Document Title:	HNG-X Counter Application High Level Design
Document Type:	Design (DES)
Release:	Not Applicable
Abstract:	Top level HLD for the HNG-X Counter.
Document Status:	APPROVED
Author & Dept:	Alex Robinson (revised by Clare Keane)
External Distribution:	None
Security Risk Assessment Confirmed	YES

Approval Authorities:

Name	Role	Signature	Date
Steve Evans	Solution Design		
Graham Allen	Development		
Andy Thomas	Architecture		

Note: See Post Office Account HNG-X Reviewers/Approvers Role Matrix (PGM/DCM/ION/0001) for guidance.

Documents are uncontrolled if printed or distributed electronically. Please refer to the Document Library or to Document Management for the current status of a document.

0 Document Control

0.1 Table of Contents

0.2 Figures

0.3 Document History

Version No.	Date	Summary of Changes and Reason for Issue	Associated Change - CP/PEAK/PPRR Reference
0.1	11/06/07	Initial Draft	N/A
0.2			
0.3 (not reviewed)	6/02/08	Document updated with feedback from Alex Robinson	
0.4 (not reviewed)	24/10/08	Document updated to reflect changes in system. Changes were made to the document for Acceptance by Document Review with the insertion of the Section containing the table of cross references for Acceptance by Document Review.	
0.5	02/11/2009	Major update and revamp for review. (Note no change markers included as so much change in this version)	
0.6	17/11/2009	Comments included ready for approval	
1.0	18/11/2009	Marked approved.	
1.1	08/07/2010	Revised for release 2 changes for PING	CP0367_4914
1.2	23/8/2010	Includes comments received from review. Specifically, added description of AppControl.	
2.0	23/8/2010	Approved	

0.4 Review Details

Review Comments by :	
Review Comments to :	Clare.Keane GRO & RMGADocumentManagement GRO

Mandatory Review	
Role	Name
Solution Design	Jon Hulme
Architecture	Andy Thomas
Solution Design	Steve Evans
SSC	Steve Parker
Business Continuity	Adam Parker
LST Manager	Sheila Bamber
SV&I Manager	Chris Maving

Optional Review	
Role	Name
Chief Information Security Officer	Tom Lillywhite
Security & Risk Team	CSPOA.Security GRO
HNG-X R1 Programme Manager	Geoff Butts
Applications Architecture	Pete Jobson

System Qualities Architecture	Dave Chapman
Architect	Jason Clark
Security Architect	Tom Lillywhite
Test Design	George Zolkiewka
Service Director	Gaetan van Achte
HNG-X Service Change & Transition	Graham Welsh
Head of Service Operations	Tony Atkinson
Service Network	Ian Mills
Data Centre Migration	Vince Cochrane
Integration Team Manager	Peter Okely
Testing Manager	Debbie Richardson
POL Test Manager	James Brett
Core Division	Ed Ashford
Core Division	Andrew Gibson
Business Architect	Gareth Jenkins
Development	Graham Allen
Solution Design Architect	Sarah Selwyn
Software & Solution Design Developer	Stuart Honey
Service Manager - Retail and RMGA	Claire Drake
Chief Information Security Officer	Tom Lillywhite
Issued for Information – Please restrict this distribution list to a minimum	
Position/Role	Name
Development	Martin Day
Development	Phil Tomlinson

(*) = Reviewers that returned comments

0.5 Acceptance by Document Review

No sections in this document have been identified to POL as comprising evidence to support Acceptance by Document review (DR).

0.6 Associated Documents (Internal & External)

Reference	Version	Date	Title	Source
PGM/DCM/TEM/0001 (DO NOT REMOVE)			Fujitsu Services Post Office Account HNG-X Document Template	Dimensions
PGM/DCM/TEM/0002 (DO NOT REMOVE)			Fujitsu Services Post Office Account HNG-X Landscape Document Template	Dimensions
DES/GEN/MAN/0001			HLD Designer Guidelines	Dimensions

ARC/GEN/REP/0001			HNG-X Glossary	Dimensions
ARC/APP/ARC/0006			Online Internal Applications Architecture	Dimensions
ARC/APP/ARC/0009			HNG-X Counter Business Applications Architecture	Dimensions
DES/APP/HLD/0035			Exceptions and Logging Framework High Level Design	Dimensions
DES/APP/HLD/0038			Peripherals Subsystem High Level Design	Dimensions
DES/APP/HLD/0039			Presentation Subsystem High Level Design	Dimensions
DES/APP/HLD/0040			Interaction Subsystem High Level Design	Dimensions
DES/APP/HLD/0041			Business Logic Subsystem High Level Design	Dimensions
DES/APP/HLD/0042			Business Services Subsystem High Level Design	Dimensions
DES/APP/HLD/0043			Communications Subsystem High Level Design	Dimensions
DES/APP/HLD/0045			Reference Data Subsystem High Level Design	Dimensions
DES/APP/HLD/0046			Business Data Subsystem High Level Design	Dimensions
DES/APP/HLD/0053			HNG-X BAL Reporting High Level Design	Dimensions
DES/APP/HLD/0057			HNG-X Counter Infrastructure Service and Process Control High Level Design	Dimensions
DES/APP/HLD/0058			UCR Document for Banking Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0059			Branch Accounting	Dimensions
DES/APP/HLD/0060			UCR Document for Branch Administration Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0061			UCR Document for Branch Support Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0062			UCR Document for Bureau de Change Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0063			UCR Document for Cash & Stock Management Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0064			UCR Document for E-Top-Up Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0065			UCR Document for In and Out Payments Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0066			UCR Document for Postal Services Use	Dimensions

			Case Barrel High Level Design	
DES/APP/HLD/0067			UCR Document for Retail and Stock Sales Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0068			UCR Document for Shared Use Case Barrel High Level Design	Dimensions
DES/APP/HLD/0069			HNG-X AP-ADC High Level Design	Dimensions
DES/APP/HLD/0076			Process Engine (PDL)	Dimensions
DES/APP/HLD/0083			HNGX Counter Subsystem : Recovery Management	Dimensions
DES/APP/HLD/0094			HNG-X End-to-End Key Usage Overview High Level Design	Dimensions
DES/APP/HLD/0096			HNG-X Applications Training System High Level Design	Dimensions
DES/APP/HLD/0102			HNG-X Counter Help Design	Dimensions
DES/APP/HLD/0109			HNG-X Counter Reporting - High Level Design	Dimensions
DES/APP/HLD/0110			HNG-X Counter Applications: Transaction Benchmarking High Level Design	Dimensions
DES/APP/HLD/0112			HNG-X Counter Applications: Online Financial Applications High Level Design	Dimensions
DES/APP/HLD/0113			HNG-X HLD - BRDB Processing of the In Day Migration Data	Dimensions
DES/APP/HLD/0114			HNG-X Counter Applications: Online Services Definition Configuration and Usage High Level Design	Dimensions
DES/APP/HLD/0123			HNG-X HLD - Settlement Functions	Dimensions
DES/APP/HLD/0126			HNG-X Branch Accounting Counter High Level Design	Dimensions
DES/APP/HLD/0127			HNG-X Branch Accounting BAL High Level Design	Dimensions
DES/APP/HLD/0128			Postal Services Reference Data Maintenance and Counter Support High Level Design	Dimensions
DES/APP/HLD/0136			HNG-X Cash and Stock Management High Level Design	Dimensions
DES/APP/HLD/0137			HNG-X Counter Printing	Dimensions
DES/APP/HLD/0141			HNG-X User Interface Components High Level Design	Dimensions
DES/APP/IFS/0006			RAC Message Flows in HNGX	Dimensions
DES/APP/IFS/0012			BAL Service Interface Specification	Dimensions
DES/APP/IFS/0018			XML Message Audit between Counter and BAL/OSR	Dimensions

DES/APP/LLD/0042			HNG-X XML PARSER/SERIALISER LOW LEVEL DESIGN	Dimensions
DES/APP/MAN/0001			HNG-X: Type A / B Reference Data Counter Object Definitions	Dimensions
DES/APP/STD/0001			HNG-X UI Style Guide	Dimensions
DES/GEN/MAN/0002			APADC Reference Manual	Dimensions
DES/GEN/SPE/0003			Counter Type X Reference Data Definitions	Dimensions
DES/GEN/SPE/0007			HNG-X Menu Hierarchy and Messages	Dimensions
DES/GEN/SPE/0008			HNG-X Counter Application High Level Design	Dimensions
DES/GEN/SPE/0009			HNG-X Receipts, Slips and Labels	Dimensions
DES/GEN/SPE/0010			HNG-X Banking, Debit Card and ETopUp Receipts and Texts	Dimensions
DES/GEN/SPE/0011			HNG-X AP and ADC Receipts	Dimensions
DES/SYM/HLD/0021			Branch Access Layer Management High Level Design	Dimensions
DEV/GEN/MAN/0003			HNG-X UI Construct Catalogue	Dimensions
DEV/INF/LLD/0113			CNIM2 Low Level Design	Dimensions
DEV/APP/LLD/0192			HNG-X Tokens System Low Level Design	Dimensions
PGM/DCM/ION/0001			HNG-X Document Reviewers/Approvers Role Matrix	Dimensions
PGM/DCM/STD/0001			HNG-X Documentation and Record Standard	Dimensions
REQ/GEN/PRO/0001			HNG-X Requirements and Acceptance Procedures	Dimensions
DEV/APP/LLD/0163			Stock Unit Balance Report Low Level Design	Dimensions
DEV/APP/LLD/0164			Branch Trading Statement Low Level Design	Dimensions
DEV/APP/LLD/0178			Migration Report Low Level Design	Dimensions
DEV/APP/LLD/0179			Counter Reports Low Level Design	Dimensions
DEV/APP/LLD/0180			HNG-X Reporting Accounting Low Level Design	Dimensions
DEV/APP/LLD/0182			HNG-X Recovery Low Level Design	Dimensions
DEV/APP/LLD/0184			HNG-X Help Viewer Low Level Design	Dimensions
DEV/APP/LLD/0185			HNGX-X Help Steps Low Level Design	Dimensions
DEV/APP/LLD/0187			AP-ADC Low Level Design	Dimensions
DEV/APP/LLD/0188			HNG-X Counter Applications: Banking and Online Financial Framework Low	Dimensions

			Level Design.	
DEV/APP/LLD/0189			HNG-X Counter Applications: DCS Low Level Design	Dimensions
DEV/APP/LLD/0190			HNG-X Counter Applications: ETU Low Level Design	Dimensions
DEV/APP/LLD/0191			HNG-X Counter Applications: PINPad Low Level Design	Dimensions
DEV/APP/LLD/0192			HNG-X Tokens System Low Level Design	Dimensions

See also Section 2.

Unless a specific version is referred to above, reference should be made to the current approved versions of the documents.

0.7 Abbreviations

See also ARC/GEN/REP/0001

Abbreviation	Definition
BAL	Branch Access Layer
BDO	Business Data Object
BLO	Business Logic Object
CD	Configuration Data
DCS	Debit Card System; service that supports payment by Debit Card
ETS	Electronic Top Up Service
HLD	High Level Design
JSN	Journal Sequence Number
LLD	Low Level Design
NBS	Network Banking System
NIC	Network Interface Component
RD	Reference Data
SSN	Session Sequence Number
TPS	Transaction Processing System
TSN	Transaction Sequence Number
UIA	User Interaction Agents
UIE	User Interaction Elements
UML	Unified Modelling Language
USN	Uniqueness Sequence Number
VOPC	View of Participating Classes

0.8 Glossary

See also ARC/GEN/REP/0001.

Term	Definition

0.9 Changes Expected

Changes
None for Counter Release 1 (other than by comments received)

0.10 Accuracy

Fujitsu Services endeavours to ensure that the information contained in this document is correct but, whilst every effort is made to ensure the accuracy of such information, it accepts no liability for any loss (however caused) sustained as a result of any error or omission in the same.

0.11 Security Risk Assessment

Security risks have been assessed and it is considered that there are no security risks relating specifically to this document.

1 Purpose

The purpose of this document is to specify the high level design of the HNG-X Counter. However, the design of the counter is split across a number of documents – see section 2 for details.

This document should be read in conjunction with the corresponding UML model, which consists of a number of

- Class diagrams
- Sequence diagrams

Section 2 shows the position of this framework HLD in the document set and its relationship to other documents.

2 Scope

The scope of this document is to be the Top-level HLD for the HNGX counter.

It follows on from the counter architecture document (ARC/APP/ARC/0009).

The diagram below shows the positioning of this document in the HLD structure.

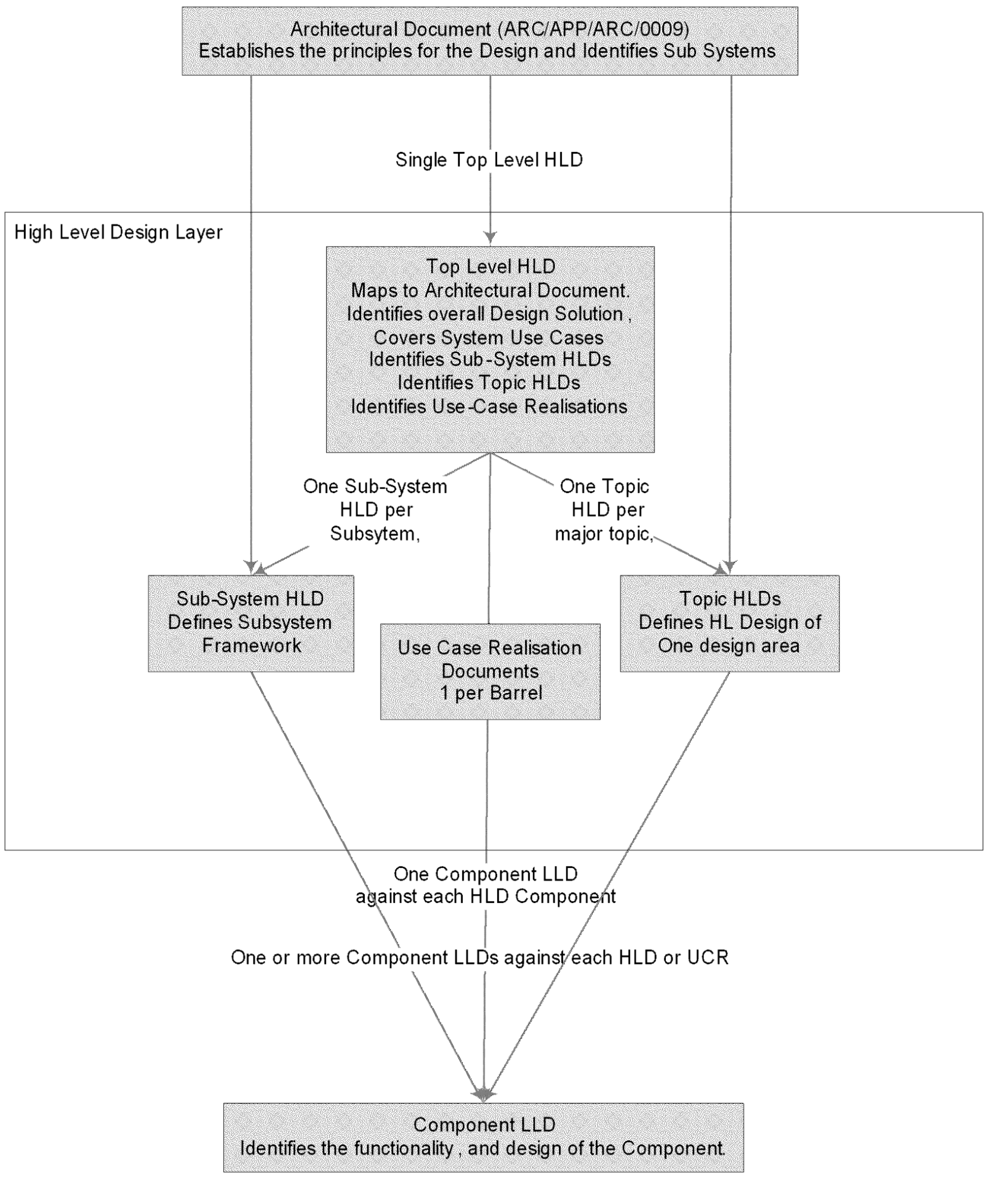

Architectural Document (ARC/APP/ARC/0009)
Establishes the principles for the Design and Identifies Sub Systems
Single Top Level HLD
High Level Design Layer
Top Level HLD
Maps to Architectural Document.
Identifies overall Design Solution ,
Covers System Use Cases
Identifies Sub-System HLDs
Identifies Topic HLDs
Identifies Use-Case Realisations
One Sub-System HLD per Subsytem,
One Topic HLD per major topic,
Sub-System HLD
Defines Subsystem Framework
Use Case Realisation Documents
1 per Barrel
Topic HLDs
Defines HL Design of One design area
One Component LLD against each HLD Component
One or more Component LLDs against each HLD or UCR
Component LLD
Identifies the functionality , and design of the Component

Figure 2-1 – Document Set

Although this document is the central point of the counter design, not all the counter design is in this document.

As shown in the diagram above the HLD space is occupied by:

- This document
- A set of subsystem HLDs, describing the subsystems as set out in ARC/APP/ARC/0009.
- A set of Use-Case-Realisation Documents responding to the specific requirements stated in the use cases
- A set of topic HLDs which cover the design of specific topic areas.

Between them these documents cover all the HLD of the HNGX Counter.

The following Section show where other counter design is contained.

2.1 Subsystem HLDs

The architecture (See ARC/APP/ARC/0009) defines a number of sub-systems, as shown in the following diagram.

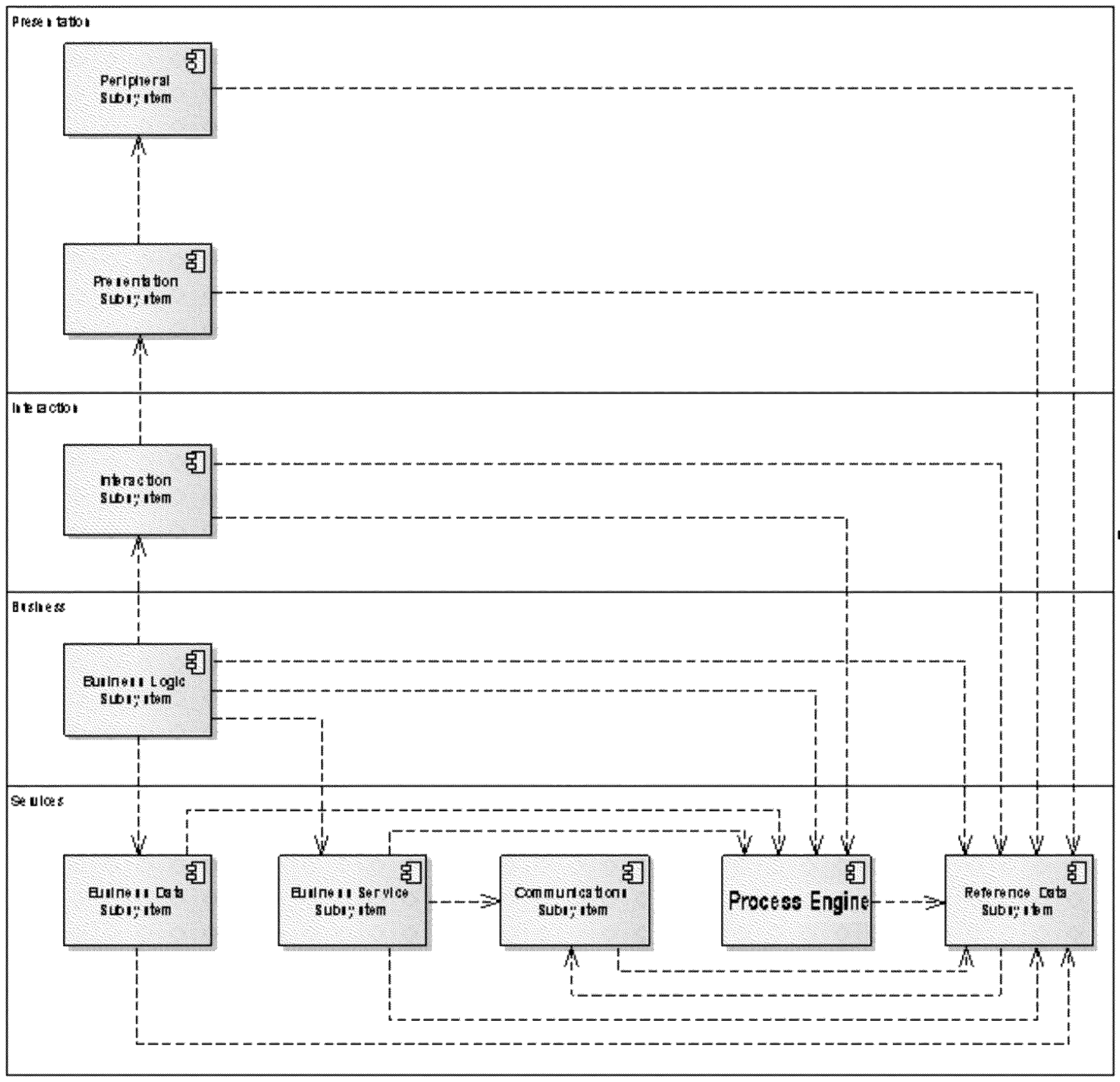


Figure 2-2 HNGX Counter Subsystems

The design of each of those subsystems can be found in the following documents.

Ref	Subsystem
DES/APP/HLD/0038	Peripheral
DES/APP/HLD/0039	Presentation
DES/APP/HLD/0040	Interaction
DES/APP/HLD/0041	Business Logic
DES/APP/HLD/0046	Business Data
DES/APP/HLD/0042	Business Services
DES/APP/HLD/0045	Reference Data

DES/APP/HLD/0076	Process Engine (PDL)
DES/APP/HLD/0043	Communications

2.2 UCR Docs

These are grouped into Barrels (matching use case barrels). Each document describes the design of the functionality to meet the specific requirements of a group of use cases.

Ref	Barrel	Use Case Abbreviation
DES/APP/HLD/0058	Banking	BNK
DES/APP/HLD/0059	Branch Accounting	BAC
DES/APP/HLD/0060	Branch Administration	BAD
DES/APP/HLD/0061	Branch Support	BSC
DES/APP/HLD/0062	Bureau de Change	BDC
DES/APP/HLD/0063	Cash & Stock Management	CSM
DES/APP/HLD/0064	E-Top-Up	ETU
DES/APP/HLD/0065	In and Out Payments	IOP
DES/APP/HLD/0066	Postal Services	PS
DES/APP/HLD/0067	Retail and Stock Sales	RSS
DES/APP/HLD/0068	Shared	GLB

2.3 Topic HLDS

Certain areas of design have warranted an HLD of their own. The following topic HLDs exist.

Ref	Title
DES/APP/HLD/0035	Exceptions and Logging Framework High Level Design
DES/APP/HLD/0069	HNG-X AP-ADC High Level Design
DES/APP/HLD/0083	HNGX Counter Subsystem : Recovery Management
DES/APP/HLD/0096	HNG-X Applications Training System High Level Design
DES/APP/HLD/0102	HNG-X Counter Help Design
DES/APP/HLD/0109	HNG-X Counter Reporting - High Level Design
DES/APP/HLD/0110	HNG-X Counter Applications: Transaction Benchmarking High Level Design
DES/APP/HLD/0112	HNG-X Counter Applications: Online Financial Applications High Level Design
DES/APP/HLD/0114	HNG-X Counter Applications: Online Services Definition Configuration and Usage High Level Design
DES/APP/HLD/0123	HNG-X HLD - Settlement Functions
DES/APP/HLD/0126	HNG-X Branch Accounting Counter High Level Design

DES/APP/HLD/0128	Postal Services Reference Data Maintenance and Counter Support High Level Design
DES/APP/HLD/0136	HNG-X Cash and Stock Management High Level Design
DES/APP/HLD/0137	HNG-X Counter Printing
DES/APP/HLD/0141	HNG-X User Interface Components HLD

2.4 Low Level Designs

Ref	Title
DEV/APP/LLD/0163	Stock Unit Balance Report Low Level Design
DEV/APP/LLD/0164	Branch Trading Statement Low Level Design
DEV/APP/LLD/0178	Migration Report Low Level Design
DEV/APP/LLD/0179	Counter Reports Low Level Design
DEV/APP/LLD/0180	HNG-X Reporting Accounting Low Level Design
DEV/APP/LLD/0182	HNG-X Recovery Low Level Design
DEV/APP/LLD/0184	HNG-X Help Viewer Low Level Design
DEV/APP/LLD/0185	HNGX-X Help Steps Low Level Design
DEV/APP/LLD/0187	AP-ADC Low Level Design
DEV/APP/LLD/0188	HNG-X Counter Applications: Banking and Online Financial Framework Low Level Design.
DEV/APP/LLD/0189	HNG-X Counter Applications: DCS Low Level Design
DEV/APP/LLD/0190	HNG-X Counter Applications: ETU Low Level Design
DEV/APP/LLD/0191	HNG-X Counter Applications: PINPad Low Level Design
DEV/APP/LLD/0192	HNG-X Tokens System Low Level Design

2.5 Other Reference Material

The following docs, while not part of the HLD of the counter, are relevant docs to which the counter conforms.

Ref	Title
DES/GEN/MAN/0002	APADC Reference Manual
DEV/GEN/MAN/0003	HNG-X UI Construct Catalogue
DES/APP/STD/0001	HNG-X UI Style Guide
DES/GEN/SPE/0003	Counter Type X Reference Data Definitions
DES/GEN/SPE/0007	HNG-X Menu Hierarchy and Messages
DES/GEN/SPE/0008	HNG-X Branch and Counter Reports
DES/GEN/SPE/0009	HNG-X Receipts, Slips and Labels
DES/GEN/SPE/0010	HNG-X Banking, Debit Card and ETopUp

	Receipts and Texts
DES/GEN/SPE/0011	HNG-X AP and ADC Receipts
DES/APP/IFS/0006	RAC Message Flows in HNGX
DES/APP/IFS/0018	XML Message Audit between Counter and BAL/OSR
DES/GEN/SPE/0003	Counter Type X Reference Data Definitions
DES/APP/MAN/0001	HNG-X: Type A / B Reference Data Counter Object Definitions
DES/APP/HLD/0057	HNG-X Counter Infrastructure Service and Process Control High Level Design

2.1.1 Associated BAL Designs

Ref	Title
DES/APP/HLD/0053	HNG-X BAL Reporting High Level Design
DES/APP/HLD/0094	HNG-X End-to-End Key Usage Overview High Level Design
DES/APP/HLD/0127	HNG-X Branch Accounting BAL High Level Design
DES/APP/IFS/0012	BAL Service Interface Specification
DES/SYM/HLD/0021	Branch Access Layer Management High Level Design

3 Requirements

3.1 Functional Requirements

The functional requirements places on the counter are contained in the use cases. See section 2.2 for the list of use case barrels.

3.2 Non Functional Requirements

This design responds to all the non-functional requirements placed on the counter (although some are covered by other HLD documents – see section 2 for list of HLD documents).

4 Migration

For HNG-X there is a new counter application which replaces the Horizon counter application. Nearly all migration on the counter is achieved by Systems Management removing the old counter application (and its environments) and installing the new HNG-X counter application (and its environments).

However, the HNGX counter application provides some specific features to enable the migrating user to get started on HNGX.

- The migrating user is forced to print the opening position report, and to confirm that there are no differences to the Horizon closing position report.
- The migrating user is asked to confirm they wish to migrate to HNG-X (the point of no return)
- The migrating user is reminded to set up passwords for other users
- The ability to reprint the opening position report

```
Note that upon migration only one, or sometimes two, users are provided with
a migrated password, and that is marked as expired so that they have to
change their password when they log in, and also need to go and provide
passwords for each of the other users; that is not special functionality for
migration so is not described here.
```

5 Design Attributes

5.1 Design Guidelines

The design approach adopted in this document is the strategy as described in the HNG-X Designer Guidelines (DES/GEN/MAN/0001).

5.2 Design Principles

The design principles adopted in this document are as described in the following documents :

HNG-X Counter Business Applications Architecture (ARC/APP/ARC/0009).

5.3 Modelling Guidance

The contents of each subsystem are presented using a number of class diagrams. Classes are used to represent capabilities of the counter application.

6 Design Overview

The target design of the counter application is shown in outline in the diagram below.

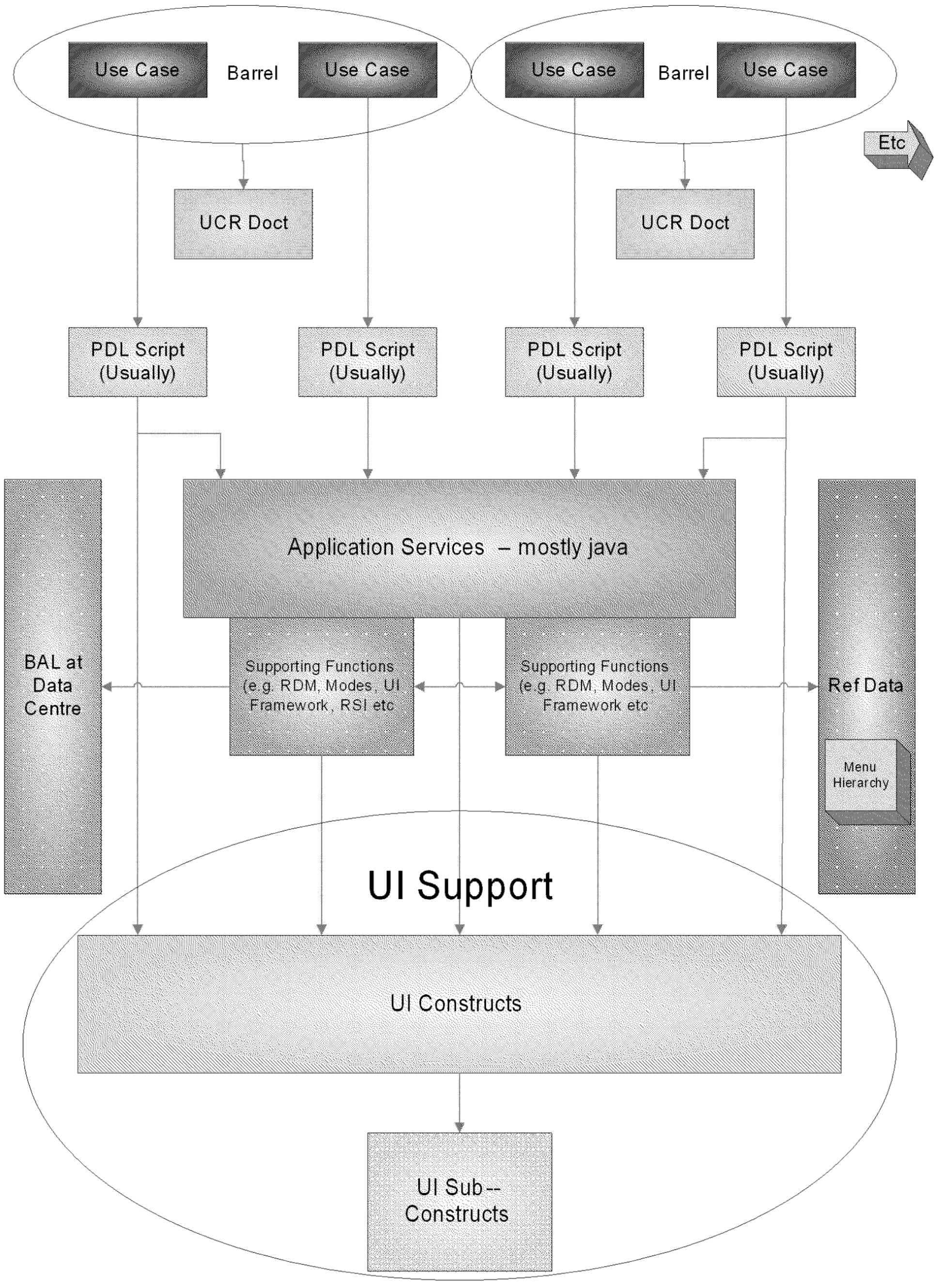


Figure 6-1 Counter Target Design

Note: the above is a TARGET design and there are places where (for good reason) the design does not conform exactly to this pattern.

Use cases are provided by the customer. There are approximately 100 use cases that are relevant the counter. They are grouped into barrels (approx 10). Section 2 lists the barrels.

Each barrel has an HLD document (known as a Use Case Realisation (UCR) document) that describes the high-level design of the business objects (and associated components) provided to support specifically the use cases in that barrel.

Most use cases give rise to a PDL script implementing the overall logic of that use case.

PDL scripts then make use of a central set of “Application Services” (see section 6.1) and underlying supporting functions to provide more generic functionality (for example the ref data manager, the UI framework and many more). These are mostly written in Java, although some are written in PDL.

These supporting functions include the ability to access the data centre (via remote business services) and the ability to access reference data.

```
Note: The reference data includes the definition of the menu hierarchy (see
document DES/GEN/SPE/0007 for details), but the menu hierarchy as such is not
a component of the counter HLD - it appears in reference data as a set of
menus, buttons, labels and navigation bars (navbars).
```

Supporting these functions and PDL scripts is a set of UI constructs and sub-constructs. These provide the basic building blocks of the user interface and are defined in the Construct Catalogue (DEV/GEN/MAN/0003).

6.1 Application Services

During the design of the HNGX counter, when trying to define the list of objects to develop we went through an exercise to go through all the use cases and identify sub-components that were required – in particular common components which were going to be of use across a number of use cases.

This exercise came out with a numbered list of sub-components which we named “Application Services”. The list included all APADC datatypes and all APADC functions. This list was used for planning purposes, to enable design to be focussed on specific areas, for carving up the work between teams and for ensuring we did not code duplicate functionality.

“Application Services” were just pieces of logic that were identified as being needed and were not meant to match specifically to any part of the architecture (i.e. they were not the same as Business Services, they could be implemented as a Business Service or a BLO or a BDO, or UIA, as a method on an existing object or whatever was appropriate to their functionality.) The appropriate implementation mechanism was down to LLD. In addition Application Services could be coded in Java or PDL – whichever was the most appropriate.

As the design and development proceeded then the original list of Application Services became less relevant and a decision was made not to maintain it as more (lower-level) supporting functions/sub-components were defined, but not identified as Application Services in themselves.

In addition it was identified that some of the Application Services defined were in fact UI components supporting UI constructs so they were re-classified as Scripted Objects (see DES/APP/HLD/0141 for a definition of Scripted Objects) and some Use-Case logic was (incorrectly) classified as Application Services.

Thus the concept of Application Services was dropped as the boundary between them and the supporting functions became blurred. In fact there was no definitive list of what they were. However, in some areas of the documentation there is still reference to Application Services (e.g. in Settlement – DES/APP/HLD/0123, and also in the wiki) and to their numbers (e.g. AS01 for Start Transaction). Thus Application Services are included in the diagram above to show how they fitted into the picture, but they are shown in tandem with the “Supporting Functions” that extended them.

Application Services are no longer a feature of the design and are not defined. The design does not attempt to include a list of Application Services, and references to them only exist for historical reasons.

FUJ00091790

7 Counter Component Design

This section covers a number of areas of counter design that are not covered in Topic HLDs or UCR documents (see section 2).

7.1 Audit

7.1.1 Purpose

This section describes the high level design of the audit mechanism in the counter application.

7.1.2 Introduction

In the counter there are Events (which are always Auditable) and Auditable Messages. Events are always delivered in Auditable Messages, but not all events force an Auditable Message of their own.

7.1.2.1 Events

There are numerous actions that can be undertaken by the clerk that trigger an audit event record to be generated.

In some cases, this auditable record needs to be communicated to the data centre immediately, but in other cases it is permissible to defer the audit records until the next auditable event that requires immediate action is undertaken, at which point, all of the deferred audit records will also be communicated in an Auditable Message.

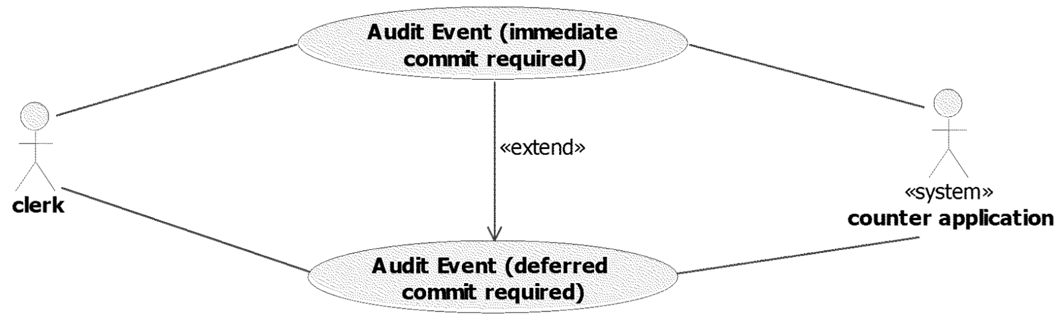

Audit Event (immediate commit required)
«extend»
clerk
«system»
counter application
Audit Event (deferred commit required)

Figure 7-1 – Audit Event Use Case Diagram

7.1.2.2 Auditable Messages

Events are delivered in Auditable Messages. Each Auditable Message has a unique JSN.

If the counter application fails to successfully communicate the audit records to the data centre, then the user will be given the option to retry the option a finite number of times before the user is forcibly logged off.

7.1.3 Classes

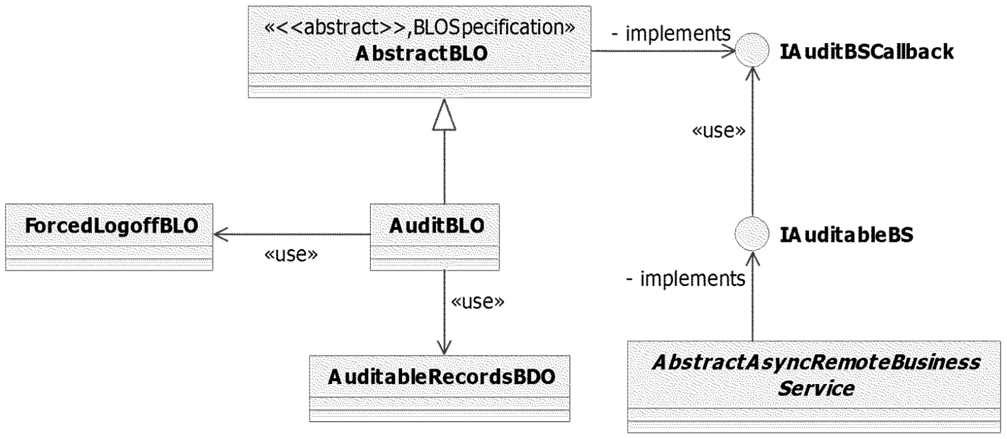


Figure 7-2 – Audit Overview Class Diagram

Class : AuditBLO

The AuditBLO is s single-instance BLO that is created by the ApplicationBLO at counter start up. It maintains the audit data using the AuditableRecordsBDO, and supports adding of audit event data – both deferred and immediate from any other object that needs to provide audit data.

Via the IAuditBSCallback interface it provides a common generic mechanism for auditable business services to ask the user if he/she wants to retry and to apply a forced logoff if not.

Class : AuditableRecordsBDO

Maintains audit data for use by the AuditBLO.

It maintains a list of deferred audit event records, and one immediate audit event record.

It manages the allocation of appropriate JSNs.

Class : IAuditableBS

Registers callback (to AuditBLO)

Class : IAuditBSCallback

Provides callback from Auditable Business Services which need to ask the user if he wants to retry or not. Also provides for forced logoff.

Class : AbstractAsyncRemoteBusinessService

Performs the remote business service operation, providing a (configurable) no of retries and via the IAuditBSCallback (linked up by the Business Services Manager to the AuditBLO) communicates with the AuditBLO to get the user asked if he wants to retry or not and to do the forced logoff if required.

7.2 Data Centre Access from Counter

The Communications Subsystem provides the route to access the BAL at the data centre. This provides for read access and write access. (See DES/APP/HLD/0043 for details).

See section 7.1 for details of auditable messages.

BAL Service Interface Specification (DES/APP/IFS/0012) provides a definition of the interface to the BAL and the structure of messages passed.

XML Message Audit between Counter and BAL/OSR (DES/APP/IFS/0018) provides detail of the message that are passed.

7.3 BusyWait Message

7.3.1 Introduction

This section covers the display of an appropriate message whilst the application is busy. This message informs the user that they must wait and that no UI interaction is currently possible. Full details are in document DES/APP/HLD/0141

7.3.2 System Requirements

The System requirements are as follows:

- Dialog Message to be displayed which matches the MSG-Busy construct as defined in the Construct Catalogue.
- Disable user interaction until operation complete.
- Once the operation is complete, the dialog is removed and UI Interactions restored.

7.3.3 Design Overview

This component is used when a potentially long running operation is being performed, such as a request/response interaction with the BAL. In this situation it needs to put up a dialog informing the user that they must wait for the operation to complete. It must also disable all user interaction until that time. Once complete it removes the dialog and restores UI interactions. There is no requirement for the user to be able to cancel the long running operation. There is no time-out mechanism inherent to the busy wait mechanism – however, long running processes such as BAL interactions do themselves have time-outs, which would cause the tear-down of the busy wait message.

7.3.4 Design Strategy

To meet these requirements the primary component to be developed is the *BusyWaitBLO*. Where any potentially long running operation is identified this BLO will be used to provide the behaviour described above. Specifically, activate should be called on the *BusyWaitBLO* before any potentially long running operation and deactivate after it. The *BusyWaitBLO* works in conjunction with the *BusyWaitUIA (which blocks UI input in the intervening time before the busy wait message appears)*.

There is one timeout read from configuration by the *BusyWaitBLO* –for the time before the busy wait dialog is displayed (typically this is about 2 seconds).

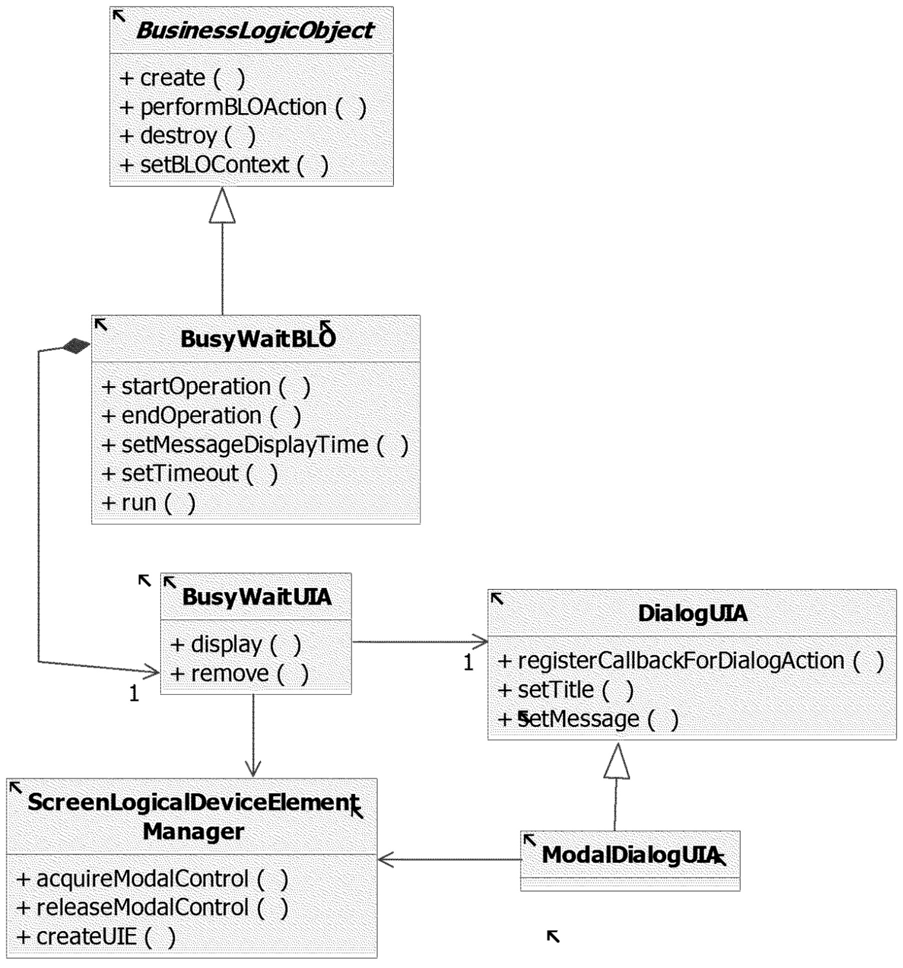


Figure 7-3 – Busy Wait and Dialog classes

7.3.5 Classes

7.3.1.1 BusyWaitBLO

activate()

- starts a new timer which checks after 2 seconds if the operation is still ongoing – if so, a dialog is then displayed.

deactivate()

- Removes the BusyWait Message at the end of the busy wait operation.

7.3.1.2 Usage of the BusyWaitBLO

The most common usage of BusyWaitBLO is via on-line service calls to the BAL – the basic framework automatically handles activation and deactivation of BusyWaitBLO (and subsequent busy wait message) via AbstractAsyncRemoteBusinessService and its related subclasses.

7.4 Configuration

7.4.1 Introduction

This section defines a standard mechanism to allow access to configuration items (items that are not present in reference data). It details the configuration items that can appear in the following two major components within the system

1. The Counter
2. The BAL / OSR (Online Service Router)

7.4.2 System Requirements

The system requirements are as follows:

1. Define a standard mechanism to allow access to configuration items.

7.4.3 Design Strategy

This section details the configuration items that can appear in the following two major components within the system

1. The Counter
2. The BAL / OSR (Online Service Router)

For each of these components there is a standard configuration file - which is located in a well known directory (see below). Items contained in these configuration files are ones that support would reasonably be expected to change/modify.

To determine if an item should be in this file then the result of a modification of the item should be considered. For example would a modification result in the changed behaviour of the application but not its functionality? If so it is placed in this file.

Examples:-

1. A “timeout” value should be contained within this file as it does not alter functionality.
2. IoC configuration files define application functionality so should not be placed in this file.
3. A connection pool size does not affect functionality so should be in the file.
4. XMLParser configuration file also defines functionality so belongs in its own configuration file.

Items (2) and (4) are there to aid flexibility in the development phase and not during live operation so support would not be expected to modify these files without help from development.

Each configuration item should contain the following information:

- The configuration key - This is the unique reference in the configuration file that represents this item - it often has a “dotted” qualifier on the key that details which component the item belongs to. This is the key by which the code can retrieve the data.
- A value that should be associated with the item
- A description of the item in question (as a comment)

The following is an example of a configuration item:

```
# length of inactivity before we display the relogon screen
templock.ui.inactivity.milliseconds=900000
```

7.4.4 The Configuration system

The configuration system is comprised of a simple singleton class that belongs in the shared code repository (note it is shared between the BAL/OSR and Counter). It will access the property file relevant to the component it is contained within. For example the file could contain the following properties:-

```
config.item.1=This is the configuration item 1
config.item.2=3
```

The first item is a string that can be read from the properties file.

The second is a simple number that can be read from the properties file.

Additionally values of the properties can be overridden by command line options:-

For example specifying -Dconfig.item.2=abc on the command line would cause the configuration system to ignore the definition in the property file and use the value “abc” from the command line instead. (This facility is only be used during development/testing – not in the delivered system)

Further to this, properties can be overridden via a further “override” properties file, one for the Counter, one for the BAL/OSR.(See below)

7.4.5 Configuration System API

The API to the configuration system is as follows:-

Singleton “Configuration” class common to all parts of the system (BAL/OSR and Counter) with methods:

- **String getString(String key, String defaultValue)** Given a key reference to the property item return the property item as a String. Use the default value if not found in the system, log a warning if the default item is used.

- **Integer getInteger(String key, Integer defaultValue)** Given a key reference to the property item return the property item as a Integer. Use the default value if not found in the system, log a warning if the default item is used.

- **int getInteger(String key, int defaultValue)** Given a key reference to the property item return the property item as a int. Use the default value if not found in the system, log a warning if the default item is used.

- **Double getDouble(String key, Double defaultValue)** Given a key reference to the property item return the property item as a Double. Use the default value if not found in the system, log a warning if the default item is used.

- **double getDouble(String key, double defaultValue)** Given a key reference to the property item return the property item as a double. Use the default value if not found in the system, log a warning if the default item is used.

- **boolean getBoolean(String key, boolean defaultValue)** Given a key reference to the property item return the property item as a boolean. Use the default value if not found in the system, log a warning if the default item is used.

7.4.6 Counter Configuration Items

The counter's default property file is located within the "classes" folder under the main start-up directory that contains the Java startup batch file – this is on the "classpath". Its name must be "application.properties".

It is possible to specify an override properties file by means of a java command line option -Dcom.fujitsu.poa.configfile=<full pathname to an override configuration file>.

The override file is combined with the standard application.properties at runtime. Where there is a clash of keys, the value specified in the override file will win.

Note that on the Counter, the override properties file is normally specified by the startup batch file as %HNGX_EM_CONFIG%, a Windows Environment variable, which can indicate any plain text properties file anywhere on the Counter machine.

See section 7.9.2.2 for how this override file is used.

7.5 Counter Threading Policy

7.5.1 Introduction

Although the counter application is essentially a single-user application, not all its processing can take place on one thread. There are various reasons why other threads are required (see below).

This section covers the reasons for having multiple counter threads and how they are controlled and managed. Its purpose is to describe the threading policy of the Counter Application.

It identifies various tasks that need to run in different threads. But the detail design of those tasks is outside the scope of this section.

7.5.2 Requirements

The threading policy of Counter application is driven by various requirements of the application. The requirements are mostly of architectural and non functional nature e.g. performance, reliability and security etc.

Here is a list of such requirements that influences the threading policy.

- Polling - Checking at a regular frequency if there is any change in the reference data. This needs to take place in the background and be invisible to the user.
- Reference data processing. Once polling has identified that new reference data is available the new reference data needs to be downloaded and processed ready for putting into use. This needs to take place in the background while the user is able to operate with the existing reference data.
- Reference data re-indexing. Even if new reference data is not downloaded the ref data needs to be re-indexed at the end of the day to pick up a new working day set for the following day.
- Polling - Checking at a regular frequency if there are any new unread memo messages. This needs to take place in the background but the user made aware that there are new messages.
- NIC – This will be running in Counter application as an interface to Data Centre, and needs not to tie up the counter so that timeouts and system busy message can be accommodated.
- Clock - The clock in a specified zone of Counter main screen needs to be updated continuously.
- System Inactivity - If the system is inactive for longer than the permissible time period, a screen saver should be displayed. (aka Templock)
- System Busy Wait - If an operation is time consuming e.g. interaction with Data Centre takes longer time, user should be presented with user interface showing the system is busy.
- Interactions with Peripherals – some peripherals will provide responses at unpredictable times which must be read when available.
- Handling of uncaught exceptions - When an uncaught exception or error occurs in the system, the system should handle this gracefully.
- The logging should be reconfigurable dynamically.
- The Counter needs to listen to and process shutdown requests at any point in time externally via a socket.
- Printing – in some cases it is necessary to allow the user to be able to cancel the printing of a large amount of data.
- UI – the interactions with the user utilise Swing and that provides an EDT thread that is used for UI and must not be blocked in various circumstances.

7.5.3 Policy

Based on above requirements, the following threading policies are in place.

7.5.4 Threads

The following threads exist in the system. These threads may run continuously or be created on an ad-hoc basis and finish.

- **Initial Thread**
    - This thread is started when the counter starts up – ie, within the main[] method of the application.
    - This thread simply performs log4j set up and other very basic system set up, before passing control to the Start-up Thread, after which the thread completes. (see section 7.11)
- **Start-up Thread**
    - This thread does little other than start the ApplicationManager initialisation on the EDT (see below).
    - Once this is achieved, the thread completes. (see section 7.11)
- **Event Dispatching Thread (EDT)**
    - Swing/AWT runs this thread internally. There is no need to create it explicitly.
    - This thread polls the EventQueue and takes event objects from queue one by one and sends them to interested parties.
    - Most of the Counter business logic runs on this thread – in fact, most of the Counter code runs on this thread. This has the advantage of avoiding the need to synchronize most of the code and components since they can all mostly assume that they are only ever called on this one thread. The disadvantage to this is that the "UI" nature of this thread leaks to other non-UI areas of the Counter. This is considered to be a lesser of two evils. This does mean, for example, that care is taken to avoid executing any long-running business logic on this thread, since this will make the Counter UI unresponsive.
- **Polling Thread**
    - This thread is used to repeatedly poll the data centre in relation to reference data status updates, and (when a user is logged in) unread memo messages. Parts of PollingBLO run on this thread. PollingBLO starts this thread.
- **Reference Data Queue Processing Thread**
    - This thread polls the queue containing the list of reference data deliverables to be downloaded.
    - Once it finds a deliverable on the queue, it fires off a new "FileDownloadHandler" thread to actually perform the download and delivery.
- **Next Day Working Day Set**
    - This is a simple timer which starts at 11:45 and regenerates the indices as if it was tomorrow, ready for index replacement at midnight.
- **NIC Threads**
    - There is a pool of 5 "worker" threads running in the NIC which handle request/responses to/from the BAL.

   - These threads run continuously and will last the lifetime of the application. They may of course by blocked most of the time waiting for a job (request) to process.
- **Peripheral Threads**
   - Various peripherals may create their own threads.
   - A peripheral may have a reader thread or a writer thread (or both) depending on the nature of the POU.
   - A reader thread will be created once when the POU is created, by the POU itself. This thread will last for the Counter lifetime and will read continuously from the COM port.
   - For a logical write operation, a new writer thread will be created. The writer thread will complete once it is finished its write task.
- **Clock Threads**
   - This is started as part of the InfoBarUIE, from the EDT. There is one Clock thread for each instance of InfoBarUIE (for which there can be many, and always at least 1).
   - This thread is responsible for updating the clock on the screen.
- **UI Inactivity Threads**
   - Started from the EDT as part of the InactivityTrackerBLO – there are various timer threads used here. They are started and stopped repeatedly.
   - This thread is responsible for checking to see if the system has been used in the last 15 minutes.
- **Remote Shutdown Thread**
   - This thread is started from EDT as part of the Remote Shutdown component. This thread handles both dynamic logging changes and remote shutdown commands.

7.5.5 Thread Group

Generally all threads are part of an HNGX-specific ThreadGroup – this allows us to ensure that any premature or unexpected thread deaths (due to exceptions) can be identified, investigated and corrected if appropriate.

All thread groups extend ApplicationThreadGroup.

Thread groups provide a mechanism for collecting multiple threads into a single object.

The uncaughtException() method of Java class ThreadGroup is overridden for graceful handling of Uncaught Exceptions and Errors.

It is envisioned that uncaught exceptions and errors occurring in any of the above mentioned threads will require similar handling. Care is taken to ensure that each thread created is part of an appropriate thread group.

7.5.6 Thread Priority

- Most threads run at a default thread priority

- Background threads (such as Polling thread, FileDownloader and Next Working Day Set) will have a lower priority to ensure that the Counter UI performance is not impacted.

7.5.1.1 Inter-Thread Communication

The call from a BS to RSI will appear to be to be synchronous. But RSI's communication with NIC is asynchronous. RSI puts a request in the Queue for NIC. NIC gives a callback to RSI.

7.6 Identifier Generator

7.6.1 Introduction

This section covers Identifier Generation of Sequence Numbers for Communications with the Branch Access Layer.

There are various places in the HNG-X system where unique identifiers are required. The identifiers are used for auditing, in reports, recovery, printing on receipts, in APADC and relating separate parts of a transaction.

The counter needs to maintain and use some of these unique identifiers when communicating with the Branch Access Layer (BAL). This design identifies a single component that is used to maintain them and increment them where appropriate.

For auditing purposes, one of the unique identifiers (the JSN) is to have no gaps in the sequence and it should therefore be possible to check whether data is missing. Other unique identifiers (e.g. Session and Transaction) are required in various places, without a requirement for them to be contiguous.

7.6.2 Design Strategy

7.6.2.1 Overview

The design strategy of the IdentifierGeneratorLDO is described below:

The SystemBDO is used to provide access to the IdentifierGeneratorLDO

The SystemBDO is a singleton BDO.

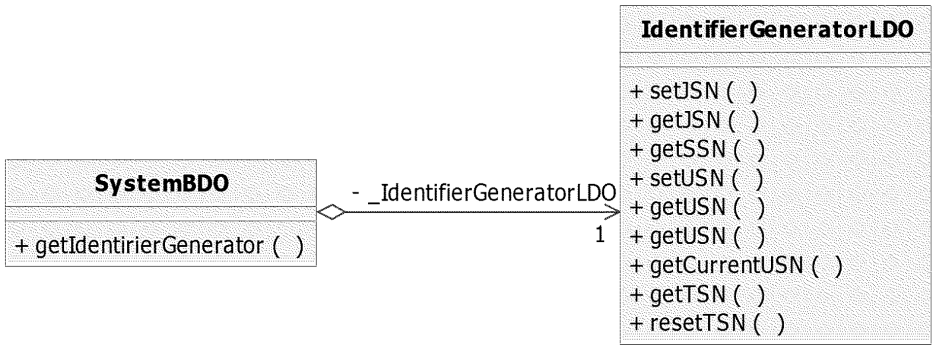


Figure 7-4 – IdentifierGenerator/SystemBDO Diagram

7.6.2.2 Unique identifiers of concern to the counter

JSN – Supplied to counter at logon. Incremented and used for every auditable message.

SSN – Unique for each customer session. (derived from the JSN)

USN – updated for every usage (e.g. for each piece of recovery data)

TSN – updated for each transaction in the basket – but reset to 1 after each basket.

7.6.2.3 Classes

IdentifierGeneratorBDO

setJSN() – sets the initial value for a JSN (this is supplied at logon).

getJSN() – Returns the JSN and then increments the stored value by 1.

getSSN() – returns JSN mod 1,000,000. (not incrementing JSN)

setUSN() – sets the initial value of the USN (this is supplied at logon)

getUSN() – returns the USN and then increments the stored value by 1.

getCurrentUSN() – returns the last value returned from GetUSN (without incrementing the USN stored value).

resetTSN() – resets the value for TSN to 1.

getTSN() – returns the TSN and then increments the stored value by 1.

7.6.2.4 Background information on the use of unique identifiers

Each branch is assigned a unique identifier, and each logical Counter within the branch is assigned a unique position – if a Counter is physically replaced its logical position does not change. These two numbers are sufficient to identify the source of any message sent to the Data Centre.

When a request to the Branch Access Layer must be audited, such as when processing settlement data or recording a significant Counter event, each audited message for a particular counter must have a unique identifier, and when the messages are journalised those identifiers must form a dense set to show that the journal is complete. The Branch Database holds a sequence number, known as the Journal Sequence Number (JSN) for each counter in the estate. The Counter is supplied a JSN at logon which it then maintains. The JSN is incremented each time the Counter sends an auditable message and receives confirmation that the audit record has been written. Without such confirmation, a Counter has no choice but to eventually force a log-off. On Counter, log-on the Data Centre tells the Counter what its next JSN should be. In this way, the Counter and Data Centre are always in step.

If the user session is terminated tidily, the counter supplies the latest JSN, which is held in the Branch Database. However, if the session is terminated abnormally, the Branch database will not have the latest JSN used by the Counter. However, at next logon the Branch Database can determine the highest JSN used from its tables and can therefore provide the Counter with the next JSN for it to use.

The Transaction Processing System (TPS) requires that every customer session (i.e. basket) is given a session sequence number (SSN) and every transaction within a session has a transaction sequence number (TSN). The SSN has a cycle length of 1,000,000. The Counter's JSN provides a unique value to use, so the SSN for the session is simply (JSN mod 1000000) and the TSN for each transaction is defined by the ordering of the transactions within the session. Both SSN and TSN must be supplied by the Counter for individual transactions in the settlement message.

On-line transactions need a Transaction Recovery Identifier. To support this, the Branch Database holds a sequence number, known as the Uniqueness Sequence Number, or USN, for each Counter in the estate. The USN to use at a Counter is supplied at logon, and the Counter tells the Data Centre the USN it has allocated for each piece of recovery data. The USN is also used elsewhere where a unique no is required (e.g. postal services). .

If a session is terminated abnormally, the Data Centre will not have the latest USN used by the Counter, but at the next logon it can determine the highest USN that has been used and can therefore provide the Counter with the next USN for it to use. Note that the Counter may have allocated a higher USN than this during the abandoned session, for some purpose other than recovery, but only USNs that have actually been written to the database need to be unique at the start of the next session.

The Authorisation Agents require a correlation identifier to tie together the Request, Authorisation and Confirmation messages that make up an online transaction. The Transaction Recovery Identifier could have been used for this purpose but the Transaction Processing System expects this identifier to be expressed as a string with a particular syntax, thus:

pp-bbbbbb-cc-nnnnnnnnnn-uu

where

pp	is a fixed prefix; 00 for HNG-X, 44 for Horizon
bbbbbb	is the Branch Identifier
cc	is the Counter Identifier
nnnnnnnnnn	is the USN
uu	is a suffix, which for HNG-X is fixed at 1

Leading zeroes are suppressed, giving identifiers such as 00-49934-2-1. This form of identifier is known as an HTxnNum, and is common between Horizon and HNG-X, but the derivation of the fields is different between the two systems.

Recovery records for NBS, ETS and DCS transactions include the HTxnNum. When a Counter wishes to retrieve status information about a recoverable transaction, it is the HTxnNum that is used as the transaction identifier.

7.6.3 UML High Level Design Module

cvsProject	
Model	//CounterHighLevelDesign/CounterHighLevelDesign.mdx//ApplicationComponents/ BusinessDataComponents/DesignElements/System/IdentiferGenerator//

7.7 Menu Design

7.7.1 Introduction

The purpose of this section is to cover the design of the handling of Menus.

7.7.2 Design Overview

The operation of the menu hierarchy is described in *HNG-X Menu Hierarchy and Messages* (DES/GEN/SPE/0007), so is not repeated in full here. In summary:

- The menu hierarchy consists of hierarchies of navbars with associated menu screens
- The menu hierarchy is described within a set of excel spreadsheets, maintained by POL/CS.
- The menu hierarchy is generated from the spreadsheets into xml reference data
- In reference data the menu hierarchy is realised as a set of Menus, Navbars, Labels and Buttons. (Note that navbar tabs are realised as Buttons.) More detail can be found in *HNG-X UI Construct Catalogue* (DEV/GEN/MAN/0003)
- The counter needs two menu hierarchies – a front-office menu hierarchy and a back-office menu hierarchy, and is told via branch reference data which navbars form the topmost points of each one.
- A third menu hierarchy (the Engineers MH) is used outside of login.
- In the menu hierarchy a menu provides a link to modes and token sets.
- In the menu hierarchy a button has an associated Action. Actions are ref data elements (see *HNG-X UI Construct Catalogue* (DEV/GEN/MAN/0003) for details), that provide a link to permissions, constraints and to code. Actions are processed by the action controller mechanism.
- There is a special case for New Reversals where (to save having to hold 2 definitions of the front office hierarchy) the front office menu hierarchy can be displayed from a button on a back office menu in New Reversal mode. (Note that the reversals menu hierarchy is technically delivered in reference data as another “root” navbar, which then re-uses parts of the front office menu hierarchy)
- A menu hierarchy has a special menu for settlement. This is not linked to the other menus in that menu hierarchy, but is displayed by the code when needed.

7.7.3 Design

MenuBLO provides the functionality for menu control. It has the following methods:

- *displayMenu(ActionData or MenuUIA)* – Creates and activates a MenuUIA that displays a specific screen (Menu parameter in Action data, or the MenuUIA parameter determines the specific instance/spec to use). It uses the WorkspaceScreenControllerBLO.
- displayHomeMenu – used for the DisplayHomeMenu action
- changeMode(Menu UIA) – sets the mode and token set as defined on the menu
- clearBasketIfInvisibleAndNonEmpty() – used to flush the basket in the occasional circumstance that an application is abandoned when the basket is not visible.

There is a GoHomeBLO (implementing AS420) that provides the ability to return to the home page in whatever hierarchy we are currently in (Front office or back office). It uses navBarBLO to do this.

There is a NavbarBLO – which manages the displaying of navbars. It has methods:

- displayBackOfficeNavBar – it finds the Back Office top navbar and uses displayNavbar to show it.
- displayFrontOfficeNavBar - it finds the Front Office top navbar and uses displayNavbar to show it.
- displayNavBar (navbar) – displays the specified navbar.
- displaySettlementNavbar – shows the settlement menu
- displayNewReversalNavbar – controls the special case for New Reversals where the front office menu hierarchy is displayed from the Back-office menu hierarchy in new reversal mode.
- getRootNavBarName – finds current root navbar name
- isHomeNavBar – returns true/false
- isLocalHomeNavBar – return true/false
- getOffice – returns Front, Back or Engineer.

There is a selectFunctionOptionsBLO – for managing token swipes.

More details on these functions can be found in *Interaction Subsystem High Level Design* (DES/APP/HLD/0040) and *HNG-X User Interface Components High Level Design* (DES/APP/HLD/0141).

7.8 Singleton Objects Design

7.8.1 Introduction

This section describes the high level design of the various subsystem Objects with regards to singleton behaviour to be realised by the counter application.

It applies to many business objects in the system. I.e. all:

- Business Data Objects
- Business Logic Objects
- Business Services
- Peripheral Objects
- Remote Services
- UIAs
- UIEs

The architecture does not specify whether or not *Objects* should be uniformly managed as singletons. However, there is a definite need to implement some *Objects* as singletons, and the purpose of this section is to explain a mechanism that will realise this requirement.

7.8.2 Design Strategy

The singleton objects are realised by two objects:

- ISoleInstance
- AbstractManager

7.8.2.1 IsoleInstance

Any of the objects listed in section 7.8.1 above can declare themselves a singleton object simply by inheriting from ISoleInstance.

7.8.2.2 AbstractManager

AbstractManager provides the logic for managing sole instances and is extended by the following subsystem managers.

- BusinessDataManager (see *Business Data Subsystem High Level Design* (DES/APP/HLD/0046))
- BusinessLogicManager (see *Business Logic Subsystem High Level Design* (DES/APP/HLD/0041))
- BusinessServicesManager (see *Business Services Subsystem High Level Design* (DES/APP/HLD/0042))
- PeripheralManager (see *Peripherals Subsystem High Level Design* (DES/APP/HLD/0038))
- RemoteServiceInterfaceManager (see *Communications Subsystem High Level Design* (DES/APP/HLD/0043))
- UIAManager (see *Interaction Subsystem High Level Design* (DES/APP/HLD/0040))
- UIEManager (see DES/APP/HLD/0039)
- UIWManager (See DES/APP/HLD/0039)

The AbstractManager object maintains (for each of the managers listed above) a list of SoleInstance objects. When a request to build an object is made (using the BuildImpl method) the AbstractManager checks if the object implements ISoleInstance. If it does not it creates a new instance of the object. If it does then it checks to see if it has that object in its list of SoleInstance objects. If it does it returns a reference to that object, otherwise it builds a new instance of that object and adds it to the SoleInstance list.

7.8.2.3 Destruction

Sole Instance objects are only destroyed if the counter closes down (and the appropriate manager relinquishes them)

It is incumbent on the objects that might be sole instances not to destroy themselves in the meantime. This is accomplished by the following objects implementing a Destroy method that will check if it is a sole instance and not destroy itself:

- AbstractBDO
- AbstractBLO
- AbstractUIE

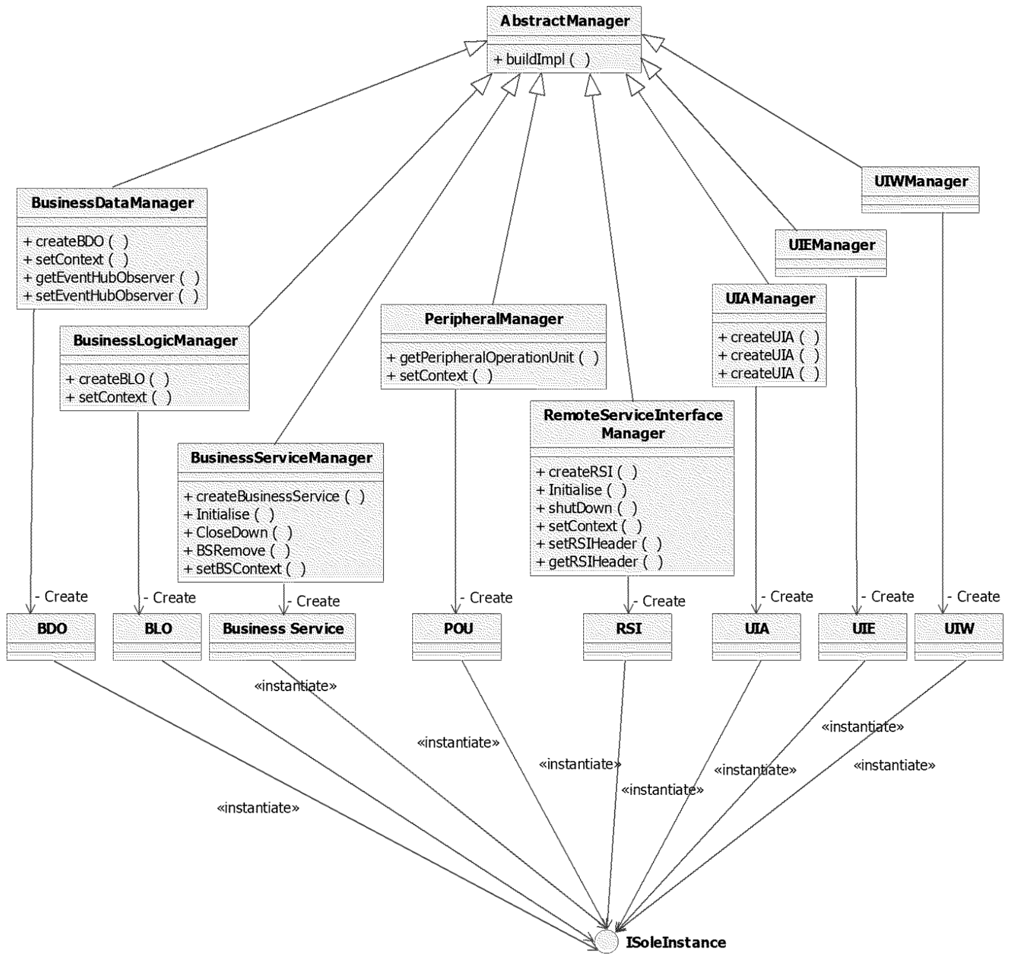


Figure 7-5 –Singleton Mechanism Overview

7.8.3 SystemBDO

A good example of a singleton object is the SystemBDO, which contains references to frequently required objects. It provides a central point for all BDOs and BLOs to get access to commonly used data. The diagram below shows how the SystemBDO is structured.

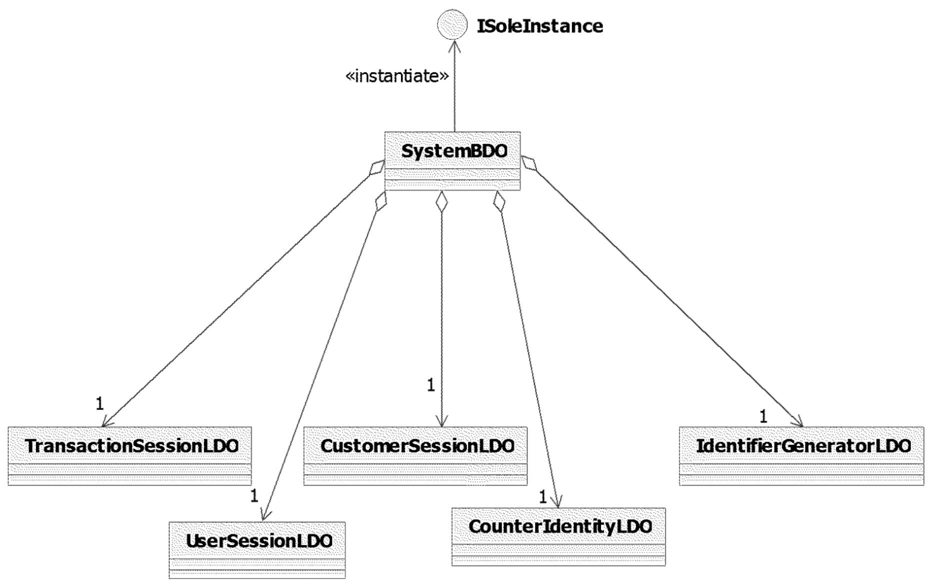


Figure 7-6 – SystemBDO

7.9 Counter Identity

7.9.1 Introduction

The section describes how the Counter finds its identity – i.e. its Branch Code (formerly called FAD code) which 6-characters in length and Counter node Id.(01-99). Note that the 7-digit FAD code is delivered in Branch Reference Data.

The Counter has an identity specified to Estate/System management by the engineer at installation time. This is maintained by EM/SM and the counter application needs to get access to its counter identity as part of its initialisation, so that it can identify itself to the data centre.

7.9.2 Design Overview

7.9.1.1 Design Strategy

The counter identity is obtained from an Application Properties override file that is set up by Systems Management,

The counter identity is set up under the SystemBDO at system start-up.

7.9.1.2 Application Properties override file

This contains the counter Id in the format:

NodeId=x

BranchCode=y

Where x is an integer from 1-99 and y is an integer from 1 to 999999 (i.e. it does not need leading zeros for numbers less than100,000)

7.9.1.3 Application Properties file

This has entries to convert the BranchCode and NodeId as follows:

com.fujitsu.poa.ctrc.businesslogic.counterid=${NodeId}

com.fujitsu.poa.ctrc.businesslogic.branchid=${BranchCode}

7.9.1.4 CounterIDLDO

This is an LDO that is linked into the SystemBDO (in the same way that the “Sessions” LDOs are). This is defined in the “singletonBDOs” design. It just has getters and setters for the BranchID and CounterID.

7.9.1.5 CounterIDBLO

A CounterIDBLO is defined whose job it is to:

- Get the BranchID (from config data as set up by SM - `"com.fujitsu.poa.ctrc.businesslogic.branchid"`)
- Get the Node ID (confusingly called the CounterID) (from config data as set up by SM - `"com.fujitsu.poa.ctrc.businesslogic.counterid"`)
- Create the CounterIDLDO and link it into the SystemBDO
- Store the BranchID and CounterID in the CounterIDLDO

This BLO is invoked from system start-up by the Application BLO (before starting the Polling BLO).

7.10 Inactivity Timer and Temporary Lock Mechanism

7.10.1 Introduction

The purpose of the Inactivity Timer and Temporary Lock mechanism is to ensure that should a Counter be left unattended for a period of time, then it is locked and cannot be used again without the user being re-authenticated (or logged out). Should the counter be left unattended for a further period of time, then the user session will be ended by forcing log-off. The Counter can also be locked by the user pressing a "Lock" button on the Counter. (See section 7.14.3.2)

Temporary Lock, including log-on after temporary lock and log-off by a new user after temporary lock, is documented in *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060) as part of BAD-11: Log On to system. System enforced log-off, which is how the user session is ended after a configured period of inactivity, is documented in DES/APP/HLD/0060 as part of the Do Forced Logoff component. This section contains a description of the Inactivity Timer, which tracks the amount of time the Counter has been inactive and invokes Temporary Lock or Do Forced Logoff if required.

7.10.2 Requirements

The requirements for the Inactivity Timer and Temporary Lock mechanism can be found in the non-functional requirements (see 7.10.2.1 Non-Functional Requirements below) and in parts of three use cases. The main driver for these requirements is security, to ensure that unattended terminals are not used by unauthorised people to carry out transactions which would be recorded against whoever was logged on.

BAD-11: Log On to system - documented in *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060)

BAD-3371: System enforced Temporary lock

BAD-3384: Clerk selected Temporary lock

BAD-3162: Log on after temporary lock

BAD-3217: Log off by a new user after temporary lock- confirmed (No open 'customer session')

BAD-3372: Log off by a new user after temporary lock- confirmed (Open 'customer session')

BAD-39: Log Off from System - documented in *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060)

BAD-3359: System enforced Log off (no open Transaction session) - documented as part of the Do Forced Logoff component

GLB-270: Transact Customer Basket

GLB-2488: Auto Settlement with successful Data Centre connection - documented in *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060) as part of the Do Forced Logoff component

GLB-2501: Auto Settlement with unsuccessful Data Centre connection - documented in DES/APP/HLD/0060 as part of the Do Forced Logoff component

7.10.2.1 Non-Functional Requirements

DOORS ID	***System Requirement ID (if applicable)***	***SRS Section Heading***

SEC-3274	T-CA-0031	Security
SEC-3274	T-CA-0032	Security
SEC-3274	T-CA-0033	Security
BAD-39	T-CA-0034	Security
SEC-3296	T-CA-0034	Security
BAD-39	T-CA-0035	Security
SEC-3296	T-CA-0035	Security
BAD-39	T-CA-0036	Security
SEC-3296	T-CA-0036	Security

7.10.3 Design

The Inactivity Timer comprises two components, InactivityTrackerBLO which keeps track of when the last activity was and takes appropriate action after configured periods of inactivity, and InactivityControllerBLO which provides methods to lock the screen or log the user off.

Inactivity means:

No keyboard usage

No touch screen (mouse) activity

7.10.3.1 InactivityTrackerBLO

InactivityTrackerBLO is initialised on system start-up. It runs on the EDT thread (see section 7.5 so any thread-related activities (e.g. invoking a timer, invoking a Runnable) must be carried out through the CounterUtilities class which has been provided for all Swing and AWT references outside the UIW layer.

The role of InactivityTrackerBLO is:

Wait for a user to log onto the Counter

After a configured period of inactivity, lock the Counter

After a further configured period of inactivity, log the user off the Counter

and it operates as follows:

InactivityTrackerBLO initialisation:

Gets the configured timeout periods (see section 7.10.4)

Registers listeners for log-on and log-off events

Listener 1: On log-on, stops any timers still running and starts InactivityTimer1

Listener 2: On successful re-log-on after Temporary Lock, stops any timers still running and starts InactivityTimer1

Listener 3: On log-off, stops all timers

InactivityTrackerBLO Timers:

InactivityTimer1

After configured period of inactivity, when it is safe to do so, locks the screen and starts InactivityTimer2

InactivityTimer2

After configured period of inactivity, logs the user off

InactivityTrackerBLO also provides a method executeLockScreen() to be called when a user locks the screen:

Stops InactivityTimer1 and starts InactivityTimer2

7.10.4 Configuration

There are two Counter configuration items for the Inactivity Timer.

Key	Default Value	Data Type	Location	Description
templock.ui.inactivity.milliseconds	900000*	Integer	Default configuration property file	The time in milliseconds of Counter inactivity after which the Counter will be locked.
templock.logout.inactivity.milliseconds	2700000**	Integer	Default configuration property file	The time in milliseconds of Counter inactivity after the Counter is locked after which the user will be logged out.

* - 15 mins

** - 45 mins – but should be 59 mins – a peak is being raised to amend.

7.11 Top Level Components

7.11.1 Introduction

This section covers the area of bootstrap and top-level functionality not covered in the architecture or use cases explicitly.

7.11.2 System Requirements

The system requirements are as follows:

1. To get the system started.
2. To enable a reference data update to cause ripping down of the architecture.
3. To allow the user to restart/reboot the counter
4. To enable Systems Mgt tooling to cause the counter to terminate (e.g. in readiness for a code update or a regular (3am) counter refresh.

7.11.3 Design Overview

The start up process is a layered process. In outline it is as follows:

Startup (for initialising threads and logging)

- Application manager (for Starting the subsystem managers)
  - ApplicationBLO (for initialising BLOs)
  - ApplicationHelperBLO (for starting the screen display)
    - Initial BLO (for dealing with the initial screen)
    - LoginBLO (for dealing with login).

There is more detail below.

7.11.4 Design

7.11.4.1 Startup

Note that startup is outside the scope of the architecture.

- Mainstart java entry point
- Sets up XML Parser
- Sets up configuration and log4j
- Sets up DOM watch delay
- Find and log the jar version nos
- Redirect Syslog and syserr to Log4j (if configured to do so)
- Create a startup thread group
- Create a startup thread
- Start Application Manager (executing asynchronously on the AWT event dispatching thread) via StartupThread and CounterUtilities

7.11.4.2 Application Manager

Note that the application manager is outside the scope of the architecture.

- Uses a port to ensure only one instance can run

- Create a UIContext
- Set up exception handler
- Create the Susbsystem Managers (RDM, RSIM, BSM, BDM, UIAM, PM, UIEM, UIWM, BLM) and create their contexts
- (Details of contexts).
- Create and run CounterIDBLO
- Create and initialise the SystemBDO
- Create and start the HelpContextIndexerBLO
- Create `PINPadEngineeringBLO`
- Create `RecordBusinessEventBLO` and register with RDM
- Set up reference data sources
- Initialise the RDM
- Create `ApplicationBLO` and run its first part `(executeBusinessLogic)`
- Start Polling
- Run second part of `ApplicationBLO (lateStartup)`
- Tell the UIWM to make the screen visible

The Application Manager also provides a shutdown interface which shuts all the managers, closes the port and exits the system.

7.11.4.3 Application BLO

The ApplicationBLO is a single-instance BLO in that it is created and used in one place only – but it does not use the SoleInstance mechanism for BLOs (see section 7.8) to achieve/enforce that.

Stage 1 (execute business logic)

- Create/initialise the following:

```
EventHubBLO();
PollingBLO();
SystemBDO();
BasketBDO();
PrinterBLO();
ActionDispatcherBLO();
AuditBLO();
RecoveryBLO();
CounterStatsMonitorBLO();
StatusBarBLO();
StepBLO();
ApplicationHelperBLO();
PerformanceMonitor();
PostalServicesBLO();
TokenInitialization();
InfoBarBLO();
HelpContentIndexerBLO();
```

Stage 2 (lateStartup)

- Create/initialise the following:

```
ActionFactoryBLO();
```

```
BasketBLO();
NumPadBLO();
PrinterBLO();
ModeBLO();
RatesBoardBLO();
InactivityTrackerBLO();
RemoteShutDownServer(counterSocket);
OnlineStatusBLO();
```

- Then the counter application is ready to run – so it sets it going using the ApplicationHelperBLO (resetDisplay method)

7.11.4.4 ApplicationHelperBLO

Reset Display.

Gets the application going by:

- Clearing screen (for safety)
- Create navBarBLO and display the home navbar
- Create cmdBarBLO and display the cmdBar
- Create and initialise softKeyboardBLO
- Create login screen (using LoginBLO)
- Use InitialBLO (displayWelcome method) to display the initial welcome screen.

7.11.4.5 Initial BLO

displayWelcome method

- Display welcome screen
- Display legal message screen (and deal with user's choice)
- Invoke loginBLO to process the rest of the login.

Between the IntialBLO and the LoginBLO the logic in the following diagram (Figure 7-7 - Counter Login overview) is maintained. Keeping the application running until one of the shutdown operations occurs (see following section).

7.11.4.6 LoginBLO

This is covered in *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060).

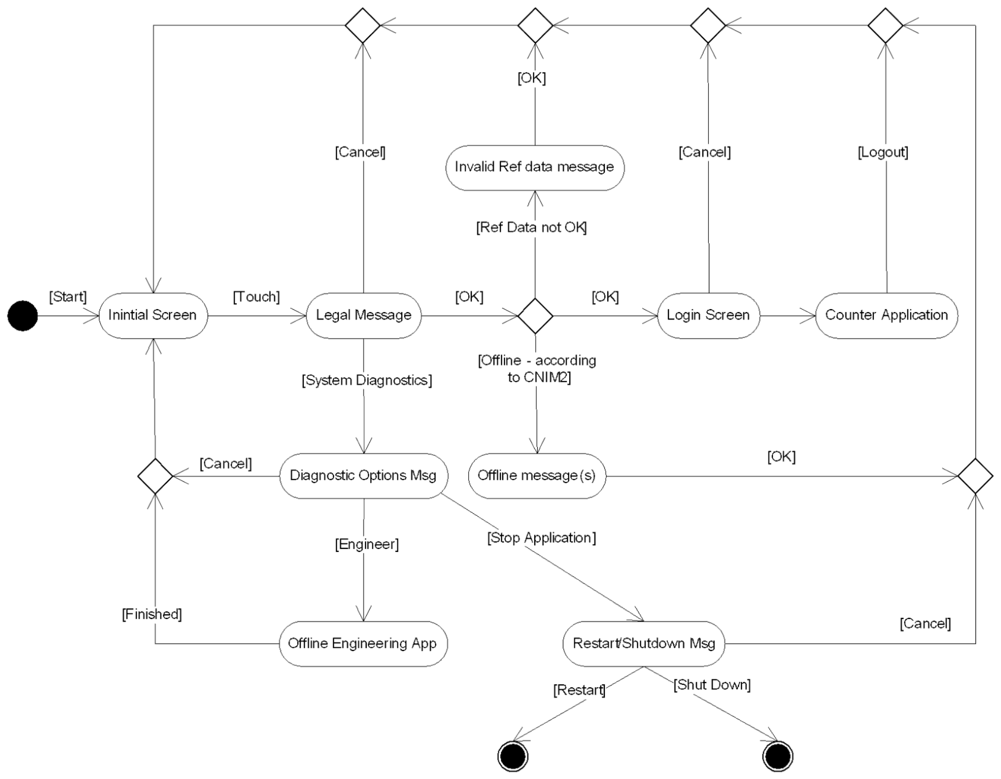


Figure 7-7 - Counter Login overview

7.11.4.7 Shutdown Process

The application stays running until one of the following happens:

- User selects (via system diagnostics) to shutdown the counter (exit code 50)
- User selects (via system diagnostics) to restart the counter (exit code 51)
- SM tools ask the counter to close down (exit code 0)
- The system crashes or fails in some way (exit code 0)

(Note – see DES/APP/HLD/0057 for how these exit codes are used).

Note that logout does not stop the application – it returns to the initial welcome screen)

To enable this the RemoteShutdownServer BLO (started by the ApplicationBLO) uses the RemoteShutdownServerThread BLO to listen on a port and fire off the ShutdownBLO if we are requested to shut down.

7.11.4.8 Contexts

The BLM requires access to the following managers, hence the '**BLO Context'** is :

- BLM
- UIAM
- BSM
- BDM
- RDM

See the BLOcontext diagram here for clarification.

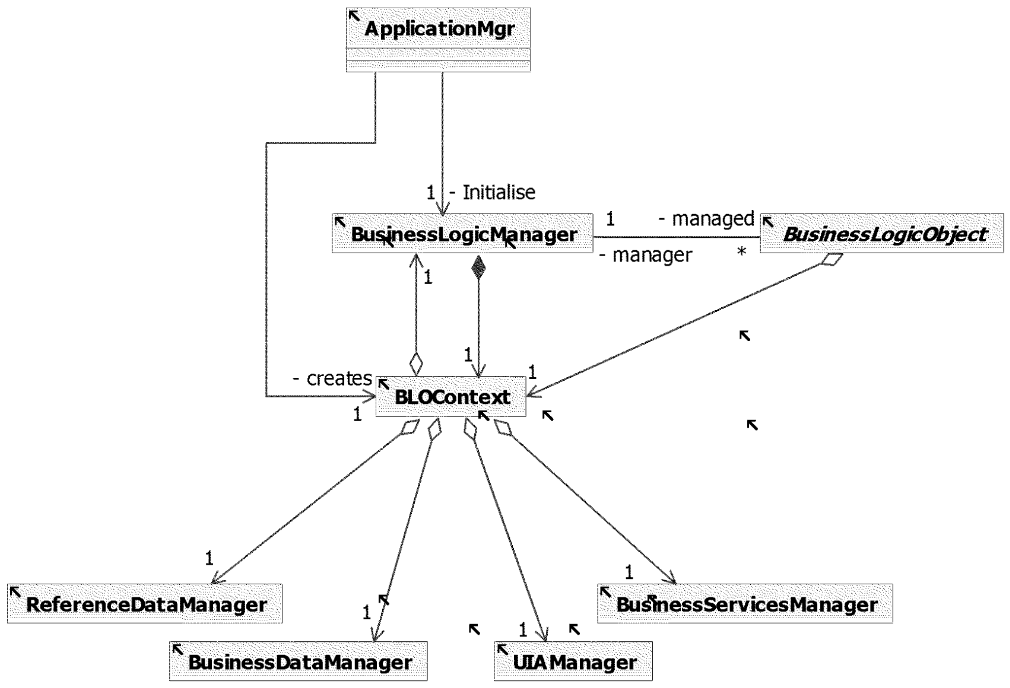


Figure 7-8 – BLOcontext Class Diagram

Similarly the BDM requires access to the following managers, hence the '**BDO Context'** is :

- BDM
- RSIM
- RDM

The BSM requires access to the following managers, hence the '**BS Context'** is :

- RSIM
- RDM

The UIAM requires access to the following managers, hence the '**UIA Context'** is :

- LDM (includes LDEMs, and LDWMs)
- RDM

The PM requires access to the following managers, hence the '**POU Context'** is :

- RDM

The RDM does not require the context of any other Framework manager.

The LDM requires access to the following managers, hence defining the '**LD Context'** as :

- PM
- RDM

The RSIM does not require the context of any other Framework manager.

Each Framework Manager class holds its required 'Framework context'.

The 'Framework context' for each Framework Manager is populated by the Application Manager that creates the Framework Managers.

7.11.5 Supporting Design diagrams

Three are no further supporting diagrams for this section as the Application Manager and ApplicationBLO interact with so many components that the diagrams become too unwieldy to be of use.

7.12 Modes

HNGX supports modes in a similar (but different) way to Horizon.

HNG-X Menu Hierarchy and Messages (DES/GEN/SPE/0007) defines how modes fit into the menu hierarchy.

7.12.1 Overview

Modes reflect the current operational context of the system.

Modes impact on the behaviour of various parts of the system, as follows:

- A product can be valid in one mode and not another (for example Christmas stamps are valid to be remmed out after Christmas, but not available for sale).
- Transactions exist in a Mode, (defined by the current counter mode) and the basket includes this mode in its Transactional information when it is submitted to the data centre at settlement time. (Specifically in the basket header). It also includes the transaction mode.
- A basket can only be in one mode at a time
- There are counter modes and transaction modes. The data centre understands transaction modes only (and these equate to the modes that Horizon had). The counter operates in counter modes, and then maps the counter mode to a transaction mode. In many case they are 1-1, but sometimes several counter modes map to one transaction mode – but never does one counter mode map explicitly to more than one transaction mode. However, when it comes to existing reversals the transaction mode will be whatever the mode of the original basket was so there is no predefined mapping there. The same is true of the recovery modes but in practice they can only map to one transaction mode (1 – Serve Customer).
- Transaction modes are in the range 1-34. Counter modes cover that range of numbers plus number over 100.
- There is a concept of not being in a mode – but this is represented by being in mode 0.
- The current mode is always displayed on the status bar.
- Modes are handled in HNG-X as integers.
- Modes are linked to menus in the menu hierarchy – see *HNG-X Menu Hierarchy and Messages* (DES/GEN/SPE/0007) for details.
- Buttons are linked to modes in two ways – valid-in and move-to. See *HNG-X Menu Hierarchy and Messages* (DES/GEN/SPE/0007) for details. A button is dimmed if it is not valid in the current mode. (If a button has a move-to mode, then it is considered as not being valid in any other mode).

- It is possible to "back out" of a mode if there is nothing in the basket. This is known as a "tentative" mode. When you enter a menu or press a button with a "move-to" mode, then a new mode value is set accordingly. If there is nothing in the basket, this is a "tentative" mode. When something is added into Basket, this then becomes a "firm" mode. (Note: When we are in tentative mode, the status bar still displays the mode name)

7.12.2 Design

ModeBDO is used for managing modes. It is a singleton BDO. It maintains a 'Current Mode', and is available for interrogation about the current mode and the mode parameters applicable at that moment. It is used in many places in the system, but MenuBLO is the place that normally sets the mode from the menu hierarchy. (Note the mode can be set in other places – e.g. recovery sets its own modes).

ModeBDO provides methods:

- getMode – returns the mode parameters values for the current mode (in the form of a ModeParemetersRDO).
- isCurrentModeValid (List of Modes) – returns true if the current mode is in the list of modes supplied.
- isModeFirm – returns true if there is something in the basket
- setMode (counter mode number) – makes that mode current if allowed (retrieving the mode parameters for that mode). Also fires off an event (MODE_CHANGED_EVENT_NAME) via the EventHubObserver (obtained via the BDM) to signify to interested parties that the mode had changed (e.g. to get the mode name displayed at the bottom of the screen)
- getModeNumber – returns the counter mode no of the current mode
- isNoMode – returns true if not in a mode
- setNoMode – sets the mode to no mode

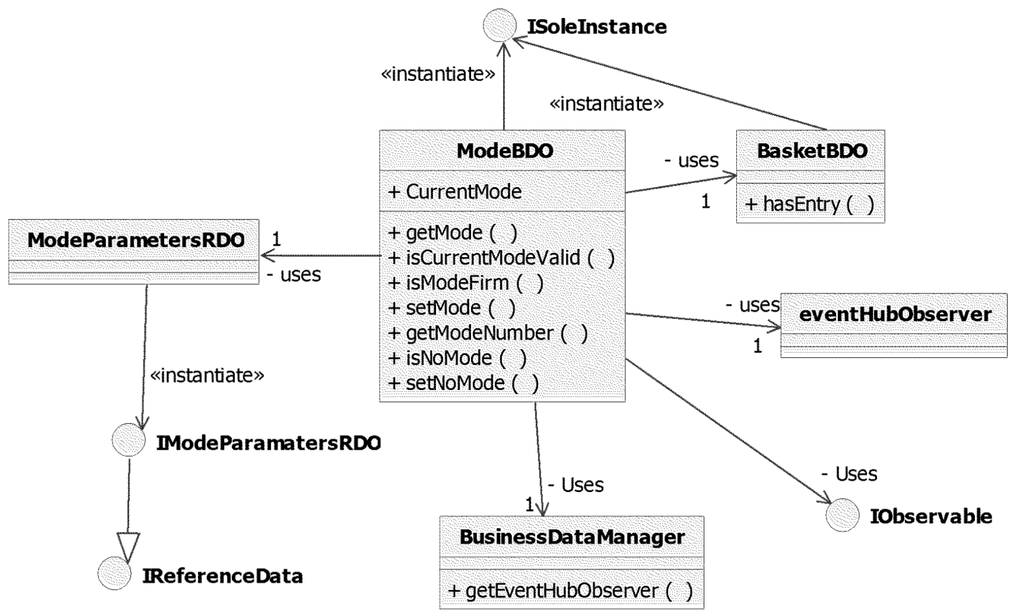
The transition between modes is recognised by Menu and button processing, ie navigation to a certain area of business (and occasionally buttons to start functions). At this point the ModeBDO is informed of the Mode change, the current Mode is updated, and the ModeBDO informs all the registered listeners of the Mode change.

7.12.3 Mode Parameters

There is in reference data a set of values for each mode. These values and their meanings are defined in *HNG-X Menu Hierarchy and Messages* (DES/GEN/SPE/0007).

When a mode is entered (tentative or otherwise) the mode BDO is populated with the contents of the menu parameters for that mode, and then those values are made available to other parts of the system that need to know about features of the current mode.

7.12.4 Diagram



7.12.5 Modes In HNGX

Below is a table that catalogues the current Horizon and HNG-X modes. Horizon modes that are not used in HNG-X are left in (highlighted) for information.

Txn Mode	HNGX Counter Mode	Acronym	Name	Comment
0	0		(Default – no text)	= No Mode
1	1	SC	Serve Customer	
2			Remit In - Hemel SSO	Not used
3			Remit Out - Hemel SSO	Not used
4	4	RU	Revalue	(Comes from code not from menus)
5	5	RD	Revalue (down)	Not used RU/RD merged using +ve/-ve settlement products.
6			Transfer (item to item)	Not used
7	7	TI	Transfer In	
8		RIOP	Remit In - Other Offices	Not used
9		ROOP	Remit Out - Other Offices	Not used

10	10	RICL	Remit In from Client	
11	11	ROCL	Remit Out to Client	
12	12	RODC	Remit Out to EDS	
13	13	TO	Transfer Out	
13	113		Transfer Reversal	When reversing a transfer out.
14		REC	Bulk Input	Not used
15	15	HK	Housekeeping	
15	115	HK(LSP)	Housekeeping	Use to resolve discrepancies to local suspense
16	16	SAP	Stock Adjustment	(Comes from code not from menus)
16	116	SAP(WDP)	Stock Adjustment	(Comes from code not from menus) Used to resolve withdrawn products.
17	17	DDP	Declaration Discrepancy	(Comes from code not from menus)
18	18	SAN	Stock Adjustment - Negative	Not used SAP/SAN merged using +ve/-ve settlement products.
19	19	DDN	Declaration discrepancy - Neg	Not used DDN/DDP merged using +ve/-ve settlement products
20	20	EVNT	Event. At the counter it is used for PDR "transactions" when generating Giro Reports.	
21		I	Can Item Be Ordered By Outlets	Not used
22		PT	Parcel Traffic	Not used
23		NAD	Non Accounting Data	Not used
24	24	RIAD	Rem in Auto Dist	Remit in from ADC – not used as a counter mode in HNGX (111/112 are used instead)
24	111		Rem in from ADC	With Basket
24	112		Rem in from ADC	Without Basket
25	25	ROAD	Rem Out Auto Dist	
25	101		Rem Out to ADC (Stock)	
25	102		Rem Out to ADC	

			(Cheques)	
25	103		Rem Out to ADC (Branch Cash)	
25	104		Rem Out to ADC (Branch Stock)	
25	105		Rem Out to ADC (Branch Currency)	
26		ROSP	Remit Out Stock from Stock Unit	Not used (106-8 used instead)
26	106		Rem out to Pouches (Notes)	
26	107		Rem out to Pouches (Coins)	
26	108		Rem out to Pouches (Currency)	
27		RISP	Reverse Pouch	
28		RODP	Despatch Pouches	
29		MG	Accept Now (Transaction Corrections)	AKA Make Good (Comes from code not from menus)
30		HD	Settle Centrally (Transaction Corrections)	AKA Plead Hardship (Comes from code not from menus)
31		WO	Send To Profit and Loss (Transaction Corrections)	AKA Write Off (Comes from code not from menus)
32		AN	Assign to Head Office (Transaction Corrections)	AKA Assign to Nominee (Comes from code not from menus)
33		EV	Seek Evidence (Transaction Corrections)	(Comes from code not from menus)
34		SW	Stock Write Off Transaction Corrections)	(Comes from code not from menus)
	109	RV	Reversal / Refund	Uses Txn mode 1
	110	ER	Existing Reversal	Linked to original transaction and its mode.
	117	Recov(New Reversal)	Recovery Reversal	Used when recovering a new reversal basket. Uses Txn Mode 1
	118	Recov	Recovery	Used when recovering, when a

				reversing cancellable transactions. Uses Txn Mode 1
	119	Recov(rfwd)	Recovery – Rollforward	Used when recovering a non-reversal basket by rolling forward (i.e. running the recovery scripts). Uses Txn Mode 1
	120	Recov(Existing Reversal)	Recovery Reversal	Used when recovering an existing reversal basket. Uses Txn Mode 1
14	14	TA	Transaction Acceptance	Used for processing Transaction Acknowledgments
10	121	RICL	Rem In from Client (no basket)	Used for processing Transaction Acknowledgments

Notes:

Instead of having ROAD for Rem out Stock and Cheque, and ROSP for Notes/Coins/Currency, we have 6 separate modes (one each for Stock, Cheque, Notes, Coins, Currency and Other).

There are 3 new modes for pouches going branch-to-branch under ROAD.

7.13 Roles and Permissions

The roles and their permissions are all defined in reference data (object Type RoleCapabilities), and they are described in *HNG-X Menu Hierarchy and Messages* (DES/GEN/SPE/0007).

This section deals with the design of how they are implemented.

7.13.1 Overview

Any user can have one role. When the user logs in the BAL tells the counter what role that user has (e.g. Clerk).

The counter can then use that role to examine reference data and determine what operations that user has permission to perform. This is done by defining a set of "Capabilities" for that role.

In most cases the permissions are checked as a result of using an action that requires a defined capability (see reference object type Action). This is then checked by the action despatcher using the RoleBLO. (Note that permissions are not defined in the menu hierarchy – instead the menu hierarchy refers to actions that it wants to perform and the capabilities are linked to actions).

In addition the capabilities for the current role are picked up by the MenuBLO and configured into the MenuUIA so that the menu buttons can be disabled if the user does not have permission to use that button. (It does this because UIA's are not allowed to talk to a BLO – i.e. the RolesBLO)

7.13.2 Design

The RolesBLO is a singleton object and has the following methods:

- isAuthorised (capability) – returns true if this user has that capability
- getcurrentUserRole – returns the user's role (or NOTLOGGEDIN if no user is logged in)
- getCurrentUserCapabilities – returns a set of capabilities for the user logged in

- isGlobalUserCreator – returns true if user is a global user (by looking in reference data object type ManageRolesCapabilities).
- isGlobalAdminRole – returns true if user role is ADMIN.
- canManageOtherUsers (role) – returns true if this user can manage other users (by looking in reference data object type ManageRolesCapabilities).
- getManagerString – returns MANAGERS

7.14 Session Suspend/Resume

7.14.1 Overview

1. CP 4382 (HNGX-0016) introduced basics.
2. CP 4530 (HNG-X CP0082) extended this.

Use cases GLB-5663 Perform Suspend Customer Session and GLB-5697 Resume Suspended Customer Session cover this area. See *UCR Document for Shared Use Case Barrel High Level Design (*DES/APP/HLD/0068).

This is not really "Session Suspend", but the capability of temporarily Settling an Incomplete Session and then resuming that session with the previous "Running total" so that at the end all items in the "chain of sessions" can be Settled to a single Settlement item (particularly beneficial if it is Plastic).

7.14.2 In summary

7.14.2.1 Suspend

1. There will be an Option in the right hand Nav Bar to Suspend a Session. Only Serve-Customer mode baskets can be suspended.
2. If Selected, this will check all items currently in the basket against Ref Data to see if they are supported in a Suspended Session. (This will use a similar technique to that used for checking if Debit / Credit cards are allowed and Bureau / Retail cards.) .It will also check for the suspend product being available at this branch and will stop if there are recoverable txns in the basket.
3. If all is OK, the Basket will be settled to a special "Suspend Settlement" Product, and the session added to a list of suspended sessions for this stock unit (in the BRDB).
4. The user is asked if he is sure he wants to suspend, highlighting his liability.
5. There is a check that the limit of the "Suspend Settlement" product is not being breached.
6. A zero value or empty basket may be suspended – but that results in a zero-value basket entry for the "Suspend Settlement" product.

7.14.2.2 Resume

1. There will also be an Option to resume a previously suspended session (only available in serve customer mode).
2. If selected, this will Display a list of all Suspended sessions for this SU.
3. If one is selected, then a normal SC Basket is initiated with the "Running Total" as an initial Basket Entry against the "Suspend Settlement" Product (ie this will cancel out with the previous entry in the Suspended session in the accounts)..
4. That Basket then behaves like any other SC basket, and in particular can itself be Suspended.
5. Once the Basket Resuming a Suspended session is Committed, then the original suspended Session is marked as "Resumed" and so will not appear when doing a further "Resume". (Note there is a race condition to be wary of here – if two users try to resume the same session)

7.14.2.3 Notes

EOS Prompts are not required when doing a suspend.

Baskets can be suspended if they are +ve, 0 or –ve.

No suspends are allowed in the default stock unit.

Suspend and resume will use the same Product id for adding things to the basket.

It will be the definition of the Suspend product id that will prevent it being deleted from the basket.

7.14.3 Design

7.14.3.1 Data

The basket header has the following fields which are used here:

- SuspendedSessionAmount – this is the value of this session at the time it is suspended (i.e. the inverse of the amount in the final balancing txn allocated to the new product). It is set by the suspend logic on the counter and used in the BAL to discern and record the value of the suspended session. Note this can be +ve, -ve or zero – do not rely on this being zero to indicate if the session is being suspended – it must be null for that.

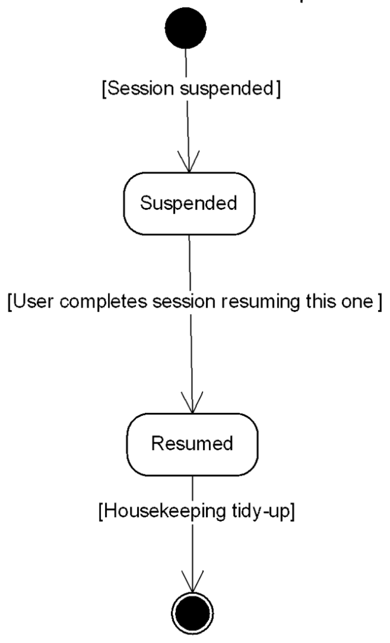
Note that the basket header also contains the following fields, but they are no longer used – as it was decided to add the resumed session id and resumed counter to the accounting line instead.

- ResumedSessionId – this is the session id of any session that this session is continuing from. It is set by the counter when resuming, and is used by the BAL to discern what session to "seal" as resumed when this session is settled (or suspended). (Note: this only signifies the current session is a resumption of an earlier session – it is not indicating if this session is suspended or not).
- ResumedCounter – this is the counter who created the session referred to by the Resumed SessionId. It is included to ensure that the session id is unique. It will only be set if the ResumedSessionId is set.

Mode Parameters has a field to indicate the MOP product (i.e. the Suspend Settlement product id – `21650`). Set just for Serve Customer – blank for all other modes.

In BRDB the table of suspended session has a status and a Node id on it. The status can take one of the following states:

- Suspended – set when the session is added to the table when the session has been suspended.
- Resumed – set when the resumed session is settled/suspended.



7.14.3.2 Invocation

(See pdl ResumeSuspendBLO)

There will be a button on the RH cmd bar labelled “Suspend Resume Lock”. Pressing this will have the following effect.

It will display a new screen with just 3 action buttons on it- Suspend, Resume, Lock.

If the mode is not serve customer

OR

This user does not have Transactions permissions

OR

This user is in the Default Stock Unit

Then

Only the Lock option is enabled (suspend and resume disabled).

Else if the basket is not empty

Then

Only the Suspend and Lock Options are enabled (Resume disabled)

Else

Suspend, Resume and Lock are enabled

If the user selects cancel or prev then the user is returned to where he was before.

If the user selects Lock – then we do templock

If the user selects Resume then obey resume logic (below)

If he selects Suspend then obey Suspend logic (below)

7.14.3.3 Suspend Logic

(See SuspensionBLO)

(Note: the above invocation logic will have taken care of permissions and mode restrictions)

Get the identity of the Suspend Settlement Product from Mode Parameters. If it does not exist, then tell the user that suspend is not allowed at this branch and stop.

Check the basket for items incompatible with suspending. Error if anything incompatible and stop. This is the process of examining inclusion and exclusion lists for the Suspend Product as follows:)

- Look for a ref data object called “SettlementRule” with a name of “Suspend”
- For the Excluded Groups (if any), form an exclusion list of products from all the exclusion groups, then check every item in the basket to see if it is in the exclusion list - if any one of them is in the list we can’t suspend this basket.
- For the Included Groups (if any), form an inclusion list of products from all the inclusion groups, then check that every item in the basket is in the inclusion list – if any one of them is not in the list then we can’t suspend this basket.

Example showing format of the InclusionExclusionRule:

<RDObject Type="SettlementRule" Key="Suspend" EffDate="2007-01-30" Action="S" State="E">

<RData>

```
<SettlementRuleData>
<SettlementRule>
        <RuleName>Exclude From Suspend</RuleName>
        <Inclusive>N</Inclusive>
        <ProductGroups>
        <ProductGroup>6796</ProductGroup>
        <ProductGroup>6797</ProductGroup>
        <ProductGroup>6799</ProductGroup>
        </ProductGroups>
</SettlementRule>
<SettlementRule>
        <RuleName>Include In Suspend</RuleName>
        <Inclusive>Y</Inclusive>
        <ProductGroups>
        <ProductGroup>6796</ProductGroup>
        </ProductGroups>
</SettlementRule>
</SettlementRuleData>
</RData>
</RDObject>
```

Check through the basket to see if any items are recoverable (i.e. if there are any accounting lines with a USN set). If there are any then advise the user that he cannot suspend this session and stop.

Calculate the value of the basket (allowing it to be empty or zero value)

Put up a liability message – and stop if he says no.

Add a Settlement item into the basket for the complete basket value - even for zero-value baskets. This will check if it is not too big for the Maximum value of the Suspend Settlement product. (Error and stop if it is)

Set the basket header SuspendedSessionAmount = Value of the basket (without the suspend settlement product)

Settle the basket using AS570 (Expanded Settle) setting Force-Settle = True, User Present = True and GoHome = True.

(Note that this can fail if the session has already been resumed on another counter)

End

7.14.3.4 Resume Logic

(See pdl ResumeSuspendBLO)

(Note: the above invocation logic will have taken care of permissions and mode restrictions, and that the basket will be empty, and that we are not in the default stock unit)

Go to BAL and get list of sessions, (and their values, and counters) that are suspended for this Stock Unit. (Note this list should include the Username too so it can be used elsewhere – date and time of suspension too). If there are none, tell the user and stop.

Display the list to the user (showing Session id, User, amount and date/time for each one - ordered in date order, oldest first) and get him to select one.

(It is possible for him to cancel or prev at this point).

Flush Basket (as a precaution)

Get the value of the Suspend Settlement Product from Mode Parameters

Add into the basket one item using the product id of the Suspend Settlement Product and the value of the chosen session. (Ensuring the sign is retained correctly)

Set the ResumedSessionId* = the chosen session id.

Set the ResumedCounter* = counter of the chosen session.

(*was in the basket header – now in the accounting line)

Return to the menu system at the point the user left it.

7.14.3.5 Other Parts of the Design

These relate to CP (4530 HNGX CP 82).

1. Users should be prevented from Logging Out of the system if they have any sessions that have been Suspended by them in the currently Attached Stock Unit
2. A User attaching him or herself or another User to a different Stock Unit is warned of any Suspended Sessions associated with the User whose attachment is being changed and the original Stock Unit before the attachment is allowed to complete.
3. Stock Unit Balancing should not be allowed to complete if there are any Suspended Sessions (for any user) associated with the Stock Unit.

Note that the following are allowed:

- Any Forced Log Off due to a timeout will continue regardless of Suspended Sessions
- Any Forced Log Out due to a comms failure will continue regardless of Suspended Sessions
- Any Forced Log Out due to the User Logging On at another terminal without Logging Off first will continue regardless of Suspended Sessions

7.14.3.6 Recovery

Suspending and resuming sessions has some impacts on recovery. This is all covered in *HNG-X Counter Subsystem : Recovery Management* (DES/APP/HLD/0083).

7.15 Tokens

The counter application has various devices attached to it including a barcode reader and magnetic card reader.

The term "Tokens" includes barcodes and magnetic swipe cards (but not smart cards in the pinpad) Note that the smart card reader in the keyboard is not supported.

Tokens are analysed against reference data to validate and identify them by the tokeniser. The operation of the tokeniser is described in *HNG-X Tokens System Low Level Design* (DEV/APP/LLD/0192).

There are two ways that the tokens are used:

- at the request of a use case, in which case it is known as a solicited token, and the tokeniser is invoked by the use case, which nominates the relevant tokenset
- The user swiping a token to initiate a transaction (in which case it is known as unsolicited). This triggers the tokeniser to be called in order to validate the data. In this case the valid tokenset is nominated by the Menu Hierarchy as described in section 7.15, and the tokeniser is invoked via the SelectFunctionOptionsBLO.

The user can use a token at any moment while the counter application is running. In order to capture the devices input, listener events are associated with these devices which wait until an action has occurred. This alerts a PeripheralInputActionListener instance which will fire off the action "ProcessTokenPeripheral". This action gets processed by the ActionDespatcher in the normal way and that will start the SelectFunctionOptionsBLO.

This BLO then assembles the relevant information inside the action in order to pass it through to the tokeniser by calling the TokenBLO method matchToken(). Upon completion the TokenBLO will return an IToken object. This object contains a flag to determine if the tokeniser checks were successfully passed. It also contains any data that was held in reference data that is related to the token which could be used later in the relevant use case and data as to which BLO to invoke in order to process the token. If the token was invalid SelectFunctionOptionsBLO reads the IToken object to obtain the error message id and displays the message to the end user through the UI framework.

Note that the current valid tokenset is nominated by the Menu Hierarchy as described in section 7.7.

7.16 Counter Persistence

7.16.1 Overview

The counters need to store certain counter-specific data. As this cannot be stored on the counters own local filestore, provision is made for each counter to be able to store and maintain its own data on the BRDB. This data includes (but is not limited to):

- Serial no of Pinpad installed on this counter
- Rates board configuration
- Flag to indicate if EOS prompts are suppressed.

The design of the counter persistence data is such that the format of the data is owned by the counter (not the BRDB schema) so it can easily be extended. (in essence it is a structured "CLOB").

(Note that this data is not completely private to the counter at present since it is generated from Horizon as part of migration. However once rollout is complete it is purely under counter control.

The stored data is private to each counter as there is no requirement for the data to be shared across different counters. Storing the data separately for each counter also avoids any possibility of update clashes when writing to the database.

7.16.1.1 Migration

In addition, there is existing counter persistence data that is migrated from Horizon to HNG-X systems. This Horizon data must be loaded into the HNG-X database in a format that can be read by the Counter (i.e. a CLOB). The format of this migration data is described in DES/APP/HLD/0113.

7.16.2 Design

The CounterPersistence Application Service is written in Java and provides a mechanism that can store and retrieve persistent data for each counter. It allows read, write and delete operations to be performed using a unique key on this data.

The first call to either read, write or delete will create a local counter data store that is loaded from the BRDB database. Note that this local data store is held in memory and is never written to disk on the counter at any time.

Read operations are permitted either inside or outside a user session, but write and delete operations are only permitted when inside a user session.

Any change to the local counter data store by calling write or delete are immediately written back to the BRDB database by sending the whole contents of the data store to the database and storing it as a single CLOB in table BRDB_BRANCH_NODE_INFO.

Both write and delete operations are auditable events.

The following functions are provided:-

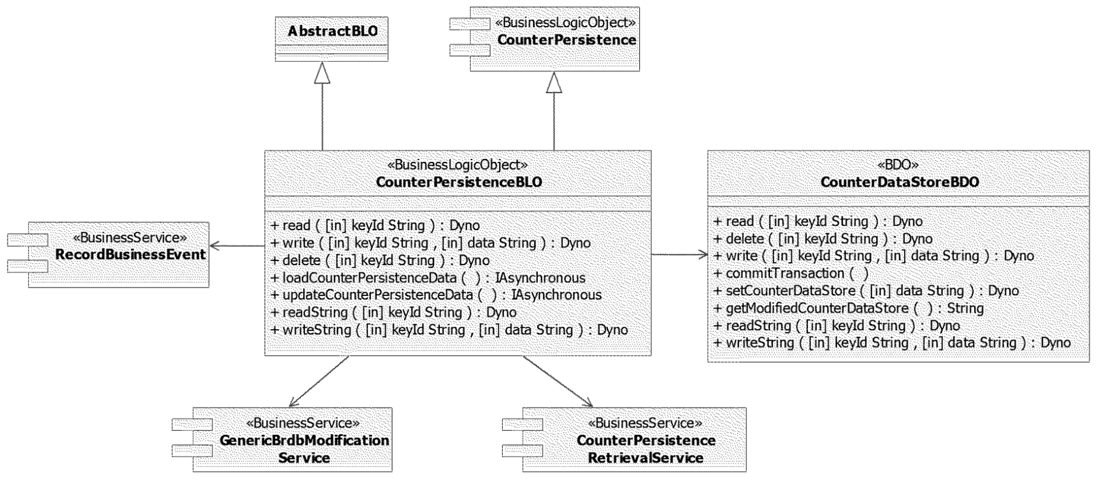
1. Read data from counter data store using a unique keyId.
2. Write data to counter data store using a unique keyId - adds or overwrites data as required (auditable event).
3. Delete data from counter data store using a unique keyId (auditable event).

Note: Data can be any Object type such as String, List, Integer, Boolean or a Dyno which can itself contain further Object types.

The CounterPersistence Application Service has no interaction with the UI.

7.16.3 Model

The following diagram shows how the CounterPersistence Application Service interacts with other components on the counter.


AbstractBLO
«BusinessLogicObject» CounterPersistence
«BusinessLogicObject» CounterPersistenceBLO
+ read ([in] keyId String) : Dyno
+ write ([in] keyId String , [in] data String) : Dyno
+ delete ([in] keyId String) : Dyno
+ loadCounterPersistenceData ( ) : IAsynchronous
+ updateCounterPersistenceData ( ) : IAsynchronous
+ readString ([in] keyId String) : Dyno
+ writeString ([in] keyId String , [in] data String) : Dyno
«BDO» CounterDataStoreBDO
+ read ([in] keyId String) : Dyno
+ delete ([in] keyId String) : Dyno
+ write ([in] keyId String , [in] data String) : Dyno
+ commitTransaction ( )
+ setCounterDataStore ([in] data String) : Dyno
+ getModifiedCounterDataStore ( ) : String
+ readString ([in] keyId String) : Dyno
+ writeString ([in] keyId String , [in] data String) : Dyno
«BusinessService» RecordBusinessEvent
«BusinessService» GenericBrdbModification Service
«BusinessService» CounterPersistence RetrievalService

7.16.3.1 CounterPersistenceBLO

CounterPersistenceBLO is a singleton BLO that provides the main API for read, write and delete operations. These methods must be 'thread safe' as they may be called from different threads on the counter. Calling either read, write or delete for the first time will invoke the loadCounterPersistenceData() method that will load the counter data from the BRDB database. The data is stored in the database as a single CLOB in an XML format as defined in section 0.1. The counter receives this data as an XML string which is then converted into a Dyno to become the data store repository. A unique keyId is then used to access and perform operations on this data using the read, write and delete methods.

The loadCounterPersistenceData() method cannot use GenericBrdbRetrievalService because read operations need to be available outside of a user session. This would result in the request message being unsigned and would fail if sent to the BAL. To allow for this unsigned request message and for security, counter persistent data is retrieved using a bespoke service handler CounterPersistenceRetrievalService that can only retrieve this type of counter persistence data.

The GenericBrdbModificationService component is used to provide the updateCounterPersistenceData() operation which is always invoked within a user session so the request message will be signed and will work correctly when sent to the BAL. These requests will be audited on the BAL using the RecordBusinessEvent Business Service.

Calls to write or delete will invoke the updateCounterPersistenceData() method that sends the changes to the local data store to the BRDB database. During these updates, the entire counter persistence data store is sent to the BAL as XML and stored in the database as a single CLOB containing the XML. Write or delete operations are performed on the counter within a transaction and the data store will be rolled back if the update to the BRDB fails. The commitTransaction() method is only called if the update to the BRDB has been successful.

It creates and maintains the CounterPersistenceBDO at start-up.

7.16.3.2 CounterPersistenceBDO

The CounterPersistenceBDO component provides the local counter data store repository. It receives the data in the form of a Dyno from CounterPersistenceBLO and provides the necessary read, write and delete operations on this data. When updating the BRDB database, the entire counter data store is sent to the BAL as XML and stored in the database as a single CLOB as defined in section 0.1.

7.16.3.3 Methods

CounterPersistenceBLO read():

Reads a data Object from the local counter data store using a unique key 'keyId' to identify the required data which is contained in the returned Dyno with key 'result'.

CounterPersistenceBLO read() Parameters:

Key	Value Type	Description
keyId	String	This is a unique Id that is used to identify the stored data Object to be retrieved, eg, 'RateBoard', 'PinPadSN', 'EosFlag' etc.

Pre-Conditions:

keyId exists	The local counter data store should already be available on the counter or be available to be loaded from the BRDB database. Data should also have been previously stored for the specified keyId.

Post-Conditions:

Key	Value Type	Description
result	Object	Returned Dyno contains the required data Object using the key 'result' and will be null if the specified data is not found. Any java Object can be read using this method, eg, Boolean, Integer, String, Dyno etc. If an error occurred, the returned Dyno will contain key 'error' that has a String value that describes the error. If the data store cannot be loaded from the BRDB database, the error value will contain the text "Failed to load counter data from database: <reason>".

CounterPersistenceBLO readString():

Reads a data String from the local counter data store using a unique key 'keyId' to identify the required data which is contained in the returned Dyno with key 'result'.

CounterPersistenceBLO readString() Parameters:

Key	Value Type	Description
keyId	String	This is a unique Id that is used to identify the stored data String to be retrieved, eg, 'RateBoard', 'PinPadSN', 'EosFlag' etc.

Pre-Conditions:

keyId exists	The local counter data store should already be available on the counter or be available to be loaded from the BRDB database. Data should also have

	been previously stored for the specified keyId.

Post-Conditions:

Key	Value Type	Description
result	String	Returned Dyno contains the required data String using the key 'result' and will be null if the specified data is not found. Only java Strings can be read using this method. If an error occurred, the returned Dyno will contain key 'error' that has a String value that describes the error. If the data store cannot be loaded from the BRDB database, the error value will contain the text "Failed to load counter data from database: <reason>".

CounterPersistenceBLO write():

Writes a data Object to the local counter data store using a unique key 'keyId' to identify the data.

CounterPersistenceBLO write() Parameters:

Key	Value Type	Description
keyId	String	This is a unique Id that is used to identify the data Object to be stored, eg, 'RateBoardConfig', 'PinPadSN', 'EosFlag' etc.
data	Object	This is the data Object to be stored. Any java Object is allowed, eg, Boolean, Integer, String, Dyno etc.

Pre-Conditions:

keyId exists	The local counter data store should already be available on the counter or be available to be loaded from the BRDB database. If the keyId already exists, then the data will be overwritten without warning. If required, a check for existing data for a specific keyId can be made by calling the read() method for that keyId.

Post-Conditions:

Key	Value Type	Description
result	Boolean	Returned Dyno contains a Boolean status Object using the key 'result'. This will be 'true' if the write was successful or 'false' otherwise. If successful, the data changes will have been written to the BRDB database. If an error occurred, the returned Dyno will contain a key 'error' that has a String value that describes the error and all changes will be rolled back. If the data store cannot be loaded from the BRDB database, the error value will contain the text "Failed to load counter data from database: <reason>".

CounterPersistenceBLO writeString():

Writes a data String to the local counter data store using a unique key 'keyId' to identify the data.

CounterPersistenceBLO writeString() Parameters:

Key	Value Type	Description
keyId	String	This is a unique Id that is used to identify the data Object to be stored, eg, 'RateBoardConfig', 'PinPadSN', 'EosFlag' etc.
data	String	This is the data String to be stored. Only java Strings can be written using this method.

Pre-Conditions:

keyId exists	The local counter data store should already be available on the counter or be available to be loaded from the BRDB database. If the keyId already exists, then the data will be overwritten without warning. If required, a check for existing data for a specific keyId can be made by calling the read() method for that keyId.

Post-Conditions:

Key	Value Type	Description
result	Boolean	Returned Dyno contains a Boolean status Object using the key 'result'. This will be 'true' if the write was successful or 'false' otherwise. If successful, the data changes will have been written to the BRDB database. If an error occurred, the returned Dyno will contain a key 'error' that has a String value that describes the error and all changes will be rolled back. If the data store cannot be loaded from the BRDB database, the error value will contain the text "Failed to load counter data from database: <reason>".

CounterPersistenceBLO delete():

Deletes a data Object in the local counter data store using a unique key 'keyId' to identify the data.

CounterPersistenceBLO delete() Parameters:

Key	**Value Type**	**Description**
keyId	String	This is a unique Id that is used to identify the stored data Object to be deleted, eg, 'RateBoard', 'PinPadSN', 'EosFlag' etc.

Pre-Conditions:

keyId exists	The local counter data store should already be available on the counter or be available to be loaded from the BRDB database. If no data is found for the specified keyId, then the operation will have deemed to have been successful. If required, a check for existing data for a specific keyId can be made by calling the read() method for that keyId.

Post-Conditions:

Key	Value Type	Description
result	Boolean	Returned Dyno contains a Boolean status Object using the key 'result'. This will be 'true' if the delete was successful or 'false' otherwise. If successful, the data changes will have been written to the BRDB database. If an error occurred, the returned Dyno will contain a key 'error' that has a String value that describes the error and all changes will be rolled back. If the data store cannot be loaded from the BRDB database, the error value will contain the text "Failed to load counter data from database: <reason>".

7.16.3.4 Components

The following components are used by this design, but are documented elsewhere:

Name	Classification	Description
CounterPersistenceRetrievalService	BAL HLD: BusinessService	This service is used to provide the required 'load' BRDB database operation for counter persistence data.
GenericBrdbModificationService	BAL HLD: BusinessService	This service is used to provide the required 'update' BRDB database operation.

7.17 Product Names Usage in HNGX Counter

LongName	Used when showing the basket on the screen (including viewing full basket). Used in picklists and PLU list. Used in back office reports for suspense account and currencies in pouches. Used in AP transactions when displaying the product discerned from the token (whether swiped or manually entered). Used in error messages.
ShortName	Not used. (Note: - in Horizon this was used for the names on the buttons but in HNGX that is no longer true – the button names in HNGX are input directly to the menu hierarchy spreadsheet.)
ReceiptName (aka Medium Name)	Used on receipts and tally roll reports. Used on back office reports other than above.

Note – APADC makes all 3 names available for an APDC script to use so we can’t list above all the uses the APADC scripts might have made of the names.

7.18 Parser/Serialiser

We have to communicate java data objects between the separate parts of the HNG-X system, i.e. the Counter application and the Data-Centre (Branch Access Layer). XML is the format of the communication, and as such, it is necessary to have a component that is capable of taking a POJO (Plain-old-Java-object) and creating an XML representation of the structure and content of that object (plus objects nested within that object) ready for transmission, and another component that is capable of the reverse process (take the XML and create a populated (set of) POJO(s) the XML represents). These components are known as the XML Serialiser and XML Parser respectively.

The SAX model is used to process the XML, rather than DOM due to the lesser memory requirements made by SAX, and increased speed.

The parser/serialiser uses a mappings file to relate the java objects their XML representation.

For a detailed explanation of how the parser/serialiser work see DES/APP/LLD/0042.

This component is also used by the Reference Data Subsystem to process the Reference Data XML files. For a description of how the RD subsystem uses the Parser, please refer to DES/APP/HLD/0045.

7.19 Global Users

The counter app supports local users (Manager, Supervisor and Clerk) who are managed by the local branch manager – and whose scope is just the branch (all counters in the branch).

In addition there are global users who can login at any counter in any branch (subject to certain limitations – e.g. a training user can only login at a training counter). These are maintained by the ADMIN user on a special terminal which runs a version of the counter application to enable them to manage the global users.

Global users are described in *Online Internal Applications Architecture* (ARC/APP/ARC/0006) and *UCR Document for Branch Administration Use Case Barrel High Level Design* (DES/APP/HLD/0060).

7.20 Counter Error Handling Strategy

7.20.1 Overview

Errors are things that happen that are not necessarily expected. At a simplistic level they can be divided into two types:

- Something that has happened that has been anticipated which is covered by the Use Case
- Something that has happened which is not anticipated and so is not covered by the Use Case.

This section describes the strategy used on the counter for handling errors.

7.20.2 General Strategy

The counter is designed to be run up and to stay running indefinitely (i.e. until it is told to stop running – e.g. by SM tooling stopping it or by the user shutting down or re-booting the counter).

Thus any error should not cause the counter to become unusable at any time. The user may be forced to logout – or he may be returned to a menu. But he should always be able to continue with the application.

(Note some errors can prevent a user logging on – e.g. network down or ref data out of date, but these are transitory and will not require the counter application to restart)

7.20.3 Anticipated errors

In most cases there are user errors. They are handled (as defined in each use case) by displaying a message to the user which he can acknowledge (or choose an option) and the use case proceeds. No further consideration is given to these errors in this section.

7.20.4 Unexpected errors

There are a number of ways in which such errors could occur:

- Timeout when contacting BAL
- Inconsistent Reference Data
- Unexpected Response from BAL
- Unexpected Response from another component (internal or external component).
- Coding/logic error

The most likely source of such errors is either a problem with Reference Data set up or a bug in some piece of code / design or a network problem. In most such cases it is unlikely that retrying will improve on the situation and so the User will need to be informed, the error message logged and the current function should be abandoned. The only exception to this approach is a BAL timeout where retrying is appropriate.

These can happen when the user is operating the counter (foreground errors) or can happen in processes operating in the background.

These are considered separately below.

7.20.4.1 BAL Timeouts

The action depends on whether the message is a journalised one or not. (Note – the reason for this is that to ensure journal sequence numbers (JSNs) are unique and that there are no gaps in JSNs it is necessary to logout the counter so the sequence can be sorted out when the user logs back in).

7.20.4.1.1 BAL Timeouts on Journalised message

This situation has been covered as part of the standard BAL / Counter design. The approach is already well documented elsewhere. In summary:

- System automatically does one retry
- If that still fails, User is invited to Retry or Cancel
- User Retry will result in the function being retried and if necessary a further automatic retry and if it still fails the User is again invited to Retry or Cancel (ie return to the previous step). This can continue until the function succeeds or the User selects Cancel
- User Cancel will result in specific failure actions being taken (eg, in the case of a failed settlement of a Customer Basket this will result in a recovery receipt being produced) followed by a Local Log Out. No further attempt is made to communicate with the BAL until the Log Out is complete and a User then attempts a Log On.
- If the User does nothing, then eventually the inactivity timer will kick in which will initially Temporarily Lock the terminal and eventually force an Inactivity Log Out. Such a Log Out will attempt to contact the Data Centre and could result in any basket being successfully committed and a tidy Log Out occurring, thus not invoking Recovery at the next Log On.

7.20.4.1.2 BAL Timeouts on non-journalised messages

NB this includes a failure to connect to a working BAL even though it isn't strictly a timeout.

The assumption here is that the failure of a non-journalised message is not necessarily fatal since it has not affected the audit trail. Therefore handling of such failures is pushed back to the Business Logic rather than being handled as part of the counter framework.

The general approach should be as follows:

- Abandon the current function advising the user that there has been a temporary failure. There is no need to Log this as an Error, though the fact that this has happened will be included in any trace logging.

Note that abandoning the function may require that it is completed in some way for reconciliation / recovery reasons. For example failure to send a Banking [R1] message requires the banking transaction to be completed for zero value, a Reversal (ie [C0]) generated and sent to the BAL and for the failed transaction to be added into the basket. It is only at this point that it is possible for the User to retry the Banking Transaction. However for a failure to obtain data from the BAL (eg a request to obtain data for a Report), then no tidying up is required and it is OK to just abort the function.

7.20.4.2 Inconsistent Reference Data

The strategy is to make sure the counter does not fail in a catastrophic way. It should report that there is a problem with reference data and return to the menu. Where possible the problem with the reference data should be included in the error message (not for the benefit of counter staff but of support/testing staff who will most likely see these errors in using the counter to test the reference data)

7.20.4.3 Unexpected Response from the BAL

Normally this is handled in the same way as an unexpected response from any other component (see below), but if the unexpected response is rejecting a settlement operation then we need to manage that in a special way in order to prevent the user being left with an unusable system (i.e. a basket he cant settle) and to ensure any recoverable transactions are properly dealt with.

Thus if the settlement fails we do the following:

If the basket contained no recoverable transactions – tell the user and discard the basket

Otherwise tell the user and force a logoff.

7.20.4.4 Unexpected Responses from other components and Logic Errors

The approach taken is as follows:

- The fact that this unexpected error has occurred should be logged as an error to any local diagnostic logs with as much information as possible to diagnose the problem.
- An Error Event should be recorded to the system Event Log (ie NT Event Log). This will ensure that the support community are made aware of the issue and can investigate if necessary (for example by examining the local diagnostic logs for more details of the problem). We protect against "event storms" by ensuring that we do not generate multiple duplicate events from a single counter.
    - The user is informed that an Error has occurred and that the Function is to be abandoned. Again, abandoning the function may require that it is completed in some way for reconciliation / recovery reasons.
    - The running environment is tidied up as much as possible. In particular any objects that are thought to be "faulty" are destroyed so that they are re-initialised if the user attempts to retry the function.
    - The User is then free to carry on with some other function. As far as the UI is concerned, then after the error has been acknowledged by the user, the system will return to the last known menu (or failing all else the top level menu).
    - Note – the user may or may not be advised to phone the Horizon System Help Desk – but that is taken care of by POL wording the messages as appropriate.

7.20.5 Background Errors

A further category of errors are those that occur in "background processes", which may be running even if there is no user present. These Background Processes can be split into two categories:

- Those that run as part of CBA initialisation

  An example of this is Rates Board or PIN Pad initialisation, which both need access to the Counter Persistence Data.

  Failure of these functions to access Counter Persistence Data will result in the function failing (and taking appropriate default actions) and relying on the Log On function retrying the function (by which time any comms errors – which are the most likely failure conditions - must have been addressed).
- Those that are scheduled to run periodically by the Counter

  An example of this is the regular background POL to check for new Reference Data or Memos. NB such background polling takes place whether a User is Logged On or not.

Failure of these functions to access the BAL, will result in the function failing (and taking appropriate default actions) and relying on retrying the function at the next scheduled time.

The general principles described above still apply, however in this case the user is not aware of these processes and so no user interaction takes place.

7.21 ETU Reversals

This section explains how ETU Reversals (or more accurately Refunds) operate on HNG-X.

Note that an ETU Reversal is a specific, separate transaction and should not be confused with a Reversal that is generated as part of an ETU transaction should a failure occur. It would be more accurate to refer to them as Refunds. However the term "Reversal" is used on the counter for refunds made to customers (e.g. New Reversals and Existing Reversals).

7.21.1 Background

ETU Transactions are primarily non-reversible. However there is a requirement to be able to correct an error, for example if a £50 Top up was requested instead of a £5 top up. This makes them different from Banking where the assumption is that an incorrect deposit could be corrected by making a correcting withdrawal from the account.

However there are severe restrictions on ETU Reversals:

- They must be done within 10 minutes of the original transaction
- They cannot have internal Reversals should a failure occur
- They can be re-attempted following a failure (or otherwise) – we rely on e-pay preventing a Reversal being carried out more than once and don't attempt to police this ourselves

Note: a Refund Transaction is carried out against a separate Refund Product and not as a negative transaction against the original product. The relationship between these products is defined in the FT Operations Reference Data that is provided by POL as Type A Ref Data and can be derived from the PAN used to trigger the ETU transaction. The fact that a separate product is used means that we cannot use Existing Reversals to carry out ETU Reversals.

7.21.2 ETU Reversal Operation

ETU Reversals are handled in HNGX as follows:

- If a mistake has been made and an ETU Reversal is required, then it needs to be triggered from a separate session using the New Reversals menus

  HNG-X doesn't support the Session Swap functionality that was available on Horizon

- An ETU Reversal could be triggered from a separate terminal before the original session has been settled.
- The user is requested to enter the Horizon Txn Id from the original ETU Receipt (the first part is pre-filled as is done on Horizon to reduce the amount of typing required)
- This Horizon Txn Id is then sent to the ETU agent as a [T1] message to:
  - Confirm that such a ETU transaction actually took place (ie the Horizon Txn Id has been typed correctly)
  - Obtain details about the original transaction (such as the PAN) needed for a Reversal

Note that a [T1] message requires the receipt_date. It should be sufficient to use the current local date. This means that an ETU reversal may fail if midnight has occurred between the Request and the Reversal.

- When the [T2] is received, the data in the [T2] can be used to generate an [R1] for the ETU Reversal and this can then be processed in the normal way resulting in a [C1] being placed in the basket

Note that for an ETU Reversal a [C0] shouldn't be generated if the transaction fails. Following a failure the clerk should be advised that a failure has occurred and that no refund should be given, but they might want to try again.

7.22 Interfacing Counter with CNIM

7.22.1 Introduction

There is a requirement for the Counter to be supplied with Network status information, for example the Status Bar must show whether the Counter is on or off-line when no user is logged on.

This section describes the Counter components that have been implemented to meet this requirement.

To achieve these aims the counter application interacts with The CNIM component (on the counter) – see DEV/INF/LLD/0113 for details.

7.22.2 Components interested in CNIM Status

Counter Stats – see section 7.22.6

Banking – see section 7.22.5

Status Bar – see section 7.22.3

7.22.3 Status Bar

The following Business Components will call the CNIM interface:

The Counter StatusBarBLO – this calls the OnlineStatusBLO which requires knowing if the network is on or offline. The status will indicate 'Offline' if CNIM indicates permanently offline (Connection Status = '2'), otherwise 'Online' (even if CNIM is down).

7.22.4 CNIM Interface Design

The following classes are implemented:

- CnimStatusBLO – uses CnimStatusBDO and CnimStatusBS
- CnimStatusBDO
- CnimStatusBS
- CnimInterfaceImpl

7.22.4.1 CnimStatusBLO

Components interested in Cnim Status changes register, using the EventHubObserver framework, as listeners to Cnim Status changes on the **CnimStatusBLO**. The CnimStatusBLO only notifies its listeners when its relevant status value changes, or if the Cnim Connection Type Changes.

Callers can call the CnimSstatusBLO directly, using these methods:

isCnimOnline()	returns a Boolean indicating if Cnim reports the Network as online
getCnimStatusBDO	Returns a CnimStatusBDO object,

7.22.4.2 The Cnim Status Business Service

The CnimStatusBLO itself registers with the **CnimStatusBS** Business Service, which calls back when the CnimStatus changes

.The CnimStatusBS is a singleton instance (it implements the ISoleInstance) and is configured, with the following properties:

Name	Value	Comment
cnimHost	DNS host name of Cnim XML Interface	
cnimPort	Port of Cnim Interface	8514

The CnimBS is responsible for instantiating the **CnimInterfaceImp**l component. It is a singleton (it implements ISoleInstance) to ensure that only one CnimInterface is present per Counter.

7.22.4.3 The Cnim Status Interface Implementation

The CnimStatusInterfaceImpl class is responsible for communicating directly with the CnimXml interface.

CnimStatusInterfaceImpl does the following:

- Opens a socket to the cnimHost and cnimPort (defined in the BSSpecification.xml);
- Starts the CnimRunnable thread which does the following:
    - Notifies the CnimStatusBS if the Cnim Interface is down;
    - If not down, sends a <GetCNIMStatus> request, including a Timeout attribute (set to -1 for infinite blocking) and blocks waiting for a GetCnimStatusResponse;
    - On receipt of the response it builds the Comsumable CnimStatusDTO object (if the response is different to the previous response), and calls the QueueManager produce method, sending the DTO as data;
    - If Cnim is down (it may have gone down during the GetCnimStatusResponse process), tries to create a connection with Cnim.

7.22.5 Banking

CNIM provides a simple service to Online Banking, including Card Payment and e-Topup. In the case where an R1 has been sent but an A3 has not been received within the timeout period, the Banking framework asks CNIM for further information. This information consists of:

- A Response Code to use as the Banking transaction outcome code. This may indicate the connection has Failed Once, is Temporarily Unavailable, or is Permanently Unavailable, depending on how log CNIM has been disconnected from the WAN.
- A text field to be displayed to the Clerk indicating how long to wait before retrying an online transaction, OR, an error code to report to the HelpDesk.

CNIM maintains this data in the Counter CnimStatusBDO, and Banking is able to query the BDO directly. The status within the BDO is updated whenever the "CNIM Service" state changes. This is achieved by the Counter making blocking calls to the CNIM XML Service.

The Banking Component requires a response code translated from the Cnim Connection Status code as shown in the following table:

Cnim Connection Status	**Banking Response Code**	**Meaning**
4	45	Failed once
5	46	Temporary failure
6, Silver Connection Type*	47	Permanent Failure, Silver service
6, Non-Silver	48	Permanent Failure, Non-Silver
Any other value	43	Timeout, Comms OK

*'Silver' service is calculated as (ConnectionType & 0x080000 > 0)

7.22.6 Counter Stats

CNIM is also used to provide counter stats information. See DES/APP/HLD/0043 and DES/APP/HLD/0110.

7.23 Other Components

This section covers a series of topics that were described in earlier versions of this document but have since been removed. It is included for completeness.

7.23.1 Action Controller Mechanism

This is now covered in *HNG-X User Interface Components High Level Design* (DES/APP/HLD/0141).

7.23.2 Action Window, CrossUseCaseButtons, Infobar and StatusBar

This is now covered in *HNG-X User Interface Components High Level Design* (DES/APP/HLD/0141).

7.23.3 Baskets and Receipts

The design of the basket and settlement is described in document HNG-X HLD - Settlement Functions (DES/APP/HLD/0123).

7.23.4 Dialog Design

This is now covered in *HNG-X User Interface Components High Level Design* (DES/APP/HLD/0141) (including displaying messages).

7.23.5 Exception Handling

This is all covered in *Exceptions and Logging Framework High Level Design* (DES/APP/HLD/0035).

7.23.6 Generic ScreenUIA Construct

This no longer exists

7.23.7 Keyboard Input and Focus Management

This is now covered in *HNG-X User Interface Component*

7.23.8 Peripherals Management

This is all covered in *Peripherals Subsystem High Level Design* (DES/APP/HLD/0038).

7.23.9 Polling

Polling is described in *Reference Data Subsystem High Level Design* (DES/APP/HLD/0045).

7.23.10 Printing

Printing is described in *HNG-X Counter Printing* (DES/APP/HLD/0137).

7.23.11 Reference Data Manager

This is defined in *Reference Data Subsystem High Level Design* (DES/APP/HLD/0045).

7.23.12 System Shutdown

See section 7.11.4.7

7.23.13 Workspace Constructs

See *HNG-X User Interface Components High Level Design* (DES/APP/HLD/0141).

7.23.14 Pick Lists

Picklists are all handled by the ResolveProduct function (AS95) which is described in *UCR Document for Retail and Stock Sales Use Case Barrel High Level Design* (DES/APP/HLD/0067).

7.24 Process Engine

The Process Engine provides the facility for the counter to interpret and execute business scripts. The scripts are defined in a Process Definition Language (PDL). PDL was developed for the HNG-X counter by Fujitsu.

As PDL is an interpreted language PDL scripts are delivered to the counters via the reference data delivery method.

Details of the process engine can be found in DES/APP/HLD/0076.

7.25 Controlling Whether a Feature Is Enabled (AppControl)

It is necessary to be able to control whether a feature of the counter application is enabled both across the whole estate and selectively by post office branch or group of branches. This is to enable features to be both turned on and off over the life time of the system and also to manage a gradual introduction of a new feature across the estate. A standard mechanism has been defined for this which is called Application Control (or AppControl for short).

In outline, the code that implements any feature of the counter which it must be possible to switch on and off whether estate wide or more selectively must be wrapped in a conditional statement which tests a reference data object specific to the feature.

The reference data object ialways has RDObject@Type="AppControl" and RDObject@Key set to a value specific to the feature to be controlled. Reference data objects of type AppControl contain the following elements and attributes which are used to define whether the counter feature is enabled or not:

- just one element: RDObject.AppControlData.FeatureOn which is of type POBoolean (i.e. true or false) and
- the standard attribute of all ref data objects of RDObject@State of type RDObjectStateType (i.e. enabled or disabled)

(See DES/GEN/SPE/0003 for a full specification of the type).

For example For CP0391, Policing MoPs, the AppControl Reference Data is:

```
<RDObject Type="AppControl" Key="CP0391.PolicingMoPs" EffDate="2010-04-26"
Action="S" State="E" Revision="App_Control.xls">

      <RData>

            <AppControlData>

                  <!-- Policing Mops - Enabled by 31608 -->

                  <FeatureOn>True</FeatureOn>
            </AppControlData
      </RData>
</RDObject>
```

The conditional test is always to be implemented in Counter Application code such that when the AppControl object does not exist or is disabled (for the counter on which the application is running), the feature is enabled. In other words, the feature is on by default when the reference data object is absent or disabled. When the object is present and enabled the FeatureOn element is also tested. The following truth table defines when the counter feature is enabled for all permutations of the AppControl's controlling element and attribute:

AppControlData@State Value	AppControlData.FeatureOn Value	Feature Enabled/Disabled
D(isabled)	True	Enabled
D(isabled)	False	Enabled
E(nabled)	True	Enabled
E(nabled)	False	Disabled

A common base class called AppControlControllerBLO has been implemented from which a feature specific class must be derived for each feature to be switchable.

7.1.1 Enabling and Disabling The AppControl Object

The AppControl Object is implemented by the Customer Support Reference Data (CS RDT) team. The team may also omit or remove the object if the feature is to be permanently enabled across the whole estate.

If the a feature is to be selectively enabled or disabled across the estate, the CS team will use various other reference data objects including objects of type RuleObject to modify both the state of the AppControl object and its FeatureOn element. Details of how this is done are beyond the scope of this design document.

When reference data is used to manage the gradual roll out of a new feature, it is called SoftLaunch.

CS use a standard naming scheme for the AppControl keys. The key names are of the form:

"CP*nnnn*.*FeatureName*.Off"

where *nnnn* is the number of the CP and *FeatureName* names the feature to be switchable.

The *FeatureName* part of the name contain a number of dot separate names to specify a hierary of sub-features. i.e.

"CP*nnnn*.*FeatureName*.*Subfeature*.Off"

8 Non-Functional Design

This section provides a summary of non-functional design. Each of the topic HLDS (see section 2.3) provides more detail of non-functional aspects of specific areas of the counter.

8.1 Recovery and Resilience

The counter has a specific recovery subcomponent. This is described in document DES/APP/HLD/0083.

Resilience is built into the design by virtue of the fact that timeouts force a logout, errors and exceptions are handled at a common point to enable the counter to continue on the current menu wherever possible or to continue with a new login (see DES/APP/HLD/0035 and section 7.11 for details). In addition the counter application is restarted automatically by the counter infrastructure service every night or upon failure (see DES/APP/HLD/0057).

8.2 Availability

The counter application is expected to be available for 100% of the time (hardware and network permitting) except for when it is being recycled by the counter infrastructure service every night. When a user logs out the counter is instantly available for a new login (see section 7.11). The counter incorporates interfaces to Cnim (see section7.22) to enable the user to be informed about network availability.

8.3 Security

One of the design objectives of HNG-X is to minimise the amount of security and crypto that exists on the counter. This objective has been met, and the counters involvement in security is limited to:

- Logging in – use of passwords (DES/APP/HLD/0060 for details)
- Use of SSL, VPN and local public/private keys for network signing/encryption. (See DES/APP/HLD/0043 for details).
- Secure hashing of PANs for PCI conformance

In addition the counter is “locked down” such that the counter application is the only application that can run on the counter platform, and it does not provide access to any operating system functionality (e.g. windows explorer) other than those functions required by the requirements.

8.4 Performance and Capacity

Each counter position is dedicated to a single user, so little opportunity for parallelism exists, and thus little is provided (other than background activities such as polling or ref data indexing – which takes place in a lower priority thread so as to minimise impact on user performance if it happens while a user is logged in, although it usually takes place overnight). .

The startup time of the counter is considerably improved over Horizon – early indications show it to be around 2 mins (instead of Horizon’s 20 mins).

Although the typical basket size is just under 2 transactions per customer session, together with a further settlement transaction, the counter allows for the extreme case of a customer session comprising up to 9999 (configurable) accounting lines.

More information about counter performance can be found in ARC/APP/ARC/0009.

8.5 Concurrency

There is no requirement for concurrency on the counter other than the counter application should be able to operate concurrently to other software on the counter. Testing has shown that it does this.

8.6 Migration

See section 4.

8.7 Supportability

The counter provides a number of supportability aides – these are documented in the Counter Application support Guide (DEV/APP/SVG/0017).

8.8 Testing and Validation

The application is designed to be testable.

It may be necessary to provide additional interfaces and capabilities specifically so that testing can be automated.

An automated "smoke" test is provided so that automated regression testing is possible.

A number of peripheral simulators are provided to support testing in various environments (e.g. printing simulation).

8.8.1 Testing During the Development Process

The counter code is subjected to the following forms of testing during the development process.

8.8.1.1 Component (Unit) Testing

Each component is tested in isolation (e.g. JUnit testing)

8.8.1.2 Regression testing (i.e. the smoke test)

An automated "smoke" test is run to check for regression when a group of changes are gathered together into a new build.

8.8.1.3 Continuous Integration

Every change in the new build is subjected to a specific a specific test during CIT before the new build is released.

8.8.1.4 Service Interface Validation Testing

A new BAL build is tested against both its matching counter build and the previous counter build to ensure backwards compatibility and allow for counters to be updated over a period of time.

FUJ00091790

8.8.2 Testing after the Development Process

Further testing of a released build is carried by system testing, performance testing and release testing of various kinds. These are not under the control of the counter design so are not documented further here.