portal hosts all Calculator-related information including end-user guides and samples of integration code through a Web browser, so that all collateral is easily accessible to local and remote stakeholders in HTML and download formats.

This also includes the Calculator Web Console, with all of the capabilities of the "Calculator Console" for operational management, testing and other related operations directly from the portal. The test capability includes the ability to run calculation requests and scripts to a selected Calculator.

While all users can have access to the Calculator user documentation, access privileges are required by default for all users who want to use the Calculator Web Console capabilities.

Connect to a Calculator

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

On Windows only:

- Open the Start menu and select Calculator 3.7 followed by Calculator Web Portal.

On all platforms, including Windows:

- 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, to 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.

Developing External Functions

An External Function is a function that resides outside of ProductXpress. It is a hard-coded piece of software, implemented in a library that can be accessed by the Calculator in order to retrieve data that is used in a calculation. External functions can be written in C or in Java. The C interface can also be used by other programming languages that can interface with C.

Typical examples of External Functions are:

- Table lookups for risks in an administration system
- Retrieve frequently changing data such as stock quote data or interest rates

Advantages of External Functions over External Tables (which can also be used to access external data) are:

- The ability to use complex data types as arguments and return value.
- It is possible to program more complex logic in an external function than just a lookup.
- External Functions can do a lookup in a database with an arbitrary schema and of any brand.
- It also is possible to return error messages from External Functions.

Disadvantages are the higher level of complexity and the fact that a fatal programming error occurring in an external function will terminate the Calculator.

Using External Functions

Calculator caches external function results. This means that when an external function is used with the same parameters for the second time, the Calculator will use the cached value. Therefore, the external functions must be "immutable", that is, they must return the same value for a given set of parameter-values.

An operational procedure must be set up to allow for any changes in data that an external function accesses.

In Service Calculators, the suspend command can be used to suspend the Calculator while a data change is taking place. This will also clear the Calculator cache.

In Embedded Calculators, the clearGlobalCalculatorCache API can be used to achieve the same result.

Developing External Tables

In cases where table data should not be part of the versioning cycle of products that use that data, external tables can be used. External tables are tables that reside and can be maintained outside of ProductXpress, but can be accessed within ProductXpress.

The advantage of using an external table rather than an external function to access external data is that the internal function *lookup* can be used on these tables. This internal function supports useful functionality, such as interpolation and rounding, that otherwise in case of external functions would have to be implemented.

Creating and Storing External Table Data

This section provides information regarding creating and storing external table data.

Create and Update in Designer

External Tables are initially created from within Designer in the *Table Editor*, but the data is saved in a file outside of Designer. This file can be reopened and edited in Designer in the Table Editor or it can be modified outside of Designer. See the *Designer Online Help* for creating and updating external tables in Designer.

Store an External Table

External tables are stored outside of the Designer repository and can be stored in one of the following ways:

- Stored in XML format on the file system
- Stored in a Database

Store External Table in a Database

ProductXpress supports the option to store an external table in a database. It creates a table in the database where the external table resides. The external table can then be maintained in the database with third party tools/applications.

Supported databases are *Microsoft SQL Server* and *IBM DB2*.

The storage in a database can either be done through the command line or the Calculator Console. Optionally, a name can be provided for the name of the table that will be created in the database. By default this name is the name of the external table.

ⓘ When the database is located on a different geographical location, it may slow down the performance due to extra infrastructure overhead including database response time and network latency.

Loading External Table Data into the Calculator

This section provides information regarding loading external table data into the Calculator.

Loading Table Data from a Database

To load table data from a database:

Command Line TableImp

The utility `TableImp` is used to load an external table in a database.

Usage: `TableImp -add=<external table file> -database=<database spec> -name=<table name>`

```
<database spec>: <driver>:<user>[/<password>]@[<server>]:[<schema>]
<driver>: "DB2" | "MSSql"
```

Calculator Console

The menu item ADD TABLE in the EXTERNAL TABLES menu is used to load an external table in a database. The field *Database URL* contains the specification for the database. Refer to the `<database spec>` in the previous section for the format of this field. The field Table Name contains the name of the table that will be created in the database. If this field is left empty, the name of the external table will be used.

Loading Table Data from a File

To load table data from a file:

Command Line

To load an external table into Calculator, the utility `TableImp` is used.

Usage: `TableImp -add=<external table file>`

Calculator Console

To load an external table through the Calculator Console, use ADD TABLE in the EXTERNAL TABLES menu.

Loading Table Data from a CSV File

When an external table is created from within Designer using the Table Editor, it is possible to enter table values either manually or from a CSV (comma separated values) file. In case large amounts of values are involved, problems might arise when displaying these values in the Table Editor. In that case, it is possible to save an empty external table from the Table Editor (meaning no dimension and no return values are entered), followed by the external table being loaded into the Calculator together with a (local) CSV file as follows:

Command Line

```
TableImp -add=<external table file> -csv=<local csv file>
```

Calculator Console

To load an external table through the Calculator Console, use ADD TABLE in the menu EXTERNAL TABLES. In addition to the external table name, the name of the CSV file can be specified.

Note that reading external table values from a CSV file is subject to a number 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. These settings must be specified in a

Settings file called 'TableDataCSVLoader.Settings'. See the *ProductXpress Calculator Reference Guide* for more detailed information on this file.

Update External Tables

There are several ways to update an external table. If the external table resides in a database, it can be updated within the database. If the external table does not reside in a database, either the XML file in which the data resides can be updated directly or this file can be loaded in Designer and updated in the Table Editor. In Designer, use the OPEN EXTERNAL TABLE option in the TOOLS menu to open an external table in a Table Editor.

Signal External Table Changes to the Calculator

Once the external table is updated either in Designer or directly, the update should be sent to the Calculator. The update is either done via the command line or the Calculator Console.

ⓘ Before updating the external table, the Calculator **must be** suspended. Then the table update can be performed. After this, the Calculator can be resumed or stopped and started to use the changed data. The exception to this rule is if the external table resides in the database. See External Tables in Database (on page 47).

Command Line

```
TableImp -update=<external table file>
```

Calculator Console

Select UPDATE TABLE in the EXTERNAL TABLES menu and select the external table file.

External Tables in Database

Once the external table is updated directly in the database (without using the Calculator Console or TableImp), the Calculator must be suspended and resumed to make sure these updates are taken into account.

However, if the changes were made on values that Calculator had not yet retrieved, the changes will be used without having to suspend and resume the Calculator, as the Calculator will not have cached them and thus must retrieve them from the database. Typically, when new rows are added, this will be the case.

TableImp Options

Deploying an external table configures which table data is used for the external table component in calculations. It is also possible to use a CSV file that contains the required values for your calculations. TableImp has a number of options.

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

These are listed and described in the *Calculator Reference Guide*.

Reference

For more TableImp options and for more detailed information about the usage of external tables, see:

- *ProductXpress Calculator Reference Guide* - Service Calculator Configuration and Maintenance, section External Table Management.

Operational Management

This section provides information regarding operational management.

Configuration Management of ProductXpress Assets and Application Development Assets

In this section, we describe the relationship between the application development assets that communicate with Calculator and the assets defined in ProductXpress (i.e. products and deployment packages) and how to manage that relationship.

As in any software development, over time multiple versions of the software will be developed and used in production. It can become complicated when the software uses external resources, of which multiple versions can also exist. In the case of ProductXpress, products and deployment packages, which can have multiple versions, are developed outside of the software that uses/embeds Calculator.

In order to manage the complexity of which version of the software using Calculator uses which version of the deployment packages, the specific versions of the deployment packages used should be versioned together with the specific version of the software within the same source control system.

ⓘ It is possible to set the precision accuracy and rounding method for each Calculator instance that is installed separately. However, caution must be exercised when comparing tests results (both in Designer, Calculator/Calculator Instances) using different precision settings. It is possible for trade offs between performance and precision to be made.

ⓘ Some degree of performance degradation is to be expected when using higher precision calculations.

Using the Service Calculator Service

From time to time, the Service Calculator will need to be stopped/suspended due to various reasons, e.g. maintenance or synchronization with changes in external data, or addition/removal of deployment packages. This section describes how to do this.

Stop and Start as Service/Daemon

When the Calculator starts, it will load all the deployment packages and make use of some settings according to the several Settings files present in `<Calculator Installation folder>/Etc`. When the Calculator is stopped, it will handle the current request and then shut down.

In order to stop and start the Service Calculator as a service (Windows) or daemon (UNIX), either the command line or the Calculator Console can be used.

ⓘ The Calculator can be stopped both in Windows and UNIX via `DepImp`:

```
DepImp -shutdown
```

Command Line UNIX

As there are many flavors of UNIX, there are as many ways to start and stop daemons. The most common method to start and stop Calculator as a daemon on UNIX systems is as follows:

- Start: From console, run Calculator without any parameters: `calcserver`. Calculator will start itself as a daemon.
- Stop: Find the process ID of the Calculator daemon and use the UNIX `kill` command to stop it.

Command Line Windows

- Start: Use the following command in a command line: `net start calcserver3`
- Stop: Use the following command in a command line: `net stop calcserver3`

Calculator Console

To stop the Calculator, select STOP from the CALCULATOR menu.

To start the Calculator, select START from the CALCULATOR menu.

ⓘ The START menu item simply executes the line (as a system command) that is specified in the file `ConsoleSystem.properties` in `<Calculator Installation folder>/bin`. For Windows, it uses the line after "calculator_start_windows=" and for UNIX it uses the line after "calculator_start_unix=". These can be changed by the user.

Suspend and Resume the Calculator

When the Calculator is suspended, it will handle requests that are still in progress but bounce new requests until Calculator is resumed. When it is resumed, it will have cleared all the caches that are based on external data.

Command Line

- To suspend: `DepImp -suspend`
- To resume: `DepImp -resume`

Calculator Console

- To suspend the Calculator, select SUSPEND from the CALCULATOR menu.
- To resume the Calculator, select RESUME from the CALCULATOR menu.

Calculator Service

This section provides information regarding the Calculator Service.

Stop and Start in interactive mode

Both in Windows and UNIX the Calculator can be started in interactive mode, i.e. the Calculator will start as a normal user process. The command is as follows:

```
calcserver -interactive
```

The Calculator is stopped by simply ending the process.

Changes in External Data

When changes have been made in external data, this will not immediately be used by Calculator. In order to let Calculator use the changed external data, it must either be stopped and started or suspended and resumed.

Retrieving the Version of Calculator

The version of Calculator can be retrieved as follows in a command line:

```
calcserver -version
```

The Calculator Console displays the version in the status bar.

Reference

- *ProductXpress Calculator Reference Guide*: Service Calculator Configuration and Maintenance , section DepImp.exe
- *ProductXpress Calculator Console Online Help* - section Calculator Commands.

Calculator Management and Monitoring

The Calculator can be managed and monitored by the Calculator Console. The Calculator Console provides a user interface for operating the Calculator and analyzing its performance. It contains the following management functions:

- Start and stop the Calculator.
- Suspend and resume the Calculator.
- Add and remove deployment packages.
- Add and remove external tables. The tables can be stored in an XML file or in a database (only applicable to a local Calculator).
- Add security to Calculator.
- Add and remove users and user groups to Calculator. When a Calculator runs in secure mode only, users and user groups that are known to that Calculator can connect.
- Import and remove encryption keys.

In addition, the Calculator Console displays real-time Calculator performance information such as CPU usage, number of connected clients, number of requests per second and memory usage. Performance data can be viewed both as a time-series graph and a real-time data histogram.

The Calculator log file is also accessible through the Calculator Console.

The Calculator Console can connect to both local and remote Calculators.

Starting Calculator Console

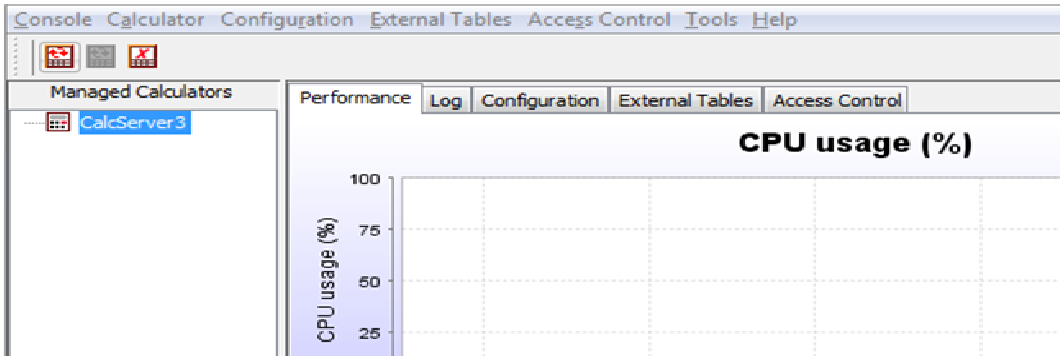
The Calculator Console is a Java Swing application and can be started by either of the following methods:

- Selecting Calculator Console in the Start\All Programs\HP ProductXpress\Calculator menu

- Executing the `CalcManConsole.exe` file on a Windows platform or the `startCalcManConsole` file on other platforms (see the platform readme file on how to configure the Calculator Console on non Windows platforms) in the Calculator bin directory

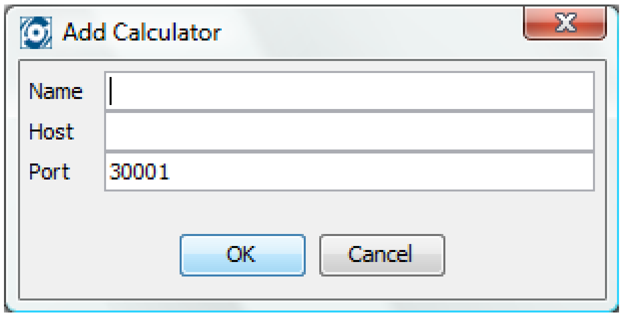
Connecting to a Calculator

Connecting to a previously added Calculator is automatic, by clicking on the Calculator of interest within the Managed Calculators pane (on the left).



After selecting a Calculator, the Calculator Console will try to establish a connection. Once connected, the performance indicators will show the information of the new Calculator.

ⓘ In order to add a Calculator, select Add from the Calculator menu and fill in the required details (i.e. Name, Host and Port Number).



Name	
Host	
Port	30001

Calculator Monitoring

This section provides information regarding Calculator monitoring.

Java Runtime Environment

The Calculator Console assumes that a Java Runtime Environment 1.5 or higher has been installed.

Management Functions

Management functions are available for both a Calculator that is running on the same machine as the Calculator Console, as well as on a remote machine.

Management functions are only available for Calculator versions 3.0 and higher. Trying to connect to an earlier version of Calculator will disable the management functionality including the option to view deployed packages and external tables. It is still possible to analyze the performance data for that Calculator.

Reference

For a detailed description of Calculator Console functionality, see the *CalculatorManagementConsoleOnlineHelp.pdf*.

Analyzing Production Problems

There are several methods in which ProductXpress provides support to analyze problems in relation to the Calculator. These problems can vary from incorrect output by Calculator to no response from Calculator.

Tracing

Tracing can be turned on to analyze a request in more detail. Tracing provides the user with detailed information about what happens internally within Calculator when it receives a request.

Tracing can be turned on by adding the `<Trace>` element with a value for the `value` attribute in the file `Calculator.Settings`.

A typical trace is to simply log the request as received by Calculator and the output returned from Calculator:

```
<Trace value="Conn:ConnectorRequestMethod" />
```

ⓘ Tracing can severely impact the performance of Calculator. It is advised to only use tracing to analyze problems.

Calculator Console

The Calculator Console provides useful information about a Calculator, such as the number of connected clients and the CPU and memory usage. See Calculator Management and Monitoring (on page 51) for details.

Particularly, in case of problems, the Log of a Calculator can be consulted by visiting the Log tab. The Log displays, among other things, which deployment packages have been loaded and what the current status is of Calculator. Also, the Settings of a Calculator can be retrieved by selecting SETTINGS in the CALCULATOR menu.

Using Multiple CPUs

The ProductXpress Calculator is designed for spreading requests over multiple concurrent execution threads. The number of CPUs that will be used by Calculator defaults to the number available on the machine and this number will be calculated at the start of the Calculator service. Dual Core and Hyperthreading are both supported by the Calculator.

ⓘ It is possible to limit the number of CPUs available to Calculator, if needed. See the *Reference Guide* for more details.

Single Versus Multiple CPUs

On a single CPU machine, Calculator can only handle one request at a time. Sending multiple requests to this machine will result in a queue of requests that Calculator will process one at a time in a sequential manner. For example, if four requests are sent to a machine at the same time, one will be processed immediately and the other three will be put in a queue. Once the first request is finished, the second request (the first in the queue) will be processed and the other two will continue waiting.

In order to process multiple requests at the same time, a machine with multiple CPUs is required.

ⓘ It is also possible to process parallel requests on multiple machines that each have one Calculator running. For more details, refer to the *Calculator Reference Guide*.

On a multi-CPU machine with one Calculator installed, all requests are sent to this Calculator on the same port. Calculator will by default utilize all the CPUs at hand and thus can process multiple requests at the same time, as each request is processed by a different CPU. For example if four requests are sent to the Calculator at the same time on a machine with four CPUs, all four requests will be processed at the same time. Effectively, this means that Calculator is four times faster compared to when it would be running on a single CPU machine. Furthermore, when we would send six requests, four requests would be processed immediately at the same time, and the two other requests will be queued and processed when one of the CPUs becomes free.

In order to test this scenario using the Service Calculator, the following steps can be followed:

1. Deploy the deployment and required external tables using the Calculator Console or DepImp/TableImp

2. Edit the PxTest.Settings file in the etc folder of your installation, and ensure that it reads:

```
   <?xml version='1.0' ?>
   <PxTest>
         <Calculators>
               <Calc id="calc1" host="localhost" port="30001"
threads="1"/>
         </Calculators>
   </PxTest>
```

3. Open a DOS prompt and run the command:

```
PxTest calc inputs -time
```

   This will process all requests in the folder inputs and give a total time for all calculation requests from 1 client to 1 CPU.

4. Edit the PxTest.Settings file in the etc folder of your installation, and ensure that it reads:

```
<?xml version='1.0' ?>
<PxTest>
   <Calculators>
         <Calc id="calc1" host="localhost" port="30001"
threads="2"/>
   </Calculators>
</PxTest>
```

5. Open a DOS prompt and run the command:

```
PxTest calc inputs -time
```

   Now, PxTest will assume that the Calculator on localhost has two CPUs available and will send the requests on two different threads.

The time recorded in step 5 should be half of the time recorded in step 3.

Publishing

This section provides information regarding publishing.

The Publishing Process (Pipeline)

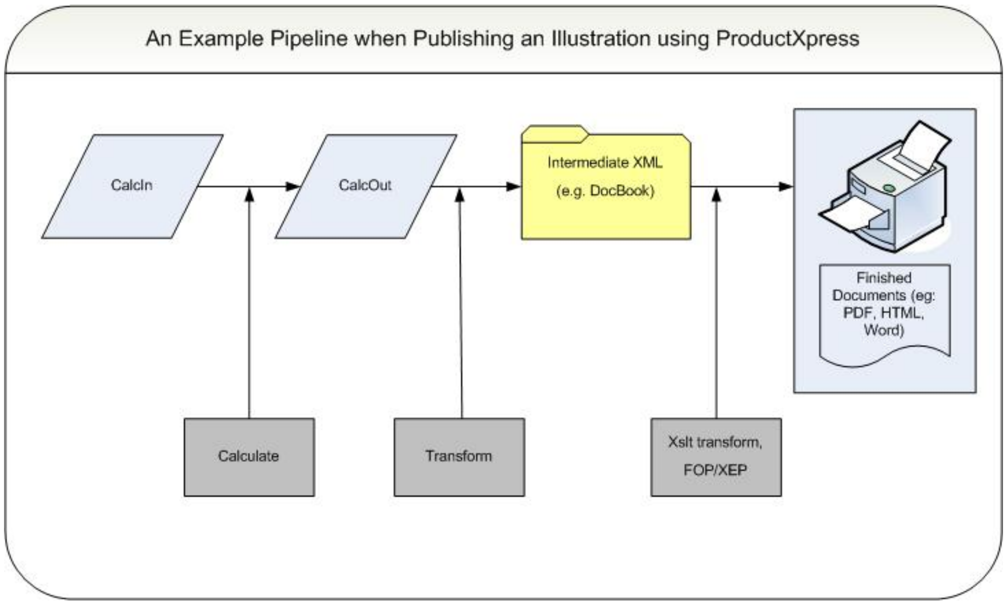
The publishing infrastructure of ProductXpress allows customers to easily design and produce high-quality documents concerning products that they have created.

It also allows the publication of illustration[2] documents (both inforce and new business), and policies.

The ProductXpress publishing pipeline is highly configurable and uses industry-standard formats to render to all prevalent output formats (PDF, Word, HTML), which makes it easy to hook up to existing in-house and commercial document solutions.

When publishing documents about data, there are usually various output-formats required based on that same data. (e.g. an HTML report, a PDF document, an Excel spreadsheet and a Word document). Additionally, this data is enriched with extra information from various sources (e.g. styling information, images). Due to these requirements, a pipeline must be established consisting of different transformations, starting with the extraction of the data and ending with the production of the document.

When publishing an Illustration using ProductXpress, the following pipeline is typical:


An Example Pipeline when Publishing an Illustration using ProductXpress
CalcIn
CalcOut
Intermediate XML (e.g. DocBook)
Finished Documents (eg: PDF, HTML, Word)
Calculate
Transform
Xslt transform, FOP/XEP

The steps in this publishing process are independent of the software packaging in which it takes place.

What Happens in Each Step?

A calculation input document is transformed into a calculation output document. The calculation output document must contain all of the instance data (including calculation results) that is required to produce the desired output at the end of the pipeline.   This step is performed using the ProductXpress Calculator’s "Calculate" call.

The calculation output document is transformed into an intermediary document. Typically, the resulting document is in some standard format used for publishing (e.g. DocBook), or it is an application specific format (e.g. WordML or SpreadsheetML).

At the end of this step, the information (i.e. the contents of the document) is typically complete. This step is performed using the ProductXpress Calculator's "Transform" call.

The final step(s) typically involve transforming the data into a more detailed, printable document. (e.g. first FOP, and then PDF). Standard pipelines and tools are available for this. (E.g. the DocBook pipeline).

The only enrichment that typically takes place in these steps tends to be formatting and layout related (e.g. adding styles, images, page-numbers, Table Of Content generation).This step can be performed using, for example, FOP or XEP – which are engines for rendering finished documents.

Other Pipelines

While the previous pipeline is typical, it is not necessarily the only possible pipeline for runtime publishing. For example, it's possible that a transformation takes place **before** the calculator step in the pipeline, like a transformation from an Origo message to a calculation input document.

Another possibility is to set up a (very short) pipeline that turns CalculationInput data directly into SpreadsheetML (the data-format used by Excel).

ProductXpress can support any number of pipelines (at the same time). In addition, all of the steps in a ProductXpress publishing pipeline are configurable. This includes the Calculate and Transform steps that are executed by the ProductXpress Calculator.

Configuration

The output produced using the ProductXpress publishing pipeline can be controlled in two ways:

- By providing different templates – this is typically done in order to change the contents of the published documents
- By providing different pipelines – this is typically done in order to change the output format of the published documents

Contents

The sample ProductXpress publishing pipelines are entirely template driven.

In order to change the contents and formatting of a document, all that is required is to change the templates that are used by the publishing pipeline.

The templates use the industry-standard XSLT format, making them easy to customize.

Output format

Pipelines are defined using a pipeline script. This script lists the steps that must be executed to produce a finished document.

Pipelines are available for PDF, Word, Excel and HTML.

New pipelines can easily be added, and because it is scripted, external tools like FOP or XEP can easily be invoked from the pipeline.

Scope

This section provides information regarding scope items for publishing.

Document Formats

Out of the box, ProductXpress provides sample pipelines for producing PDF documents, Word documents and HTML documents. However, it is also possible to produce other types of documents.

Applications

The publishing infrastructure in ProductXpress is designed to easily integrate into Web-servers, and into existing publishing pipelines. Example integrations are provided.

Performance Optimization

The performance of a ProductXpress calculation is influenced by many factors:

- The type of product
- The type of calculations that it must perform
- The platform that the Calculator runs on
- The client application that makes use of the Calculator
- The way in which this client application makes use of the Calculator
- The network infrastructure

This multitude of factors means that it is impossible to recommend rules that will always work in every situation. However, this section contains a set of recommendations that helps achieve optimal performance of the Calculator.

Some of the recommendations in this section are dependent on the version of the ProductXpress application that is being used. The ProductXpress application suite is constantly being tuned for performance, and some of the optimizations that this guide recommends the user to perform will be performed automatically by future versions of the ProductXpress application suite.

Performance Profiling

In order to gain more insight into the performance of ProductXpress Calculator, several things can be done using existing tools to obtain profiling data.

PxTest Program

Regardless of the status of the actual Calculator integration with the production system, independent tests can be performed for a deployment package utilizing the test tool `PxTest`, which is delivered with Calculator. This tool simulates the client or production system to invoke the Calculator with a request.

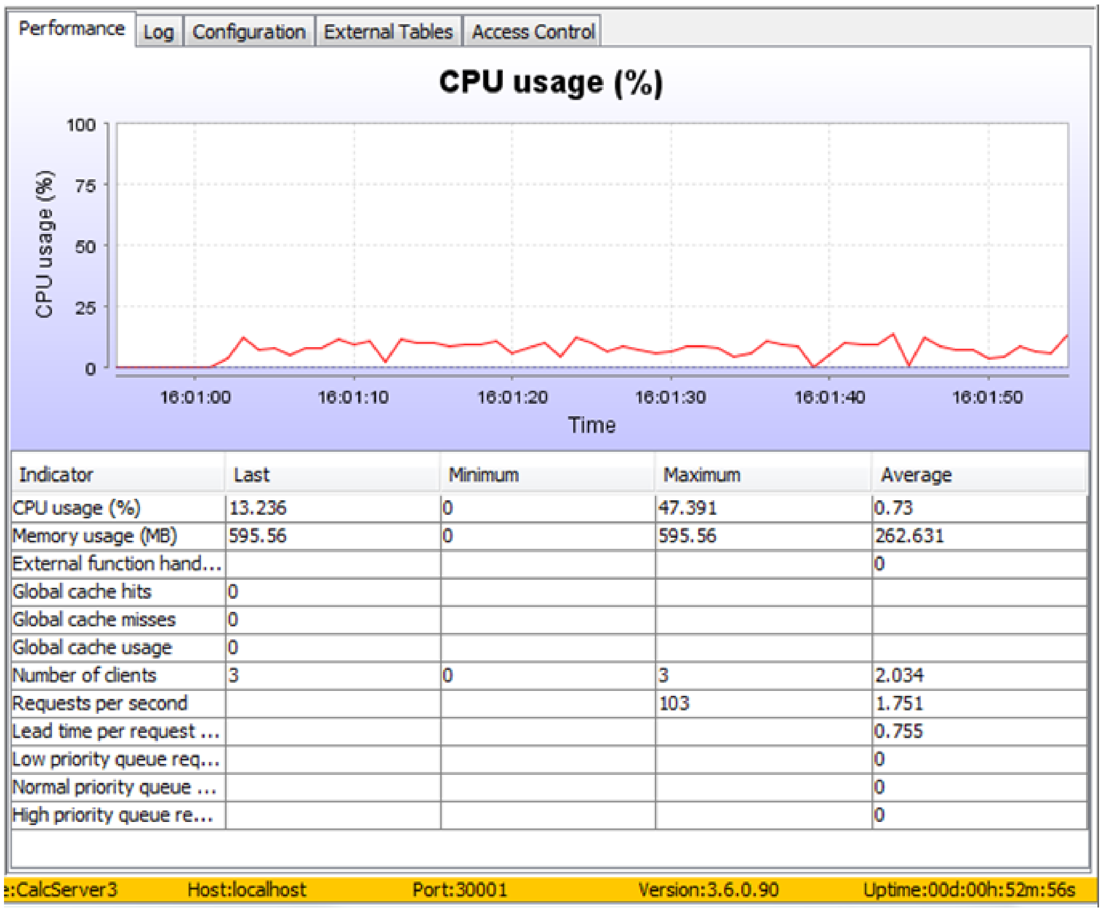
PxTest uses the SOAP protocol. If other communication methods are required, the CalcServerTest tool can be used, for example to evaluate performance of the products on the production hardware, even before integration has taken place with the host system(s).

Calculator Console

The Calculator Console has lots of performance indicators that can be monitored. These can provide insight into bottlenecks that degrade the performance of the (Service) Calculator. Some examples are:

- CPU load
- Memory usage
- Requests per second
- Number of clients

The output can be viewed in the Performance tab of the console as follows:



Indicator	Last	Minimum	Maximum	Average
CPU usage (%)	13.236	0	47.391	0.73
Memory usage (MB)	595.56	0	595.56	262.631
External function hand...				0
Global cache hits	0			
Global cache misses	0			
Global cache usage	0			
Number of clients	3	0	3	2.034
Requests per second			103	1.751
Lead time per request ...				0.755
Low priority queue req...				0
Normal priority queue ...				0
High priority queue re...				0

:CalcServer3 Host:localhost Port:30001 Version:3.6.0.90 Uptime:00d:00h:52m:56s

For more information on the Calculator Console and its performance indicators, refer to the *Calculator Console Online Help*.

Profiling Calculation Requests

By specifying the optional 'timer' attribute (`<CalculationInput timer="true"/>`) in the input section of the Calculation Request XML, the CalculationOutput contains some Calculator internally measured times. These times can also be used for understanding the behaviour and then tuning for performance optimization:

- receive-time - the time spent to process the CalculationInput XML
- queue-time - the time spent in the Calculator queue
- calculation-time - the time spent to calculate the requested calculations
- external-function-time - the time spent in external functions during the calculation

Profiling Example

The following is an example of where to place the timer attribute and a possible output.

```
<?xml version='1.0'?>
<clc:CalculationInput
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
```

```
xmlns:clc='http://www.solcorp.com/ns/ProductXpress/CalculationInpu
tOutput/CalculatorElement' xmlns='Testlab_Deployment'
timer="true">>
```

Example XML output:

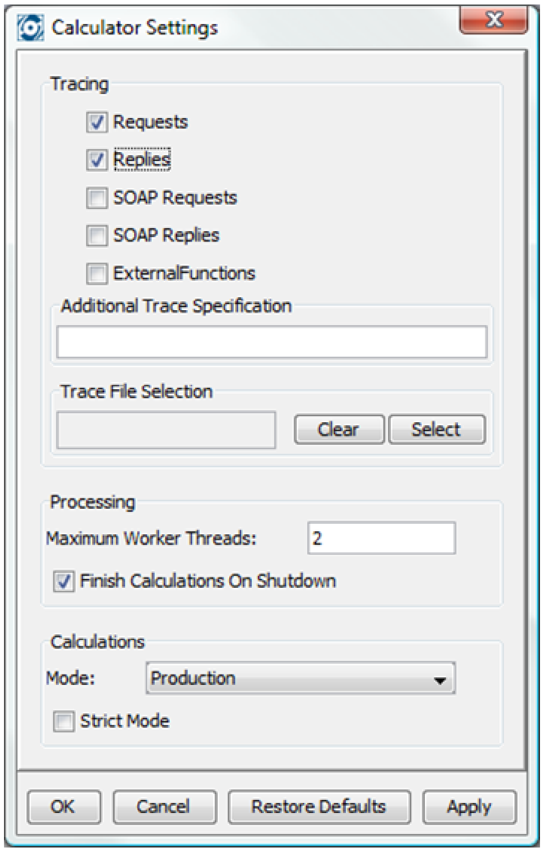
```
<CalculationOutput receive-time ='0.02' queue-time='0.01'
calculation-time='0.12' external-function-time ='0.3'>
```

For more information, refer to the *Calculator Reference Guide*.

Tracing

The actions performed by the Calculator can be logged using the tracing mechanism. As expected, tracing should not be activated in production situations as this can actually cause overall performance degradation; however it serves well to zoom into specific areas of investigation.

Tracing can be switched on through the Calculator Console using the CALCULATOR SETTINGS menu option as shown below:



- [✓] Requests
- [✓] Replies
- [ ] SOAP Requests
- [ ] SOAP Replies
- [ ] ExternalFunctions

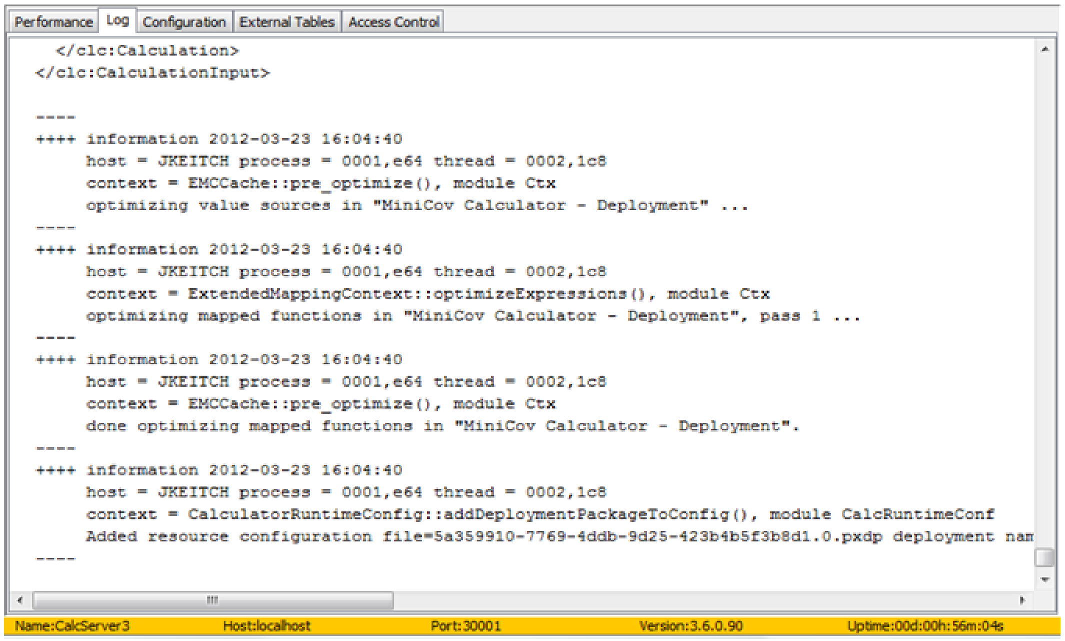
Maximum Worker Threads: 2

- [✓] Finish Calculations On Shutdown

Mode: Production

- [ ] Strict Mode

The tracing output is then directed to the Log Tab of the Calculator Console as shown below:



```
  </clc:Calculation>
</clc:CalculationInput>

----
++++ information 2012-03-23 16:04:40
     host = JKEITCH process = 0001,e64 thread = 0002,1c8
     context = EMCCache::pre_optimize(), module Ctx
     optimizing value sources in "MiniCov Calculator - Deployment" ...
----
++++ information 2012-03-23 16:04:40
     host = JKEITCH process = 0001,e64 thread = 0002,1c8
     context = ExtendedMappingContext::optimizeExpressions(), module Ctx
     optimizing mapped functions in "MiniCov Calculator - Deployment", pass 1 ...
----
++++ information 2012-03-23 16:04:40
     host = JKEITCH process = 0001,e64 thread = 0002,1c8
     context = EMCCache::pre_optimize(), module Ctx
     done optimizing mapped functions in "MiniCov Calculator - Deployment".
----
++++ information 2012-03-23 16:04:40
     host = JKEITCH process = 0001,e64 thread = 0002,1c8
     context = CalculatorRuntimeConfig::addDeploymentPackageToConfig(), module CalcRuntimeConf
     Added resource configuration file=5a359910-7769-4ddb-9d25-423b4b5f3b8d1.0.pxdp deployment nam
----
```

Name:CalcServer3 Host:localhost Port:30001 Version:3.6.0.90 Uptime:00d:00h:56m:04s

The Trace File in this example is called ClcTrace.out and can be found in C:\Windows.

Reference

- *Calculator Console Online Help* - Section Performance Indicators and Enable a Trace Option.
- *Calculator Reference Guide* - Common Configuration and Maintenance, section Tracing.
- *Calculator Reference Guide* - Calculator Input-Output Specification for the Push Calculator section XML Reference.

Improving Performance

In order to get the best performance out of ProductXpress Calculator, there are some rules that are good to follow in general. Other tips to improve performance are more specific to a particular situation.

Appendix A discusses detailed troubleshooting steps for performance investigations.

Ensure Adequate Hardware

To reduce time and effort when implementing a product, ProductXpress provides flexibility. However, it also comes at a performance cost, which can be addressed by using faster hardware.

The cost of purchasing and maintaining adequate hardware is relatively low, whereas the cost of implementing products is relatively high.

If the same calculations are hard-coded in C++ or COBOL and they are performed on the same equipment that Calculator runs on, expect the Calculator to be slower in most cases. Due to the built-in optimizations that the Calculator performs (e.g. caching), this is not always the case.

Ensure Adequate Network Infrastructure

A low network speed or a high latency between the client application and the Calculator can result in slow performance. This can be addressed by ensuring that the machine that Calculator runs on is located close (in terms of network latency) to the machine on which the client application runs.

Ensure ProductXpress is Used as Intended

As with all services, the ProductXpress architecture has a large call overhead compared to calling a function in the same environment (such as calling a C++ function from within a C++ program). This means that the number of calls to the Calculator should be minimized and the amount of work done by the Calculator per call should be maximized.

Allocate Time for Optimization of Projection Calculations

When calculations are implemented that do forward projections, like illustration calculations, they typically do a lot of calculation work. It can be useful to spend a limited amount of time optimizing these calculations, once they are functionally complete.

Write Clean, Simple, Short Calculations

Most calculation optimizations ensure that the same thing is only calculated once, and that expressions are written as simply as possible. This is a good principle for defining calculations in any case, as this will simplify the maintenance for calculations.

Don't hesitate to use two functions instead of one, especially if one of those functions can be used somewhere else.

Specific Measures for Scalability

ProductXpress Calculator can utilize multiple CPUs. It can also run on multiple machines under cluster management software.

If the performance indicators discussed previously indicate that the machine on which the Calculator is running receives lots of requests (maybe from multiple clients) resulting in a high CPU load, extra hardware in the form of multiple CPUs or multiple machines may be a valuable and affordable option.

In the case of multiple machines or CPUs, utilization of the Calculators batch request functionality is a good measure to improve performance.

The general idea of batch requests is that a client merges several Calculator input requests into one big request. Each 'sub-request' gets an ID, which needs to be realigned with the calculation results once the Calculator returns.

Sending Batches to the Calculator

This section provides information regarding sending batches to the Calculator.

Concurrent Calls to Calculator

- Another option is to enable multiple clients to call the Calculator with multiple concurrent socket connections to the same port number. The batch request is split into multiple sub-batches and these are then sent to Calculator concurrently.

  The Batch splitting mechanism is comparatively inefficient compared with this option. The reason for this is that in the former case, only one result file is produced. Therefore all internally initiated parallel thread results need to be combined before sending the single result file. So there can be waiting if a particular calculation takes much longer compared with the other smaller less complex calculations.

- Therefore, for maximum throughput to process a large number of calculation requests, it is best to split the input batch into smaller XML documents and then call Calculator in parallel. The best practice is to have as many parallel client connections as the number of CPUs on the machine (there can be one more or to handle client latency) and check that 100% CPU time is being utilized.

- The exact number of independent calculation requests in each XML document varies with the complexity of the product, the request size, memory, etc. and therefore tuning this is a custom optimization effort.

Aggregation of calculation results is another important means of speeding up performance. When a client contains multiple connections to the Calculator that share the same inputs, but request values for different features, it is good practice to try to combine the requests in one merged request. In this way, the number of calls to the Calculator is minimized.

ⓘ The PxTest program has extensive capabilities to support users performing concurrent and batch calls to Calculator.

Reference

- *Calculator Reference Guide* - section Operational Settings Files.

Appendix A - Performance Tuning

This Appendix provides an overview of the parameters that may affect the speed of calculations and suggestions to help you while fine tuning performance.

Before You Start

The performance of Calculator may be affected by a number of parameters, ranging from the actual calculations to the software/hardware involved.

The best moment to fine tune the performance of Calculator is when the development phase is relatively stable or final. The reasons behind this are twofold:

- It is more unlikely that actual mistakes occur in calculations during the final stages, where what seemed initially as a performance problem is actually expected for the requested calculation.
- If the calculations are not completed yet, and extra ones will be added later, optimizations and fine tuning during the earlier stages will be not relevant or obsolete at later stages.

Furthermore, before proceeding with any performance tuning actions, it is highly recommended to ensure that all the results of the calculations of your products, are the expected ones (to the required accuracy and rounding) for all the test cases that you intend to run against them.

How to Begin

When the above preconditions are satisfied, the first thing to do is to analyze the performance bottleneck. Do not make assumptions about what is causing the problem until you have verified the cause. Most of the time, the bottleneck is somewhere else than you initially think it is.

Factors Influencing Performance

In the ProductXpress architecture, there are three distinct steps that take place when a calculation is being performed:

1. The call from the client application to the Calculator.
2. The data processing that takes place within the Calculator.
3. The calculation inside the Calculator.

Each of these steps can form a performance bottleneck. When a product is performing badly, all three steps must be checked.

The bottlenecks differ for different kinds of usages of the Calculator. For an administration system, it is typically the call to the Calculator and the data processing that are the bottleneck. For an illustration system it is typically the calculation itself that forms the bottleneck (although data-processing can play a role here as well).

Analysis

The Calculator is able to provide some timing information that allows an initial analysis of the performance on the side of the Calculator. In order to turn this on, the client application needs to provide a timer flag with a calculation request. For more information, see the *ProductXpress Calculator Integration Guide*.

The time spent between the client application and the Calculator must be measured by the client application itself.

The Call from the Client to the Calculator

The call from the client to the Calculator can be a bottleneck when many calls need to be performed serially. This is typically the case when a batch calculation is performed.

There are a number of factors that influence the speed of calling the Calculator.

A call takes place in the following phases:

1. Building the XML by the client application

2. Sending the XML to/ receiving it from the Calculator

3. Parsing the results.

Each of these phases can contain a bottleneck. In addition, the amount of calls itself can be minimized.

Minimizing Number of Calls Between Client and Calculator

The ProductXpress architecture is most efficient when the number of calls that need to take place between the client application and the Calculator is minimized.

In batch applications, this can be achieved by performing calculations for more than one policy per calls.

For illustration applications, this can be achieved by calculating the entire projection for an illustration in one go, rather than making a call for each iteration in the illustration.

For more information, refer to the *ProductXpress Calculator Integration Guide*.

Building the XML by the Client Application

The fastest way to build the XML document that is sent to the Calculator is to build it natively in the client.

Using xslt (or similar tools) is a way to speed up the implementation process when the client application has an existing XML format, but there is a significant performance penalty to using xlst processors.

The implementation time saved by building xslt is not usually very large. So it is recommended to build the XML natively instead.

Sending the XML to, Receiving it from, the Calculator

The time it takes for the XML to be sent to the Calculator and the time it takes to get back, depends largely on the network speed and the network latency.

For batch processes, it can help to have multiple threads sending batch requests to a Calculator at the same time, especially if the Calculator is running on a multi-CPU machine.

Parsing the Results

As with building the XML, parsing the XML is best done natively, and not through xslt.

Data Processing

This section provides information regarding data processing.

Minimize Redundant Data

When the same instance data is used in a calculation, it is more efficient to send this only once. This both minimizes the data transmission and parsing time, and it allows the Calculator to use its caches more efficiently.

For example:

A UL product has a single link to Basic UL Coverage which has a multiple link to withdrawal.

It’s more efficient to make the Policy Issue Date attribute part of the UL product element than to make it part of the withdrawal element.

Simple Data Structure

It is faster to have a data structure with links of single multiplicity than one with links of many multiplicity. If a product can be split out into multiple explicit structure elements, rather than a single generic structure element that is linked with a multiple link, the data processing will be faster.

For example:

Product A can have two riders: Rider X and Rider Y. This can be modeled in two ways:

1. Product A links to a module called ‘Rider’ through a multiple link. Rider X is distinguished from Rider Y by an attribute on the module called ‘Rider Type’.

2. Product A links to a module called ‘Rider X’ through a single link and to a module called ‘Rider Y’ through another single link.

Model number 2 is faster than model number 1.

Calculated Attributes

In this version of ProductXpress, when a calculation is performed through a calculated attribute, some of the functions that have been calculated by this attribute are not cached the next time a similar attribute is calculated.

This means that it is most efficient to perform a calculation at the highest level possible in the product structure and avoid the use of calculated attributes.

ⓘ The calculated attribute value itself *is* cached.

Calculation Optimization

When a calculation is slow, this is most often the result of a piece of the calculation that is done 100s or 1000s of times per calculation.

As with the higher level optimization strategy, it is important to find out where the bottleneck in a calculation is, prior to making changes to the calculation.

It is very useful to have a set of test cases **and** test sets available, so that any changes that are made in the calculation can be tested against them immediately, to verify that the changes do not make a difference in calculation results.

Most calculation optimizations ensure that the same thing is only calculated once, and that expressions are written as simply as possible. This is a good principle for defining your calculations in any case, as this will simplify the maintenance for your calculations.

Analysis

As previously mentioned, a slow calculation is usually the result of a loop (for example, when a sum function is used somewhere, or when a recursive function has been created). The first thing to do when analyzing is to find where this loop is. Quite often, there is more than one loop nested within another loop.

Usually, the person who has created the expression knows where the calculations perform loops. Another way of finding loops is by examining the functions through the component usage editor, or by looking at the intermediate results for a calculation.

If you have difficulty finding the bottlenecks in a calculation, there are some tracing settings that may help you find these bottlenecks. The tracing information can be very voluminous, so this should be a last resort.

Once you've found the loops, you only need to look at the functions that are being used inside the loops. Functions that are not being used in the loop are not part of the bottleneck and so optimizing them will not provide a lot of benefit. Break up recursive expressions

Ensure that recursive expressions can be cached as much as possible. Use as few attributes and arguments as possible inside the inner loop. Consider for example:

```
f(t)=f(t-1)+(1/interest)^t*StartAmount, where StartAmount is an attribute.
```

Rewriting this as   f(t)=StartAmount*g(t)

```
g(t)=g(t-1)+(1/interest)
```

Ensures that it is much easier to cache g(t), because it's dependent only on interest, which typically has only 1 or 2 values for a set of policies, whereas startamount can have hundreds of values. Each time the loop has to be recalculated requires a lot of performance.

An additional advantage is that your expressions will be smaller and easier to read.

Simplify Expressions

The simpler the expression, the faster the calculation.   One example of this is the following.

Two expressions need to be calculated: NKpre and NKpost. They are defined as follows:

```
NKpre=sum(t=0 to 100) of something^t

NKpost=sum(t=1 to 100) of something^t
```

If NKpre is simplified as follows:

```
NKpre = 1+Nkpost
```

Then the total calculation time for NKpre and NKpost will be almost halved. In addition, the expressions are simpler, and if 'something' needs to change, it only needs to be changed in NKpost.

Work in whole months, if possible, rather than 18.1666667 years.

Bring attributes to as high a level of the expression tree as possible (use arguments instead). Most importantly, bring any 'calculation date' dependencies to as high a level of the tree as possible. This can have a major impact on caching efficiency. It also helps you spot inadvertent mistakes in the use of 'calculation date' more easily.

Arguments are faster than using a cached sub-expression several times. Arguments are also faster than calculating the same piece of expression several times.   For example:

Expr_slow: (1+months/12)*expr1+months/12*expr2

Can be rewritten as

Expr_fast1(years:real): (1+years)*expr1+years*expr2

Expr_Fast2:Expr_fast1(months/12)

If possible, use the operator ‘in’ instead of using the internal function ‘Exists’.

For example: Expr_slow:

$$(\exists_x |x \in setOfMonths: x = someFunction())$$

For example: Expr_fast:

(someFunction() Î setOfMonths)

ProductXpress Calculator doesn’t cache functions that pass tuples as an argument. As a result, the function will be evaluated over and over again. For better performance, instead of passing a tuple as an argument, create additional parameters for the function of type simple value definition, corresponding to each tuple member.

For example: Expr_slow:

SomeFunction(FundCharacteristicsTuple)

For example: Expr_fast:

SomeFunction(Fund Charges, Fund Spread)

If possible, optimize the expression such that fewer mathematical operators are used.

For example: Expr_slow:

$$\frac{\left(1-\frac{x}{100}\right)}{\left(2-\frac{y}{100}\right)}$$

For example: Expr_fast:

$$\frac{(100-x)}{(200-y)}$$

The internal functions ’first’ & ‘rest’ are resource intensive. If possible, achieve the same functionality using the internal function ‘condsum’ or ‘condset’.

For example: Expr_slow:

TotalOfPremium(setOfPremiums) =

$$\begin{cases} first(setOfPremiums)+TotalPremium(rest(setOfPremiums)) & setOfPremiums \mathrel{!=} \phi \\ 0 & setOfPremiums = \phi \end{cases}$$

For example: Expr_fast:

TotalOfPremium(setOfPremiums) =

$$\sum_{x,\, x \in setOfPremiums} (x)$$