FUJ00001368

Document Title: **Generalised API for OPS/TMS**

Document Type: Technical Design Standard

Release: N/A

Abstract: This document provides the information required to plan the development of new applications and describes in more detail the architecture set out in the OPS Architecture Specification.

It is supplied under the terms of the Codified Agreement to POCL to facilitate the procurement of applications to run on the Service Infrastructure (interfacing with OPS and TMS).

Document Status: APPROVED

Author & Dept: Patricia Morris, Jon Cruise, Janet Dore: Technical Design Authority

Contributors: Tony Hayward, John Allen, Peter Morgan

Reviewed By: ICL Pathway: John Allen, Terry Austin, John Dicks, Phil Hemingway, Alan Hodgkinson, Gareth Jenkins, Dai Jones, Duncan Macdonald, Steve Warwick, Peter Wiles

POCL: Bob Booth

Comments By:

Comments To: Document Controller & Authors

Distribution: ICL Pathway Library and Reviewers

FUJ00001368

0.0 Document Control

0.1 Document History

Version No.	Date	Reason for Issue	Associated CP/PinICL No.
0.1	15/10/99	Section 1—4 only.	
0.2	24/10/99	Section 5 added. Section 4.4.5 amended. Cash Account: Figure 4-36 added. Subsequent figures renumbered.	
0.3	28/10/99	Section 9 added.	
0.4	08/11/99	Section 6 and 7 added.	
0.5	12/11/99	Complete draft issued, incorporating comments from Alan Ward and Dai Jones and additional material from Janet Dore.	
0.6	17/11/99	Revised draft issued incorporating comments from Tony Hayward and Janet Dore.	
0.7	22/11/99	Revised draft issued incorporating comments on draft 0.6.	
0.8	13/12/99	Revised draft issued incorporating POCL's comments on draft 0.7: New section 2 added.	
0.9	15/12/99	Revised draft issued incorporating internal review comments on draft 0.8.	
0.10	27/01/00	Revised draft issued incorporating POCL review comments on draft 0.9. Section 2 and 3 swapped. Sections 6 and 7 swapped. Sections 8 and 9 swapped. For internal review.	
0.11	28/01/00	Revised draft issued incorporating internal review comments on draft 0.10.	
0.12	11/02/00	Revised draft issued incorporating review comments on draft 0.11.	
0.12a	15/02/00	Revised draft issued incorporating POCL's review comments on draft 0.12.	
1.0	18/02/00	Approved version.	

FUJ00001368

0.2 Approval Authorities

Name	Position	Signature	Date
T. Austin	Development Director		
J. Dicks	Customer Requirements Director		
R. Booth	POCL		

0.3 Associated Documents

Reference	Version	Date	Title	Source
CR/SPE/007	0.3	07/09/99	Development of Manual Describing Use of OPS, TMS and EPOSS APIs within ICL Pathway ref	ICL Pathway
TD/ARC/030	0.4	12/11/99	OPS Architecture Specification	ICL Pathway
TD/ARC/029	0.4	12/11/99	TMS Architecture Specification	ICL Pathway
BP/FSP/004	4.0	05/03/99	EPOSS Functional Description	ICL Pathway
CS/IFS/001	2.2	12/08/99	Reference Data Change Catalogue	ICL Pathway
CS/PRO/095	0.4	21/01/00	EPOSS PPD	ICL Pathway
RS/POL/003	3.0	22/12/98	Access Control Policy	ICL Pathway
SD/STD/001	2.1	12/12/99	OPS Style Guide	ICL Pathway
SD/DES/005	7.1	13/12/99	OPS Reports and Receipts	ICL Pathway
SD/SPE/016	9.0	21/01/00	OPS Menu Hierarchy	ICL Pathway
TI/IFS/001	5.7	05/07/99	Pathway to TIP Application Interface Specification	POCL
RDP/AIS/001	4.1	25/6/99	AIS Reference Data to Pathway	POCL
RDP/AIS/009	1.0a	12/2/99	AIS Reference Data to Pathway Non-Automated Type B Data	POCL
	2.3	30/06/99	Riposte Broker API Reference Manual	Escher
	1.0		Riposte Desktop Session Management	Escher
	1.0	29/06/99	Design Studio Receipt Layout Editor User's Guide	Escher
	1.3	03/02/99	Riposte Validation Object Developer's Guide	Escher

FUJ00001368

	1.1	03/02/99	Riposte Validation Broker API Reference Manual	Escher
	1.0	24/11/98	Riposte Peripheral Broker API Reference Manual	Escher
	1.2	30/06/99	Riposte Retail Broker API Reference Manual	Escher

0.4 Abbreviations/Definitions

Abbreviation	Definition
ACL	Access Control List
AIS	Application Interface Specification
AP	Automated Payments
API	Application Programming Interface
APS	Automated Payment Service: counter application supported by Horizon.
BP	Balance Period
C	A UNIX-derived programming language
C++	Object oriented version of **C**
CAP	Cash Account Period
CM	Configuration Management
CRC	Cyclic Redundancy Check
Decimal pounds	For amounts that are whole pounds, no pence field (decimal point, followed by two digits) is present. For amounts that include pence, the pence field is present.
Design Studio	Escher product that supports application design and development. (It provides a set of templates and add-ins for common receipt layout types that can be adapted to the needs of a particular application.)
DLL	Dynamic Link Library
DSS	Department of Social Security; a Client of POCL
EFTPoS	Electronic Funds Transfer at Point of Sale
EoD	End of Day
EPOS	Electronic Point Of Sale

FUJ00001368

EPOSS	Electronic Point Of Sale Service: counter application supported by Horizon.
FAD	Financial Accounting Division
FTF	File Transfer Facility
FTMS	File Transfer Managed Service
GMT	Greenwich Mean Time (see UTC)
HCI	Human Computer Interface
HTML	Hypertext Mark-up Language: standard used for production of Internet type pages.
KMS	Key Management Service
Mon	First three letters of month. Used in date format in messages: 12-Jan-2000, for example.
MoP	Method of Payment
NFS	Networked File System
NIFTP	Network Independent File Transfer Protocol
NMS	Network Management Server platform used to perform network management functions.
NTFS	Windows NT File System
OBCS	Order Book Control Service; counter application supported by Horizon, which supports a similarly named DSS application.
OCX	OLE Custom Control
OPS	Office Platform Service. The provision and support of the hardware and software at Outlets including the Desktop environment of the Horizon system.
PBS	Product Breakdown Structure
PIT	Product Integration Testing
POCL	Post Office Counters Ltd
PVCS	Configuration Management tool
RDDS	Reference Data Distribution Service: application which feeds ‘soft’ data to counter applications.
RDMC	Reference Data Management Centre: application that accepts data from POCL and converts it to a format suitable for distribution.
RDMS	Reference Data Management Service; encapsulates RDDS and RDMC.
RDS	Reference Data System: POCL’s application for Reference Data

FUJ00001368

	Management.
RIPOSTE	Retail Integrated Point Of Sale system in a Transaction Environment: product from Escher that provides both the infrastructure and the Desktop environment of the Horizon system. The definitions in this manual refer to version 6.0 onwards.
RMS	Riposte Message Server: message storage and replication mechanism of Riposte.
RPC	Remote Procedure Call
SLA	Service Level Agreement
SQL	Structured Query Language; language commonly used to access Relational Database systems.
TIP	Transaction Information Processing: POCL application that handles transaction data returned from Horizon.
TMS	Transaction Management Service. The hardware and software required for the replication, transmission and management of transactions committed to the Horizon Riposte Message Store and Pathway Data Centres, or vice versa.
TPS	Transaction Processing System: application that collects transaction information and returns it to TIP.
UTC	Co-ordinated Universal Time (same as Greenwich Mean Time)
VPN	Virtual Private Network

0.5 Changes in this Version

Version	Changes
1.0	Includes changes arising from POCL's review of V0.12a.

0.6 Changes Expected

Changes
Changes required as a result of issues raised on V1.0.

FUJ00001368

0.7 Table of Contents

FUJ00001368

FUJ00001368

FUJ00001368

FUJ00001368

FUJ00001368

1 Introduction

1.1 Purpose

This document is designed to facilitate application development in the ICL Pathway environment. It is intended to augment the documentation supplied by Escher by setting the Horizon implementation into context. Readers require access to Escher's documentation set when developing applications.

An overview of the architecture of the ICL Pathway Office Platform Service (OPS) and the Transaction Management Service (TMS) is provided in the related documents *OPS Architecture Specification* and *TMS Architecture Specification.*

This document provides additional information to application developers about the architecture and facilities of OPS, TMS and Electronic Point Of Sale Service (EPOSS), to enable them to plan the development of new applications operating in this environment.

1.2 Readership

This document is intended for application developers within ICL Pathway or elsewhere. It has been developed to enable developers to plan the development of new applications to operate within the ICL Pathway solution.

1.3 Related documents

Documents that are referred to in this document, and that should be read in conjunction with it, are as follows:

OPS Architecture Specification

This document describes the overall systems architecture and the functionality of the various sub-systems that comprise the suite of software. It covers the following topics:

- The architecture of counter applications used in the Riposte environment.
- The functionality provided by the Riposte software, including the Retail Broker interfaces, the Peripheral Broker interfaces and the Riposte OCXs (ActiveX Control Modules).
- Security and administration features provided by Riposte.
- How counter applications make use of help and training mode.

FUJ00001368

- The ICL Pathway software distribution process.

TMS Architecture Specification

This document covers the following topics:

- How Riposte 6 facilities are used across the TMS domain.
- How counter applications interface with the TMS.
- How agents interface with the TMS.
- Security domains that are relevant to the TMS.
- Resilience features in use in outlets and across the TMS domain.

Access Control Policy

This document defines the access control policy to be followed across the ICL Pathway solution.

Horizon Office Platform Service Style Guide

This document defines the user interface style to be used in counter applications.

Reference Data Change Catalogue

This document defines how changes to Reference Data are implemented, using the operational business change process.

FUJ00001368

2 Scope

This document provides a description for the application developer of the context in which the APIs are used.

Section 2.1 gives a summary of the contents of each section of this document.

Section 2.2 gives a more detailed description of the contents of each section and relates each one to the three classes of interfaces that are described here: business function interfaces, application interfaces and other sorts of interface.

2.1 Contents

This document describes the interfaces that are used in the Horizon system in various contexts. The document is organised as follows:

Section	*Contents*
Section 1	introduces the document.
Section 2	defines the context of document with reference to associated documents.
Section 3	is an overview of the system's interfaces.
Section 4	describes how OPS interfaces are used in the business context: EPOSS, settlement, stock unit management, reporting and balancing.
Section 5	describes how applications utilise Application Program Interfaces (APIs). It covers architecture, Peripheral Broker interfaces, Retail Broker interfaces, Desktop interfaces and Riposte 6 functions.
Section 6	describes other functions provided by Riposte 6: administration, security, availability, usability, performance and resilience.
Section 7	describes agent interfaces: General Acknowledgement Agent, bulk agents, interactive agents and enquiry agents.
Section 8	describes system management procedures: POCL Reference Data, event reporting, software packaging, system testing, integration and implementation.
Section 9	describes naming standards: application components, messages, persistent objects, events and attributes.
Appendix A	describes the interfaces used in SmartMan, the interface that manages interactions with smart cards.
Appendix B	describes the interfaces used in applications that have a requirement to use cryptography and Key Management software.

FUJ00001368

Appendix C	describes the various system management interfaces that have to be supported, and the facilities that have to be implemented, during the various stages in providing a new application as part of the Horizon environment. The interfaces and the facilities must be considered when designing and developing an application.

2.2 Interface types

The context in which the APIs are used are described in the following terms:

- Business functions supported by EPOSS
- Application functions supported by OPS and TMS
- Other application interfaces that are required in the OPS/TMS environment

The document identifies relevant Escher-supplied APIs. The SmartMan API , which has been created by ICL Pathway, is described in Appendix A.

2.2.1 Business functions supported by EPOSS

Section 4 describes EPOSS in the business context. The following concepts are described:

- Electronic Point of Sale Service (EPOSS)

  The sale of a product, its relationship to POCL Reference Data and the data structures used.

- Settlement

  The concepts used in the settlement of a customer session, session swapping and transfer, keyboard inactivity timeout and its effects, and how settlement data is sent to Clients.

- Stock unit management

  Concepts, shared and individual stock units.

- Reporting

  Reporting is covered, with reference to the *OPS Reports and Receipts* document.

- Stock Unit Balancing

  Achieving stock unit balance, relationship to the Cash Account.

2.2.2 Application functions supported by OPS and TMS

Section 5 refers to *OPS Architecture Specification* and describes the following interfaces:

- Peripheral Broker

FUJ00001368

The interface provided to support the use of input and output devices.

- Retail Broker

  The interfaces that are used to add the sale of a product as a transaction to the stack.

- Desktop interfaces

  Standard OCXs available to the application developer; OCXs controlled by the system.

- Riposte 6 functions

  Interfaces used to handle messages and persistent objects.

Section 6 describes the use of the OPS and TMS administrative and security functions provided by Riposte 6:

- Administration

  System-supplied administration and configuration functions.

- Security

  Security functions relevant to application development; brief description of other security functions.

- Availability

  Service Level Agreements; the impact of end of day processing and of disconnected counters.

- Usability

  The interface to the clerk; *Horizon Office Platform Service Style Guide* rules.

- Performance

  Minimising the impact of new applications.

- Resilience

  Resilience provided by Riposte 6; applications' handling of hardware, software and communications faults.

FUJ00001368

Section 7 describes the following agent interfaces and any constraints associated with them:

- General Acknowledgement Agent

  Interfaces used by the General Acknowledgement Agent.

- Bulk Agents

  Interfaces used by bulk inbound and outbound data transfer agents.

- Interactive Agents

  Interfaces used by interactive inbound and outbound agents.

- Enquiry Agents

  Interfaces used by enquiry agents.

2.2.3 Other application interfaces in the OPS/TMS environment

Section 8 includes the following aspects of systems management:

- Reference Data

  How POCL Reference Data is accessed; the temporal nature of Reference Data; the process used to maintain it.

- Event reporting

  Application interfaces to be used for event reporting; the reporting of exception conditions.

- Software packaging

  Software handover to ICL Pathway for system integration testing; documentation needed to support handovers; overview of the system and integration process; implementation of a new application.

- Diagnostics

  The implications of recording diagnostic information in a distributed and remote estate

Section 9 describes the basic rules of naming and the naming standards for:

- Applications and their components
- Worksets
- Configuration items
- Transient messages
- Persistent objects
- Attributes
- Events

FUJ00001368

Appendices

The appendices cover the following topics:

- SmartMan Interfaces

  This appendix gives details of the interfaces needed to support the use of smart cards.

- Cryptography and Key Management

  This appendix gives details of the functions for key management in the ICL Pathway environment, including the distribution of keys to post offices.

- System Management

  This appendix describes the system management interfaces and the facilities that must be considered when designing and developing an application. Support for the interfaces must be incorporated into a new application so that it can be configured as part of the Horizon system. The necessary facilities must also be provided by the application for handling by the integration, testing, release, and support activities within ICL Pathway and Horizon.

FUJ00001368

2.3 Road map

Section 3 is an overview of the system's interfaces. The remaining sections of this document contain detail on the topics shown in Figure 2-1.

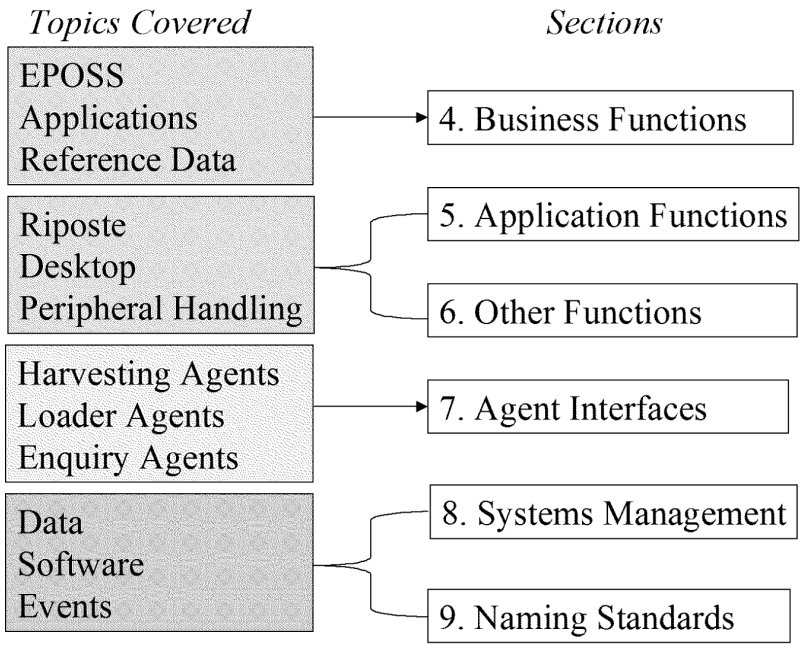


Figure 2-1 Document road map

FUJ00001368

3 Overview

This document forms part of the set that defines the environment that supports counter applications within the Horizon system. The documents involved are shown in Figure 3-1 together with an indication of their scope:

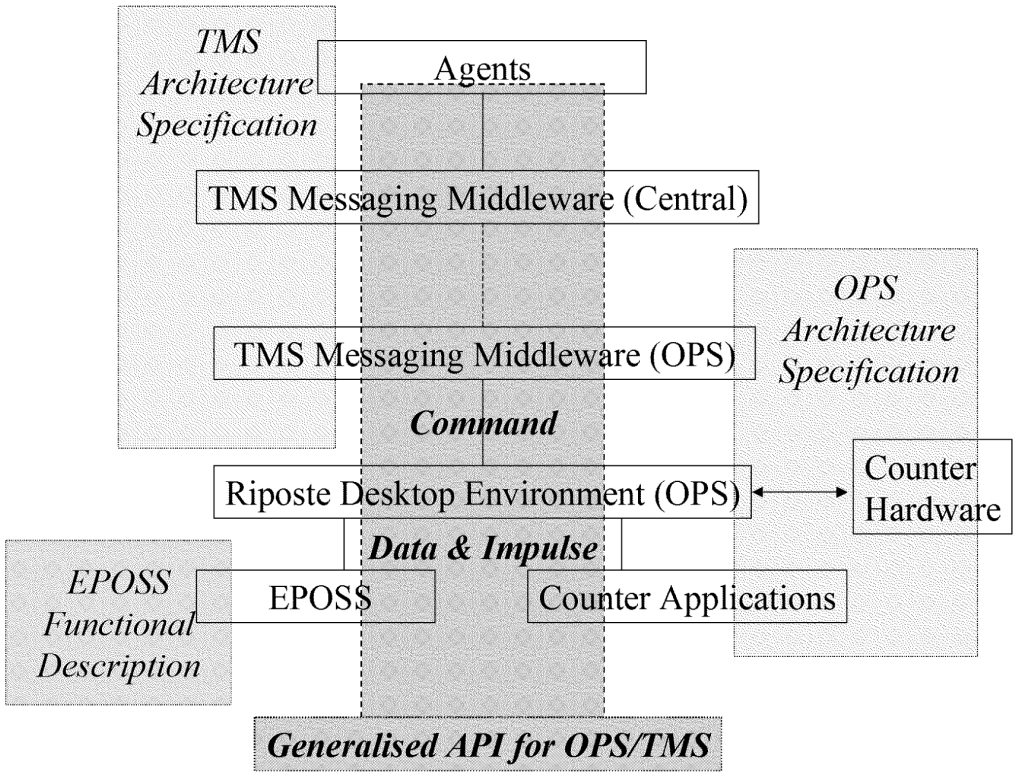


Figure 3-1 Document set

The scope of both the architecture documents is given in section 1. *EPOSS Functional Description* defines the business processes supported by EPOSS counter applications.

All applications and hence services within the OPS are supported by three types of interface and it is these interfaces that are defined in this document:

1. ***Command*** where this provides the interfaces between the application and its Riposte-based environment. These interfaces are provided by the Riposte messaging software and its associated Desktop environment and as such are only identified in this document. The only exceptions to the rule that these interfaces are provided via Riposte, are the driver extension provided by ICL Pathway to support smart card

FUJ00001368

interactions and the use of standard NT print spooling capability for the back office A4 printer. Details of the NT interfaces are not given here, as the default printer is used, but the smart card interfaces are included in this document.

2. ***Data*** where this defines the business rules and the result of counter transactions. All interfaces between applications are data driven. Each application needs to provide data to support EPOSS functionality and the models of this data, and the data that contains the rules needed by all applications, are defined within this document.
3. ***Impulse*** where this determines when and under what conditions the application is invoked. These are generated by use of peripherals or by use of the buttons that define common functions within the Desktop environment. The data involved is identified within this document.

3.1 Data driven

All business applications that have been developed within the OPS domain, such as APS, are data driven. Those that are developed in the future also need be data driven. The Riposte Message Server (RMS) interface within the TMS Messaging Middleware provides the secure data interface for all applications and for the Desktop components themselves. In this environment, application interfaces are data interfaces supported by the command sets provided as part of the Riposte Message Server and other Desktop components. Defining the rules for each application in data has the following advantages:

- It removes the need to 'hard code' business rules into the application making them easier to change when business circumstances change. Strict control has to be exercised however over such data interfaces in order to establish the impact of migrating from one version to another.
- It makes rules that apply to more than one application readily available and consistent across all applications that use them.
- It enables a consistent approach to be adopted when defining the way messages are displayed to clerks during a customer session.

3.1.1 Data ownership

The data that passes across each interface is 'owned' by a particular application and service. Some data is created by one application to be shared with others. Any data that is application-specific, provided it conforms to the system wide rules given later in section 4, can be defined in the most appropriate way to support the processing undertaken by that application. Any data that is shared must be formally defined and these definitions are given later in this document.

FUJ00001368

3.1.2 Data interface types

The data interfaces used by each application within the OPS contain four types of data as illustrated in Figure 3-2:

1. ***Business Reference Data***

   Business Reference Data defines business rules. This is supplied by POCL across the Reference Data Interface as Type A or B:

   - ***Type A***

     Type A data can be applied automatically without any manual interventions. Product price is an example of this type of data, or the definition of a token such as a bar-coded bill.

   - ***Type B***

     Type B data requires manual intervention by ICL Pathway before it can be applied. Scales data is an example of this type of data; ICL Pathway enrich the data provided by POCL with Scale Service product numbers.

2. ***Client Reference Data***

   Client Reference Data defines the business rules that are specific to a particular Client. This Reference Data is not normally supplied via the POCL Reference Data System interface but is delivered across a client-specific interface. POCL Client Reference Data is handled directly by applications such as APS.

3. ***Implementation Reference Data***

   Implementation Reference Data defines how a facility or function is implemented, the messages to be displayed to the clerk and system variables. This is supplied by ICL Pathway as Type C or D:

   - ***Type C***

     Type C data is used to define the temporal data needed by business applications, and the menu hierarchy used to define the products and applications available on all menu screens displayed to the clerk. The definition of a product button is an example of this type of data.

   - ***Type D***

     Type D data is used to define application- or Desktop-specific data such as messages to the clerk and implementation parameters.

4. ***Transaction Data***

   Transaction Data defines the result of a transaction for both EPOSS and any Client involved, and includes, for example, the amount transacted, the product code used and Cash Account mappings.

FUJ00001368

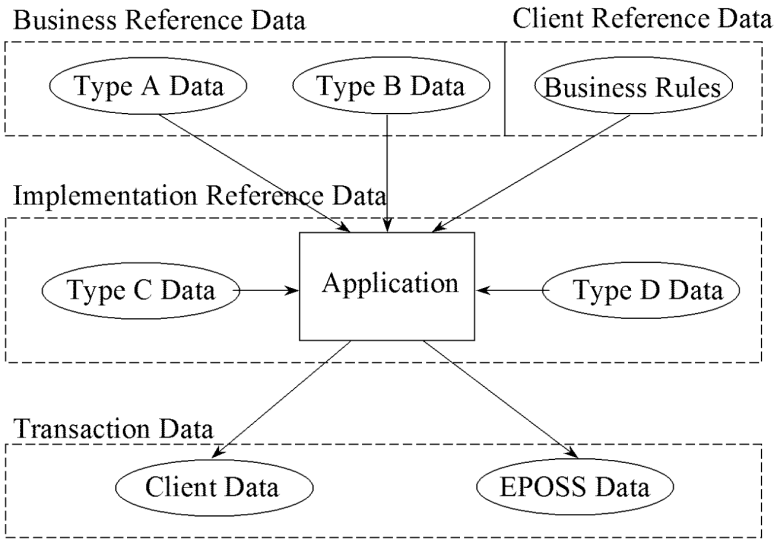


Figure 3-2 Application interface data types

3.1.3 Business and Client Reference Data

Business and Client Reference Data define the business rules that must be followed by the application. The rules come from two sources and therefore have two owners:

- Token definitions: the definitions of any bar-coded paper, magnetic cards or smart cards that must be processed by the application. This data is owned by POCL and it forms part of the overall POCL Reference Data set. This data is formally defined in the POCL Application Interface Specification (AIS) and Technology Interface Specification (TIS) appropriate to the application. This data is provided from the POCL Reference Data System to the ICL Pathway Reference Data Management Service. After conversion into attribute grammar format, the data is distributed to the appropriate outlets.
- POCL data defining the product and accounting rules to be applied. Again this data is owned by POCL and is provided via the same mechanisms as token definitions.
- Client data definitions: the definition of client-specific rules such as tariffs. This data is owned by POCL on behalf of the Client. This data is formally defined in the Application Interface Specification (AIS) for the application and is provided via a separate interface that is responsible for ensuring that it is distributed to the appropriate outlets.

FUJ00001368

3.1.4 Implementation Reference Data

Each application and the Desktop itself must have implementation level reference data defined for it that covers:

- Definition of the mappings of all menu buttons on the screen to either system or application functions. This data is defined in the *OPS Menu Hierarchy,* which is maintained via the Operational Business Change process. It is owned by the OPS service, and is not covered further in this document.
- Definition of all messages output to the clerk. This data is owned by the application to which it belongs and must be available to the writers of counter procedures.
- Definition of any receipts and reports produced by the application. Again this data is owned by the application and must conform to the definitions and layouts laid down in the Horizon Reports and Receipts document.
- Definition of system variables such as the mode or type of session in which the application is being executed. This data is owned by the OPS service.

3.1.5 Transaction data

Each application must ensure that throughout a customer session the appropriate transaction messages are recorded against the list or stack of transactions maintained for the clerk, to indicate all the products or services transacted during a customer session. This data is produced and owned by the application, but must be shared with EPOSS for both Point Of Sale and Cash Account purposes. There are occasions when additional data needs to be recorded as the transaction proceeds, such as the status information concerning smart card values, and this data can be created independently of the transaction data and at the most appropriate time. This data may or may not be sent to a Client.

3.1.6 Data Forward Compatibility

When data needs to be added or changed to cover changes or new functionality, it is important that forward compatibility is maintained and that both old and new versions of the data can be processed correctly during a migration period, by any existing or changed application code. Existing applications that have changed, or new applications, will coexist in the distributed estate during the period of migration, and data remains available until it is archived.

To achieve forward compatibility, the following rules must be followed:

- Both the changed data and new data attributes must be treated as new data.

FUJ00001368

- Processing in existing applications will ignore the new data and will not create data in the new format, so any agents that deal with the data during the migration period must be capable of processing both old and new data.
- Processing in the new or changed applications will use only the new data.
- For deleted items, the existing application will continue to create the attribute during migration, but any harvesting agent involved should ignore the attribute once the changed or new application exists in the estate. For agents that load data, the attribute will have to be provided until it is no longer required.

3.1.7 Optional and Null Data

Not all data is required in all circumstances. For example, many items that are sold in post office outlets are fixed priced, but there are some that are not, and these are defined in Reference Data with minimum and maximum permitted values. There is no need to record the minimum and maximum values for fixed price products, since they have no meaning. Data attributes such as these are therefore optional and should be omitted if they have no relevance.

In other circumstances, it may be a business requirement to record the absence of value, in which case the data value should be left null. For performance reasons null data items should be kept to a minimum. Any data items that represent true and false values should be optional items, with the values set so that data is included for the exceptional case, but not for the normal case.

FUJ00001368

3.2 Command set

The command set defines the call interfaces to the Riposte software components that provide the environment in which each application operates. The commands:

1. Provide access to the various kinds of Reference Data through the Riposte Message Server interfaces.
2. Allow transaction and Client data to be created via both the Message Server and Retail Broker interfaces.
3. Support interactions with both peripherals and the Desktop controls via the Peripheral Server and Retail Broker respectively.

The Riposte software provides these interfaces except in two cases. For smart cards, ICL Pathway has developed a generic solution and in the case of A4 printing, standard NT facilities are used.

3.2.1 Application subset

The Riposte environment provides both the underlying Message Server capability and the components that make up the Desktop itself. Any Message Server command that impacts the OPS service is not to be used in application code. These are identified in section 5.2.3 (Peripheral Broker interfaces) and 5.5.3 (Message-handling interfaces). Similarly, any Desktop commands that could impact a whole customer session by, for example, cancelling the whole session, should not normally be used by applications, unless the whole session is controlled by the application. These commands are identified in section 5.3.1 (Retail Broker interfaces).

3.2.2 Data handling

The interface to the Message Server via the Riposte Broker is shown below in Figure 3-3. It allows both persistent data such as the various classes of Reference Data to be read, and transient data such as Transaction Data to be written. If data is displayed to the clerk via the transaction stack, the Retail Broker interfaces are used and the messages are written to the message store at the end of the customer session. All other data is written to or read directly from the message store.

FUJ00001368

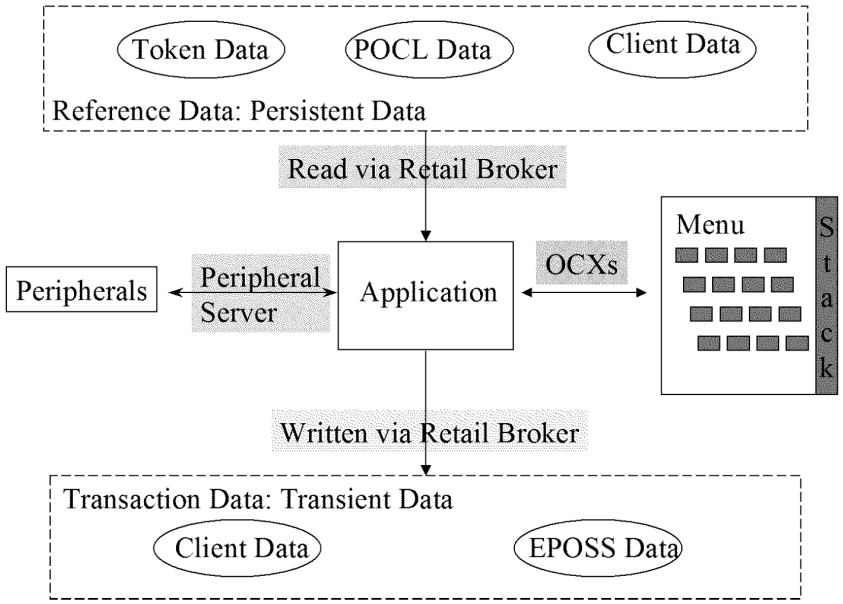


Figure 3-3 Message Server / Riposte Broker interface

3.2.3 Peripheral commands

All peripherals with the exception of the back office A4 printer are handled via Peripheral Server commands. NT interfaces are used to communicate with the A4 printer. The interfaces provided via the Peripheral Server for the Horizon system are described in later sections and cover:

- Bar-code readers
- Magnetic card stripe readers
- Smart card readers/encoders
- Scales
- Touch-sensitive screen
- Keyboard
- Tally roll printers

3.2.4 Desktop buttons

The Desktop displayed to the clerk is controlled by a number of OCXs that define the shape, content and position of all the standard controls and those controls available to the application developer. Use of a standard set ensures that the image presented enables the clerk can react in a consistent and predictable way. The Desktop estate, which comprises the areas allocated to system and menu buttons, the transaction stack,

FUJ00001368

instructions to the clerk and messages that require actions, is controlled by the Riposte software but allows the input and output of application specific messages and data.

3.3 Impulse invocation

All the applications within the OPS are invoked by impulses. An impulse is generated when peripherals are used or when specific events occur. Touching a button on the screen or a message arriving at the counter position are examples of impulses that create events. The impulses can be divided into three types, as shown in Figure 3-4:

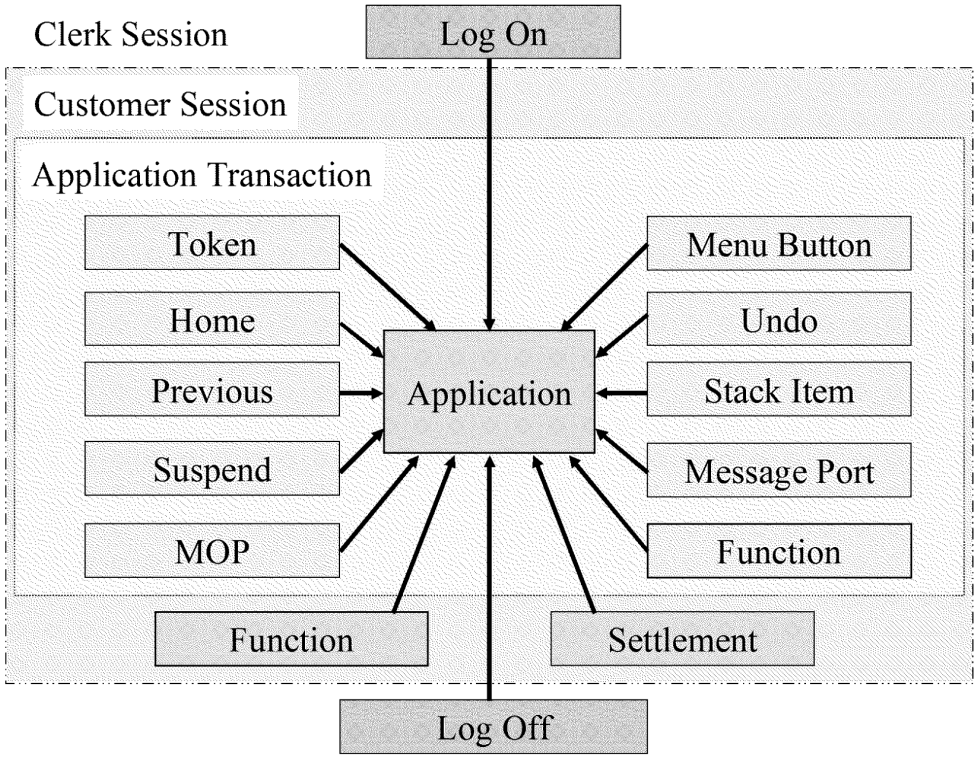


Figure 3-4 Impulse types

Different types of impulse are generated at the beginning and end of a clerk session, at the end of a customer session and during the dialogue with the clerk that results in the transaction being completed.

It is possible for an application to receive and process messages when no user is logged on; the results are displayed at an appropriate time when the user next logs on.

For a definition of sessions, see section 3.4.1.

FUJ00001368

3.3.1 Clerk session impulses

Clerk session impulses relate to the clerk logging on to and off from the desktop service. A clerk may log on and off more than once a day, or not for many days, depending on work patterns. The log-off may be initiated by the clerk, or forced by the system after a defined period of inactivity.

3.3.2 Customer session impulses

As far as the customer session is concerned there is no initial impulse, but final settlement completes the session for all applications. There are other functions such as session discounts that can apply at the session level, as well as dealing with any outstanding balances. The settlement impulse allows the application to deal with any additional processing not provided by the EPOSS application.

3.3.3 Transaction impulses

During a customer session, any number of transactions may take place using the applications available within the OPS. These transactions are normally initiated by the use of a token such as a bar-coded document, magnetic card, or smart card. If none of these is appropriate, a button on the Desktop screen when touched provides an impulse to initiate the application transaction. It is possible also for the application to be initiated by the receipt of a message that has been distributed from the centre.

Once a transaction has been initiated, it is possible that before the customer session is completed, the clerk will return to the home menu, a previous menu, or even suspend the session whilst dealing with another customer. It is also possible that the customer may change his mind and either decline to complete the transaction or change details within the transaction. If the transaction being completed needs to check details held by an external system, an enquiry is made and the application waits for the result of the enquiry to be received at a message port. All these scenarios provide impulses to allow the application to take the most appropriate action.

Once the transaction has been completed and there is an outstanding balance, the MoP (cash or cheque or vouchers, for example) must be established, together with any additional functions such as transaction discounts.

FUJ00001368

3.4 Application structures

The structure of applications within the OPS depends upon three factors:

- Applications must support the session structures inherent in the Horizon system.
- Each application must be constructed so that it can support the range of impulses it receives within the session structure.
- For each impulse type, the application must be capable of interpreting and creating the nested data structures provided by the Riposte messaging software.

3.4.1 Support of session structures

The session structure is illustrated in Figure 3-5 and reflects the following system and accounting imperatives:

- The Cash Account Period (CAP).
- The fact that the clerk may log on and off more than once during the period that the outlet is open.
- EPOSS must support POCL accounting processes, so Client sessions must be nested within an EPOSS session.
- End of Day (EoD) processing has to be co-ordinated so that the same data is sent both to TIP and to Clients; this can result in customer sessions being started and finished on different trading days.
- The clerk must be able to transfer a session from counter position to another (in most circumstances).
- The clerk must be able to suspend a session and to start another one.
- The system must be allowed to log off a user who has exceeded the system's parameter for the maximum elapsed time allowed for completing a session.

FUJ00001368

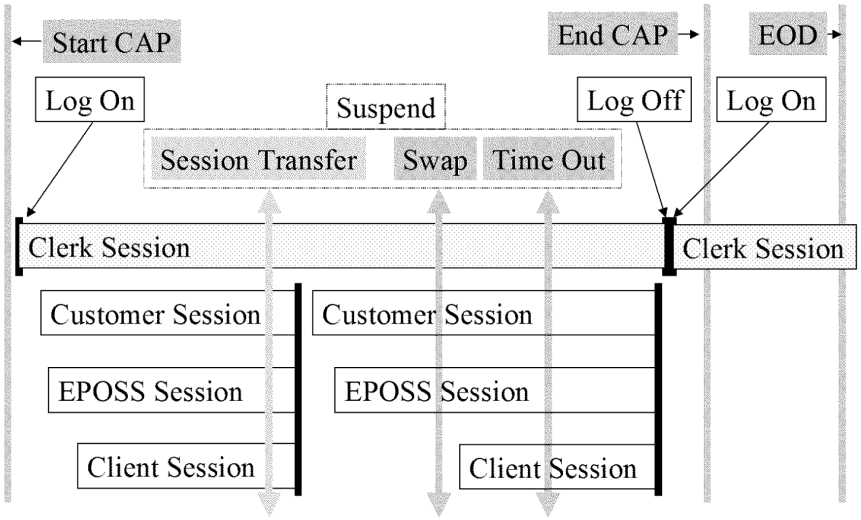


Figure 3-5 Session structures

3.4.2 Application structures

Applications are structured to support the impulse events that they receive. The generic structure to be supported is shown in Figure 3-6. In this diagram, and those that follow, if the item is optional, it is marked with an 'o'. If it can occur many times in the same structure, it is marked with an '*'.

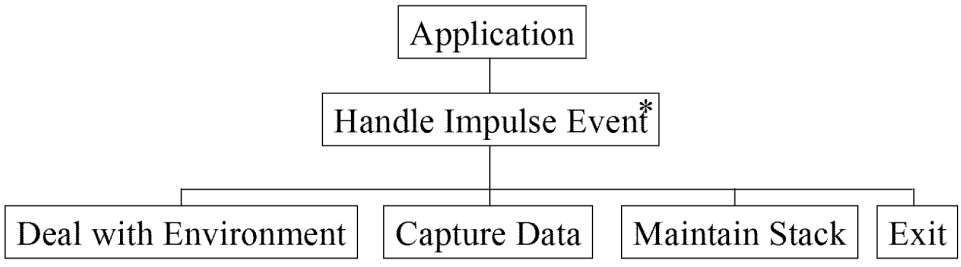


Figure 3-6 Application structures

For any impulse received, the following conditions must be met:

- The correct environment must be established.

FUJ00001368

- Any dialogue with the clerk to obtain additional data or to deal with errors or options must be correctly structured.
- Transaction data recorded as a result of the dialogue must be consistently maintained.

The structure for processing each impulse is simple; complexity is only introduced as the result of the number of different types of impulse that may be received and which must be processed.

The environment established for each application, shown in Figure 3-7, includes:

- Configuration data such as NT registry settings and Riposte variables.
- Reference Data, as described earlier, for both business and system context.
- Recovery data that supports the suspension of a session and also session transfers.

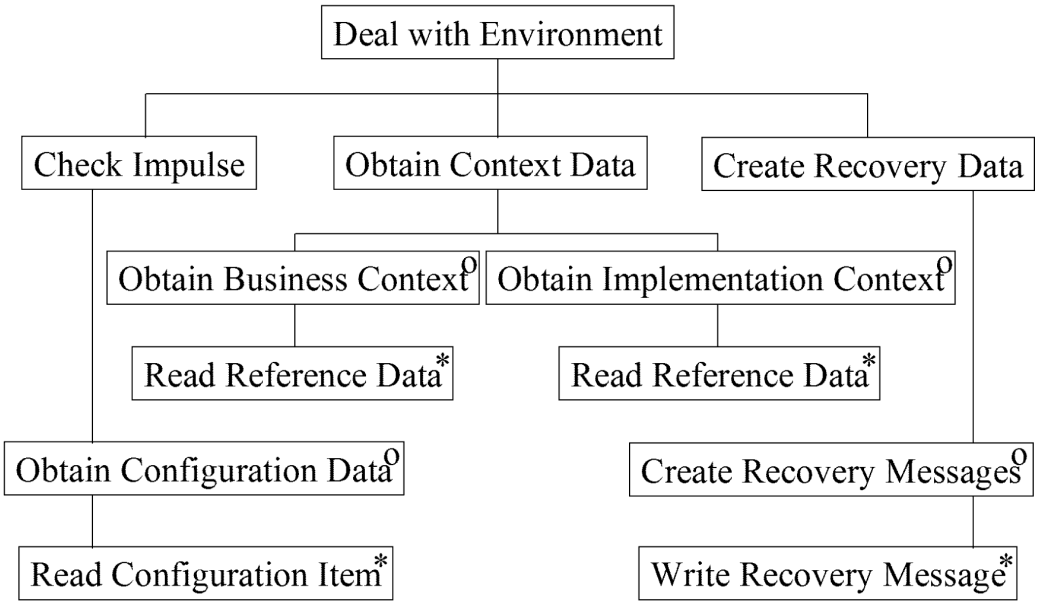


Figure 3-7 Application environment

When data is to be captured with each impulse, the dialogue with the clerk follows a script. The script allows the processing to be completed for a particular item and the next step in the dialogue to be established, as shown in Figure 3-8.

FUJ00001368

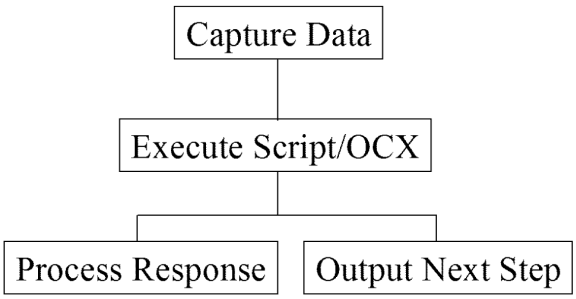


Figure 3-8 Script for data capture

Final processing involves any maintenance of the transaction stack that applies to that particular impulse. Transactions can be created, updated and cancelled as shown in Figure 3-9.

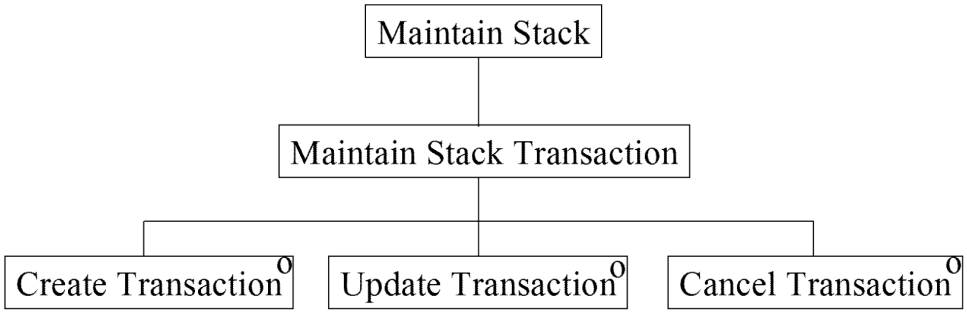


Figure 3-9 Transaction stack maintenance

FUJ00001368

3.5 Nesting of data

The Riposte message structures are de-normalised in that they allow blocks of data to be repeated and allow the nesting of data, each set of data contained within angle brackets. The kinds of structures used in Riposte messages are illustrated in Figure 3-10. All messages include data attributes provided by Riposte; those created by applications include attributes created by the Retail Broker. Application data is separately identified and when used by more than one application, must follow the conventions identified in sections 4 and 5 of this document. Applications interpret and create attribute grammar strings using Riposte-supplied functions.

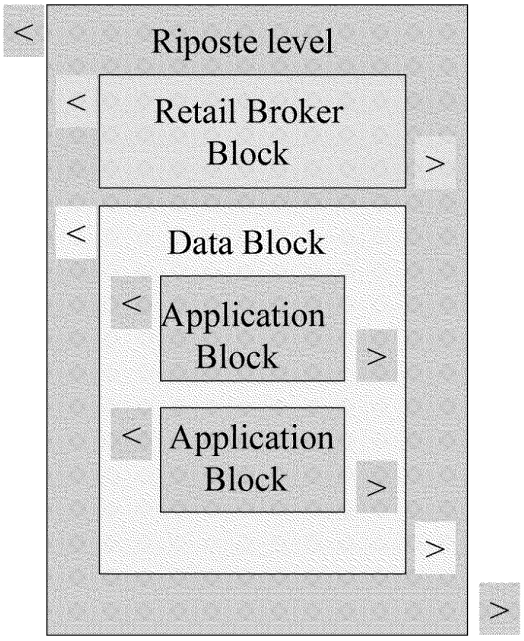


Figure 3-10 Nesting of data

FUJ00001368

4 Business Functions

This section describes the business functions that are supported by the Horizon system and the context within which application development takes place. It describes the concepts involved and how they are handled by the system. It identifies and specifies the structures that are required and that may be used to pass data between different system elements.

The section describes the following topics:

- The Horizon EPOS Service
- Settlement
- Stock unit management
- Reporting
- Balancing

Business is transacted in an outlet by a clerk (or other employee) at a till, using a shared or individual stock unit.

Stock units

Each outlet operates one or more stock units. Although there are many patterns of working, typically one clerk is responsible for the contents of any one stock unit, so it is possible to reconcile (balance) them on demand.

Balance Period

The Balance Period defines the periods between committed balances. There may be many Balance Periods for a particular stock unit within a Cash Account, particularly when the responsibility for a stock unit is handed over from one clerk to another – possibly owing to part-time working practices. Note that in the case of a shared stock unit, several physical drawers may comprise the stock unit.

Cash Account Period

The outlet reporting period is by Cash Account Period (normally a week running from Thursday to Wednesday) although a facility is provided to extend the Cash Account Period if necessary.

Sessions

During a Balance Period, business is transacted in sessions. A mode of operation is associated with each session and is one of the following: Sale, Refund/Reversal, Remittance, Revaluation, Transfer. The current mode is checked by the system before a transaction is performed.

FUJ00001368

Transactions

Transaction records are collected into sessions and written to the message store. They refer to individual products that have been transacted. Transaction record additional data is written to the message store for some product types; it is derived from Reference Data supplied by POCL and is used to capture additional data for a POCL Client.

4.1 EPOSS

The Electronic Point Of Sale Service (EPOSS) is the Horizon application that provides Electronic Point of Sale (EPOS) functionality. EPOSS is driven by the rules that define the products that can be sold, their retail price and any business constraints, such as discounts, that apply. This section describes how products are defined and sold in EPOSS, and the structure of the data interfaces involved.

The interfaces described in section 4.1.3 are those that provide EPOSS, TIP and other Client applications with transaction and Cash Account details.

The interfaces described in section 4.1.4 and 4.1.5 describe the Reference Data structures that define POCL products and tokens.

4.1.1 EPOSS processes and products

Product definitions are supplied by POCL in the form of Reference Data, which is distributed to the outlets so that EPOSS and other applications can use the same set of business rules. Not all data is relevant to each outlet and data is delivered only to those outlets that require it; its distribution is determined by a number of data items defined in Reference Data.

Data that is required by all outlets is referred to as core data. Non-core data is data that is required by only some outlets: Vehicle Licensing, for example, is performed at a limited number of outlets and only they receive the relevant data. New and changed Reference Data is supplied by POCL and POCL Clients in files that must conform to the appropriate Application Interface Specification (AIS). Once validated by the ICL Pathway Reference Data Management Centre (RDMC) POCL Reference Data is converted into a format suitable for transmission to outlets, using Riposte

It is the structure of this data, held in attribute grammar format, which is described below. Each application at the counter has access to this data and must use it to ensure that the correct accounting data is recorded, and that any additional data required by POCL or the Client is captured when a transaction is completed.

All products have a unique reference number and a number of characteristics that are referred to in the Horizon environment as attributes.

FUJ00001368

Products with numbers in the range 10,000 – 20,000 are reserved as private products required by ICL Pathway. The range has been agreed with POCL.

4.1.1.1 Product attributes

Product attributes are those that are required to support the transaction of the product, such as price and description, and are as originally defined in POCL Reference Data before its transformation into the Reference Data that is transmitted to the counters. Products include settlement products (Methods of Payment) as well as traded products and services.

4.1.1.2 Product relationships

There may be a relationship between two products, ensuring that they are traded together. An example of this is the range of postal order products, which must always be transacted with an associated fee (mandatory relationship). Transcash may be transacted with or without a fee (non-mandatory relationship).

4.1.1.3 Product pre-conditions

There may be pre-conditions governing the use of a product, and these are defined in Reference Data. Within the EPOSS application, for example, TV licence stamps can only be used as a settlement product if a TV licence has been transacted earlier in the same session. This pre-condition is checked both when the Method of Payment (the TV licence stamp) is selected and again at the point of settlement. If the transaction, TV licence, is voided before settlement is selected, then the MoP, the TV licence stamp, must also be voided before settlement can be completed.

4.1.1.4 Additional data

Products are made up of ‘core’ and ‘extended’ attributes of which some attributes or attribute groups are defined as optional. The core attributes apply to all products and the extended attributes apply only to certain product types.

EPOSS has only one set of extended attributes. It is called ‘Additional Data’ and is used to capture additional data that is specific to a particular class of transaction. Different classes of transaction expect different sets of additional data and these are recorded as separate Client transaction attribute groups.

4.1.1.5 Session Effect attribute

The **SessionEffect** attribute describes a product’s behaviour at point of sale in absolute terms, thereby allowing the system to calculate the session balance in differing functions such as Serve Customer and Refund/Reversal.

FUJ00001368

The **SessionEffect** attribute defines 'In' as meaning an increase in the balance due to the Post Office. An example of an 'In' product is a stamp sale or a National Savings deposit. An example of an 'Out' product is a green or violet Girocheque.

The attribute does not drive the accounting and reporting mechanisms; an 'In' product may not always appear on the receipts side of the printed balance report. For example, for consistency, Rent Summaries appear on the same side of the balance as Rent Vouchers.

4.1.1.6 Token impulses

Token impulse definitions identify and route data from data capture devices through to interested applications, transforming the data into the elements required by the target application.

The token impulse contains core attributes and a repeating attribute defining each element of the token data. Elements have a defined 'Picture' that is used to identify a token when it is read and to uniquely identify its type.

Tokens are only valid in assigned modes, ensuring that token driven transactions only take place in an appropriate context. For example, an automated payment card may be swiped and will invoke the relevant application when the clerk is in 'Serve Customer' mode, but in 'Rem In' mode, it will have no effect.

Token input is validated against token definitions registered by the application with the Peripheral Server. Because of special security requirements, a separate process deals with details for smart cards.

4.1.1.7 Selling a product

A product is sold when it is transacted during a customer session, via the Retail Broker interfaces, and appears on the transaction stack. A customer session may consist of the sale of a single product or it may be a series of transactions that terminate when settlement of the complete session occurs. At this point all the transactions on the stack are committed and the generated transaction records are written to the message store.

The sale of a product is started when a product is selected from a menu or when input is received from another peripheral device, such as a card swipe. Whatever the route, the required transaction must be identified, and after any additional clerk interaction to capture additional data, a transaction record created. For token input, default transaction details, derived from token data can be used.

Some products have mandatory links with another product, which may result in several transaction records being generated. A transaction on the Retail Broker stack may contain several records that constitute a single transaction; for example, if a postal order is transacted, its associated fee generates a second record.

FUJ00001368

Transactions are described in a similar way to products and contain core and extended attributes. The core data applies to all transactions, and the extended data only to transactions of a specific type. One extended transaction type is Additional Data, which is a transaction where specific extended data has been captured from Reference Data, for purposes irrelevant to the core EPOSS. Data other than this must be supplied using a separate structure that is specific to an application and Client.

4.1.2 EPOSS data model

EPOSS data interfaces are based on the need to access the Reference Data that defines POCL products and tokens, and the need to provide EPOSS and the POCL TIP system with transaction and Cash Account details.

These interfaces are described in sections 4.1.3 and 4.1.4 as data models, in a structured manner that identifies the 'owner' of the data (the software responsible for its creation), and the 'recipient' of the data (the system outside the ICL Pathway service to which it is sent).

In these data model diagrams, the leaf in each structure represents the data item that must be provided to support the interface. If the item is optional, it is marked with an 'o'. If it can occur many times in the same structure, it is marked with an '*'.

The data models are implemented as attribute grammar, which is itself a structured way of defining data and so lends itself to this approach.

4.1.2.1 Attribute grammar structure

Attribute grammar defines each data item as an item name followed by a colon (:) and the data itself. Each data item is delimited by < and > characters (angle brackets).

The sequence of attributes is not important. Any application processing this string looks for the name or value, rather than its position. Similarly, the application ignores any field that is of no interest.

Attribute name cannot be null and can contain no full stops, angle brackets, colons or double quotation marks. Attribute value is either a string or an attribute grammar string and can contain blanks but not angle brackets. Some other characters, notably ampersands, can cause difficulties; refer to Escher documentation for details. Where these characters can be accommodated, the resolution is to include two consecutive instances of the characters in question.

The structure of messages can become complex; Riposte's **redit** utility can be used to display the nested structure of attribute grammar. The example:

<EG1:<AAName:<AANext:bbb><BBNext:ccc>><AAOther:yy>>

appears as follows, using the **redit** utility:

FUJ00001368


```
<EG1:
    <AAName:
        <AANext:bbb>
        <BBNext:ccc>
    >
    <AAOther:yy>
>
```

It may be useful at this point to refer to section 5.5, which describes basic Riposte functions and concepts, before continuing with the rest of section 4.

4.1.2.2 Transient records

The data records created to support EPOSS and TIP, and any other Client application, are transient; they are deleted from both the correspondence servers and the counter after an expiry period, which is defined (in days) by the application and which varies depending upon the business rules that apply. The expiry period should be as short as is consistent with the business rules so that system performance and capacity are not impacted. Minimum and maximum values for expiry are defined at system level and may override application-specified values.

At the correspondence server there is a separate audit function which ensures that all messages created, either at the outlet or correspondence servers, are archived for a period of 18 months for audit purposes.

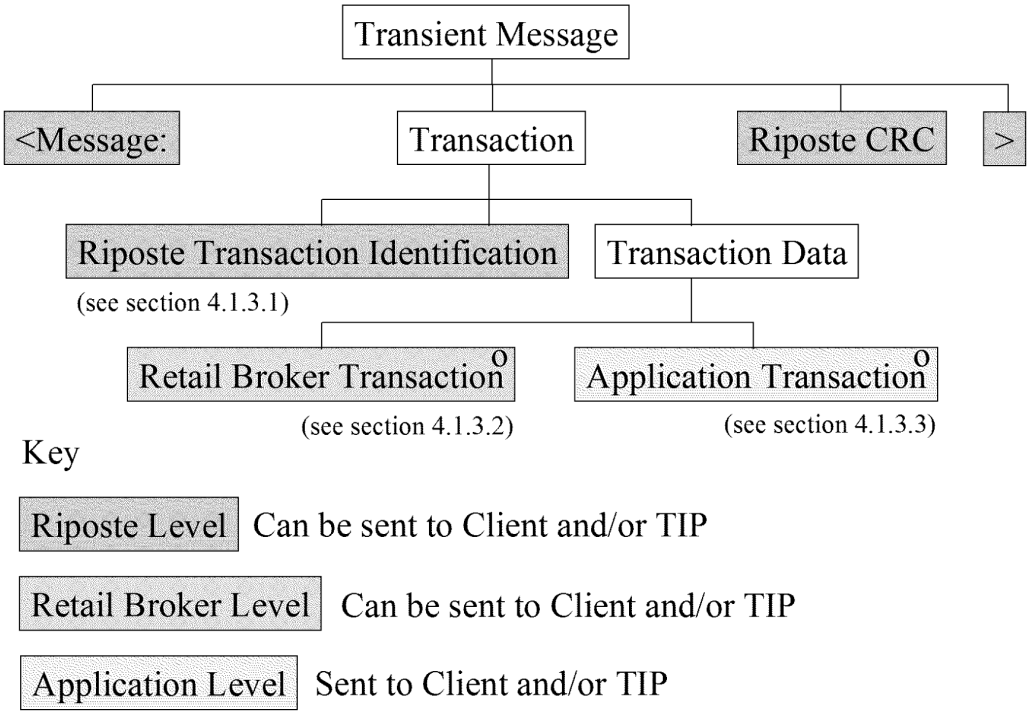
4.1.2.3 Persistent objects

The data records that define Reference Data are by their nature temporal and remain as persistent objects until their defined ‘End Date’ is reached. Non-Reference Data persistent objects do not have an expiry date and are not temporal in this way. The attributes of a persistent object remain unchanged throughout a transaction, even if a new version of the persistent object becomes current during the duration of the transaction (see section 6.3.4 for information about Desktop reload). However, because the current product data persistent object is obtained each time the product is transacted, it is possible for different versions of a persistent object to be used during a single customer session. If a new version becomes current after it has been transacted during a session, any subsequent transaction of the product will use the new version of the product data persistent object.

FUJ00001368

4.1.3 Transient messages

The generic format of a transient message is shown in Figure 4-1 below. Each item has structure <ItemName:ItemValue>:



o optional
***** can occur many times in the same structure

Figure 4-1 Transient messages

All Riposte messages, whether transient or persistent, have the same high-level structure. They all:

- Start with a defined attribute or tag
- Contain transaction data
- Contain security data including Cyclic Redundancy Check data
- Define each data item as an item name followed by a colon (:) and the data itself.
- Delimit each data item by < and > characters (angle brackets)

FUJ00001368

Each transaction in a transient message has to be identified and this identification is provided automatically by Riposte. If the message is written via the Retail Broker, it contains items that identify it in a session context. If the message is not written via the Retail Broker, this session data is not present.

Each application has to create the following data:

- The data it needs to support its own Client application.
- The data needed by EPOSS to support its Electronic Point Of Sale (EPOS) requirements as defined in POCL-supplied Reference Data.
- Other data required by EPOSS for Cash Account reporting purposes, also defined by Reference Data.

Each level of the data model is described separately in the following sections.

FUJ00001368

4.1.3.1 Riposte level

It is at the Riposte level that the identification of the overall transaction is maintained. The post office and counter position are identified, as are the time at which the transaction occurred and the clerk or system responsible for creating the transaction. Most of this data is passed to TIP and can be passed to Client systems.

(from section 4.1.3)

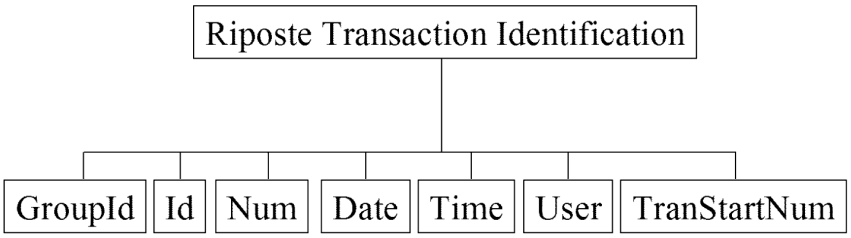


Figure 4-2 Riposte-level data

4.1.3.1.1 Data names and values

Name	*Value*	*TIP usage*
Date	Date: dd-Mon-yyyy	
GroupId	FAD code (without the check digit)	Organisational unit code
Id	Counter node	Till identifier
Num	Sequence number of message	
Time	Time: hh:mm:ss UTC time (GMT)	
TranStartNum	Always present but value not used by applications.	
User	User ID. Only present if message written by a logged on user.	Employee identifier

4.1.3.1.2 Attribute grammar fragment

```
<GroupId:123456>
<Id:1>
<Num:43181>
<Date:13-Dec-1999>
<Time:08:13:27>
<TranStartNum:43181>
<User:AMGR01>
```

FUJ00001368

4.1.3.2 Retail Broker level

If the Retail Broker is used to create the message, session and stock unit data is automatically included in the message. Session data identifies the sequence number that applies to all transactions in the session: each transaction has its own unique transaction sequence number within the session. Times are recorded in hours, minutes, seconds and time fractions (TFs).

Mode identifies the mode in which the transaction was conducted. For applications other than EPOSS these cover normal Serve Customer (SC) mode, reversal either in session or not (RV, ER) or fallback recovery mode (REC).

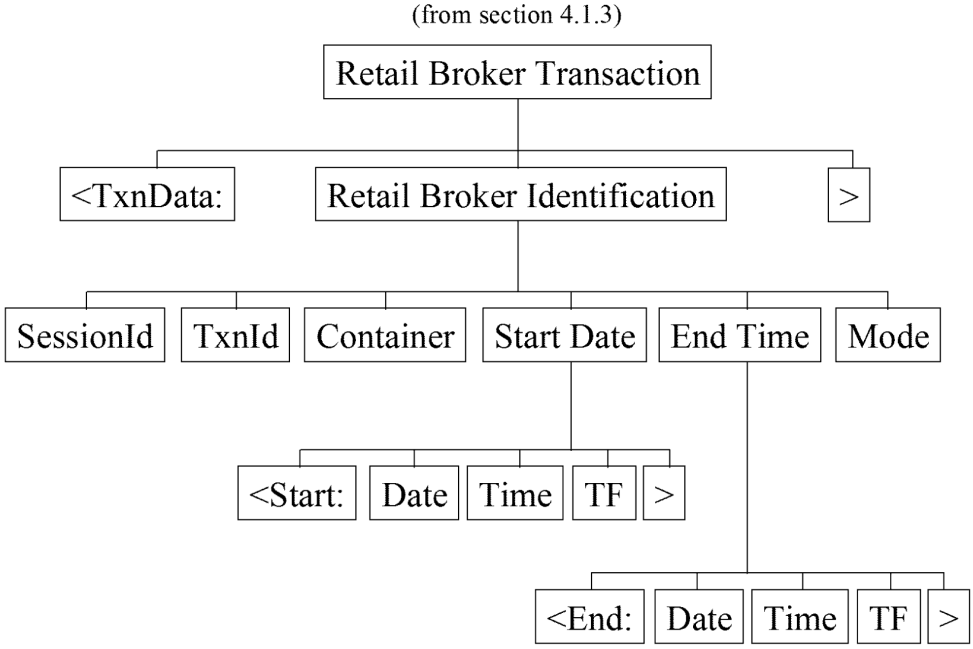


Figure 4-3 Retail Broker data

FUJ00001368

4.1.3.2.1 Data names and values

Name	*Value*	*TIP usage*
Container	Stock unit name	Stock unit identifier
Date (End)	Date the transaction ended (dd-Mon-yyyy)	Date of transaction
Date (Start)	Date the transaction commenced (dd-Mon-yyyy)	Date of transaction
Mode	Mode of the system when this transaction was written (session mode): ER = Existing Reversal EV = Reversal not in session HK = Housekeeping NAD = Non Accounting Data PT = Parcel Traffic RD = Revaluation Down REC = Recovery RIAD = Remittance In (AutoDistribution) RICL = Remittance In (Client) RIOP = Remittance In (Other PO) RISD = Remittance In (Supply Division) ROAD = Remittance Out (AutoDistribution) ROCL = Remittance Out (Client) ROOP = Remittance Out (Other PO) RODC = Remittance Out (Data Centre) ROSD = Remittance Out (Supply Division) RU = Revaluation Up RV = Reversal in session SC = Serve Customer TI = Transfer In TO = Transfer Out	
SessionID	Unique identifier for all transactions within a customer session, currently derived from the licence number of Riposte, the GroupId, Id and a value of Num.	Session sequence number
TF (End)	Time fraction (0-9). Nearest tenth of a second.	
TF (Start)	Time fraction (0-9). Nearest tenth of a second.	
Time (End)	Time the transaction ended (hh:mm:ss)	Time of transaction
Time (Start)	Time the transaction commenced (hh:mm:ss)	Time of transaction
TxnId	Unique identifier for all the messages within a customer transaction. Related to SessionID	Transaction ID

FUJ00001368

4.1.3.2.2 Attribute grammar fragment

```
<TxnData:
    <Container:A02>
    <Mode:SC>
    <SessionId:44-123456-1-43180-1>
    <TxnId:44-123456-1-43180-2>
    <Start:
        <Date:13-Dec-1999>
        <Time:08:13:02>
        <TF:9>
    >
    <End:
        <Date:13-Dec-1999>
        <Time:08:13:03>
        <TF:1>>
    >
```

FUJ00001368

4.1.3.3 Application level

The responsibility for data is as follows:

- It is the responsibility of the application to record its own identification, and to set the expiry period (the time after which the message may be archived). Both values must be agreed with ICL Pathway.
- Debit and Credit are supplied by the Retail Broker and reflect the unsigned value recorded for each transaction. They are relative to the stock unit content.
- If additional data is required to support a Client application, it can be included, using an attribute name agreed with ICL Pathway.
- The application is also responsible for providing the transaction data needed to support EPOSS.

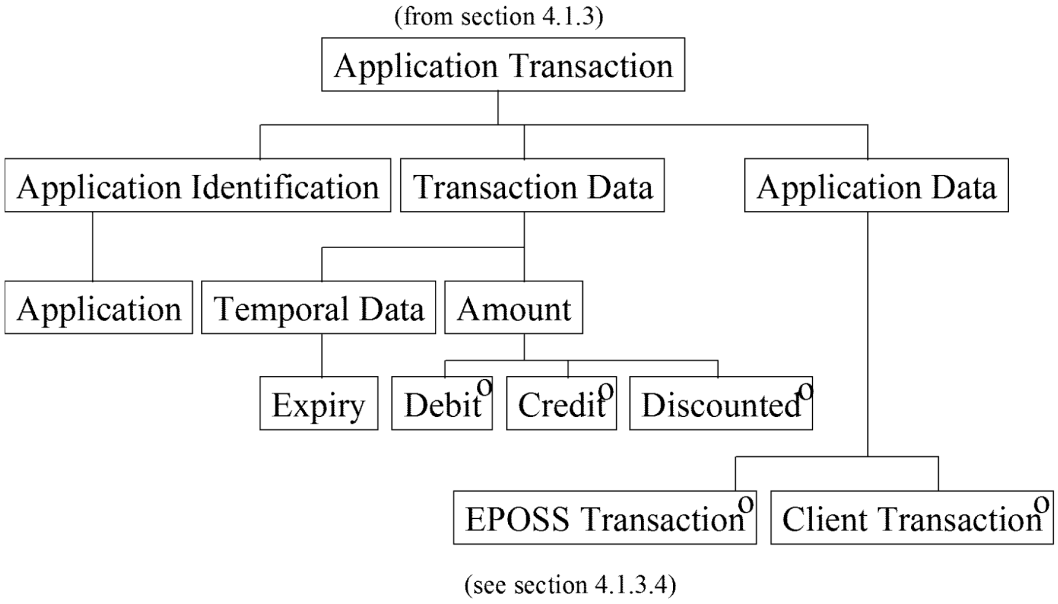


Figure 4-4 Application-level data

4.1.3.3.1 Data names and values

Name	*Value*
Application	Name of the application generating the transaction
Client Transaction	Actual format depends on the application and so is not included in this document.
Credit	Unsigned pence (optional) - used when money is being 'paid' into the drawer
Debit	Unsigned pence (optional) - used when money is being 'paid' out of the drawer

FUJ00001368

Discounted	Transaction was discounted (optional): 'True' or 'False'.
Expiry	Number of days after which the message is considered for archiving and deletion

4.1.3.3.2 Attribute grammar fragment

```
<Application:EPOSSAppMain>
<EPOSSTransaction:
      <a:b>
>
<Credit:10100>
<Discounted:True>
<Expiry:35>
```

FUJ00001368

4.1.3.4 EPOSS Transaction level

Each application is responsible for the following:

- Extracting from POCL Reference Data the data items needed to identify the POCL product involved.
- The EPOSS data and any additional data that needs to be captured as defined in the Reference Data.
- The Cash Accounting data as defined in Reference Data.

The application has to take account of Item Transaction Mode when dealing with products, as this information is required by POCL's Transaction Information Processing (TIP). If it is possible to transact more than one unit of the product, both the quantity (Qty) and sale value (SaleValue) have to be recorded. The sale value is signed to indicate its relationship to the stock unit.

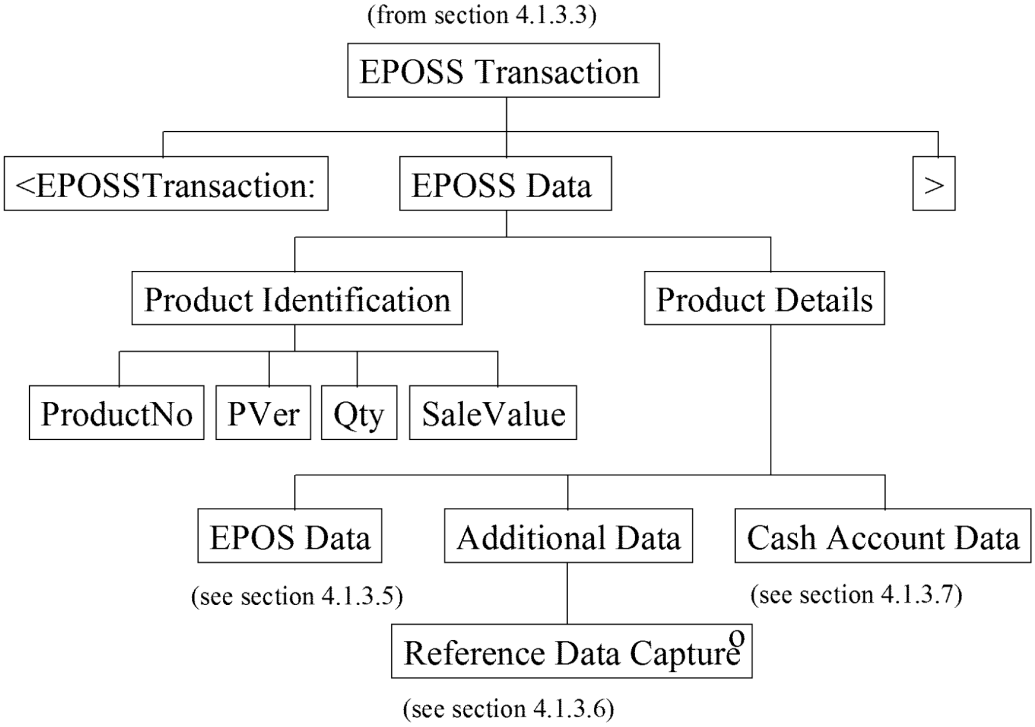


Figure 4-5 EPOSS Transaction-level data

FUJ00001368

4.1.3.4.1 Data names and values

Name	*Value*	*TIP usage*
ProductNo	Product reference number	Item ID
PVer	Product version when this transaction took place	Version number
Qty	Signed count of products. Positive denotes quantity leaving the stock unit	Quantity
SaleValue	Signed value of transaction in decimal pounds. Positive denotes value leaving the stock unit	Amount

4.1.3.4.2 Attribute grammar fragment

```
<ProductNo:2220>
<Qty:1>
<PVer:17>
<SaleValue:101>
```

FUJ00001368

4.1.3.5 EPOS level data

Most EPOS-level data is not relevant to any application other than EPOSS, but EPOSS does require the mode (M) to be set for each transaction, irrespective of which application created it.

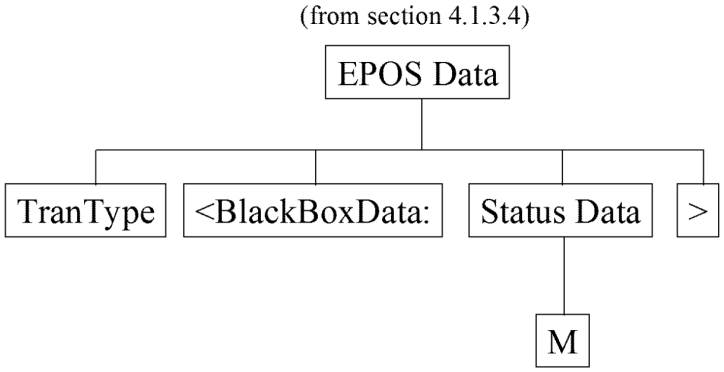


Figure 4-6 EPOS-level data

4.1.3.5.1 Data names and values

Name	*Value*	*TIP usage*
M	Transaction mode. For a list of possible values, refer to *TIP Application Interface Specification*	Reversal indicator if M= ER or RV Fallback mode flag if M=REC Transaction mode if M=C.

4.1.3.5.2 Attribute grammar fragment

```
<TranType:S>
<BlackBoxData:
      <M:SC>
>
```

FUJ00001368

4.1.3.6 Reference Data capture level

Reference Data may require an application to capture additional data during the customer session. The structure of that data and the name of each item are defined in the Reference Data persistent object for that product.

For TIP, the captured data is passed across the TIP interface in the way defined in Appendix D of *TIP Application Interface Specification*.

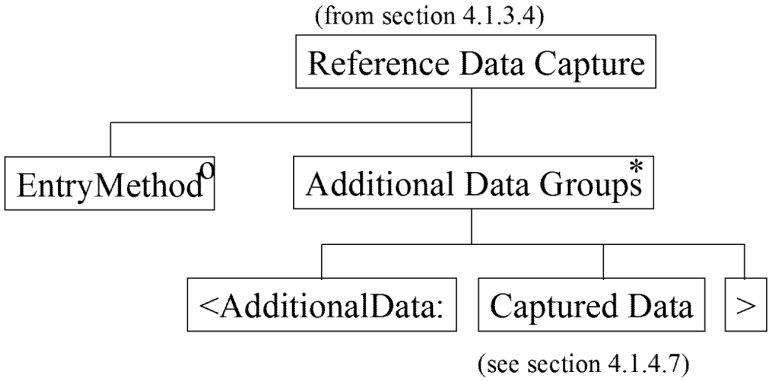


Figure 4-7 Additional data

4.1.3.6.1 Data names and values

Name	*Values*	*TIP usage*
EntryMethod	0 bar-code 1 manual 2 magnetic card 3 smart card 4 smart key 5 scales	Method of Data Capture

4.1.3.6.2 Attribute grammar fragment

<EntryMethod:1>

FUJ00001368

4.1.3.7 EPOSS Cash Account data

For EPOSS Cash Account purposes, data must be recorded to allow the transaction to be recorded against the appropriate Cash Account lines. This definition is again recorded in the Product Reference Data and must be copied by the application into each transaction.

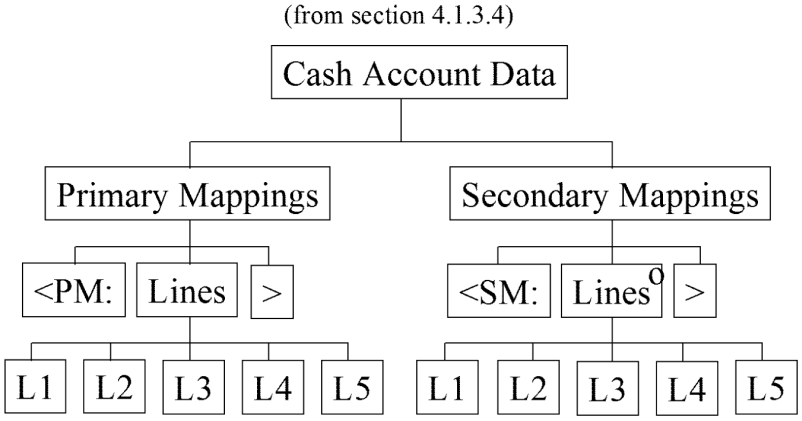


Figure 4-8 Accounting level data

4.1.3.7.1 Data names and values

Levels 1 and 2 are owned by POCL. Levels 3, 4, and 5 are owned by ICL Pathway. All elements (L1, L2, L3, L4, L5) are present but may be null (<L1:>, for example).

Name	*Value*
L1 (PM)	Primary mapping: line 1
L2 (PM)	Primary mapping: line 2
L3 (PM)	Primary mapping: line 3
L4 (PM)	Primary mapping: line 4
L5 (PM)	Primary mapping: line 5

FUJ00001368

Secondary mappings are used to track transactions in modes other than Serve Customer, such as transfers and remittances. There may be a null string (<SM:>). If present, all elements (L1, L2, L3, L4, L5) are present but may be null (<L1:>, for example).

Name	*Value*
L1 (SM)	Secondary mapping: line 1
L2 (SM)	Secondary mapping: line 2
L3 (SM)	Secondary mapping: line 3
L4 (SM)	Secondary mapping: line 4
L5 (SM)	Secondary mapping: line 5

4.1.3.7.2 Attribute grammar fragment

```
<PM:
    <L1:1061>
    <L2:2032>
    <L3:3006>
    <L4:3013>
    <L5:3017>
>
<SM:>
```

FUJ00001368

4.1.3.8 Example transaction record

```
<Message:                                          section 4.1.3
    <GroupId:123456>                               section 4.1.3.1
    <Id:1>                                         section 4.1.3.1
    <Num:43181>                                    section 4.1.3.1
    <Date:13-Dec-1999>                             section 4.1.3.1
    <Time:08:13:27>                                section 4.1.3.1
    <User:AMGR01>                                  section 4.1.3.1
    <TranStartNum:43181>                           section 4.1.3.1
    <TxnData:                                      section 4.1.3.2
        <SessionId:44-123456-1-43180-1>            section 4.1.3.2
        <TxnId:44-123456-1-43180-2>            section 4.1.3.2
        <Container:A02>                        section 4.1.3.2
        <Start:                                    section 4.1.3.2
            <Date:13-Dec-1999>                     section 4.1.3.2
            <Time:08:13:02>                    section 4.1.3.2
            <TF:9>                                 section 4.1.3.2
        >
        <End:                                      section 4.1.3.2
            <Date:13-Dec-1999>                     section 4.1.3.2
            <Time:08:13:03>                    section 4.1.3.2
            <TF:1>                                 section 4.1.3.2
        >
        <Mode:SC>                                  section 4.1.3.2
    >
    <Expiry:35>                                    section 4.1.3.3
    <Application:EPOSSAppMain>                     section 4.1.3.3
    <EPOSSTransaction:                             section 4.1.3.4
        <ProductNo:2220>                           section 4.1.3.4
        <Qty:1>                                    section 4.1.3.4
        <PVer:17>                                  section 4.1.3.4
        <SaleValue:101>                            section 4.1.3.4
        <BlackBoxData:                             section 4.1.3.5
            <M:SC>                                 section 4.1.3.5
        >
        <TranType:S>                               section 4.1.3.5
        <PM:                                       section 4.1.3.7
            <L1:1061>                              section 4.1.3.7
            <L2:2032>                              section 4.1.3.7
            <L3:3006>                              section 4.1.3.7
            <L4:3013>                              section 4.1.3.7
            <L5:3017>                              section 4.1.3.7
        >
        <SM:>                                      section 4.1.3.7
    >
    <Discounted:True>                              section 4.1.3.3
    <Credit:10100>                                 section 4.1.3.3
    <CRC:79665F1A>                                 section 4.1.3
>
```

FUJ00001368

4.1.4 Persistent objects

The generic format of a persistent object is shown below in Figure 4-9.

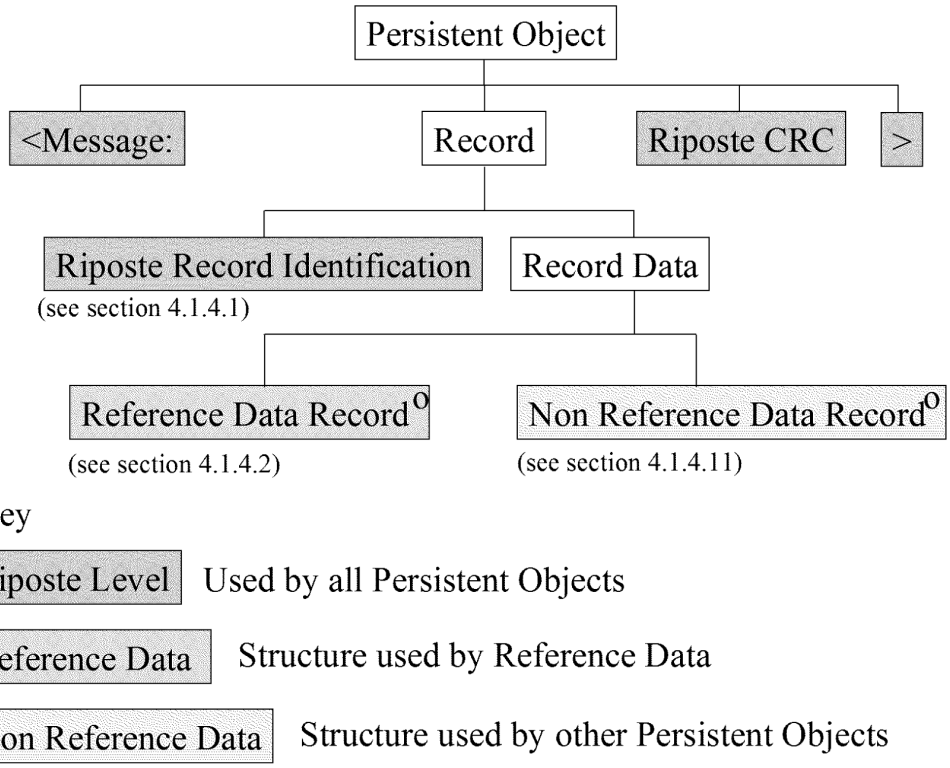


Figure 4-9 Persistent object

Each persistent object has to be identified. The identification is a combination of that provided by Riposte, and any temporal information recorded for that object. Persistent objects fall into two categories:

- Persistent objects that apply to Reference Data

  From the point of view of the interfaces needed for application development, it is the Reference Data definitions for products, Cash Account mappings and tokens that are important. There are many other types of Reference Data records but these are either desktop- or EPOSS-specific. Reference Data persistent objects always contain the attribute name 'RData' in their attribute grammar strings (see section 4.1.4.2).

- Persistent objects that do not apply to Reference Data

  Types of non-Reference Data persistent objects define messages to the clerk and error messages. These are application-specific and belong to collections whose names are defined in section 9 of this document.

FUJ00001368

Non-Reference Data persistent objects do not have a tag 'RData' (see section 4.1.4.2).

4.1.4.1 Riposte level

The name of each persistent object is held at the Riposte level, together with the name of the collection to which it belongs. Each Reference Data object is temporal in that it has a start date and an end date. To support this concept, Riposte maintains a separate object that points to the latest version of the object. The suffix within this object is used to determine the latest version.

Not all Reference Data is applicable to each outlet and it is the 'Depend' attribute that defines this relationship for all product data.

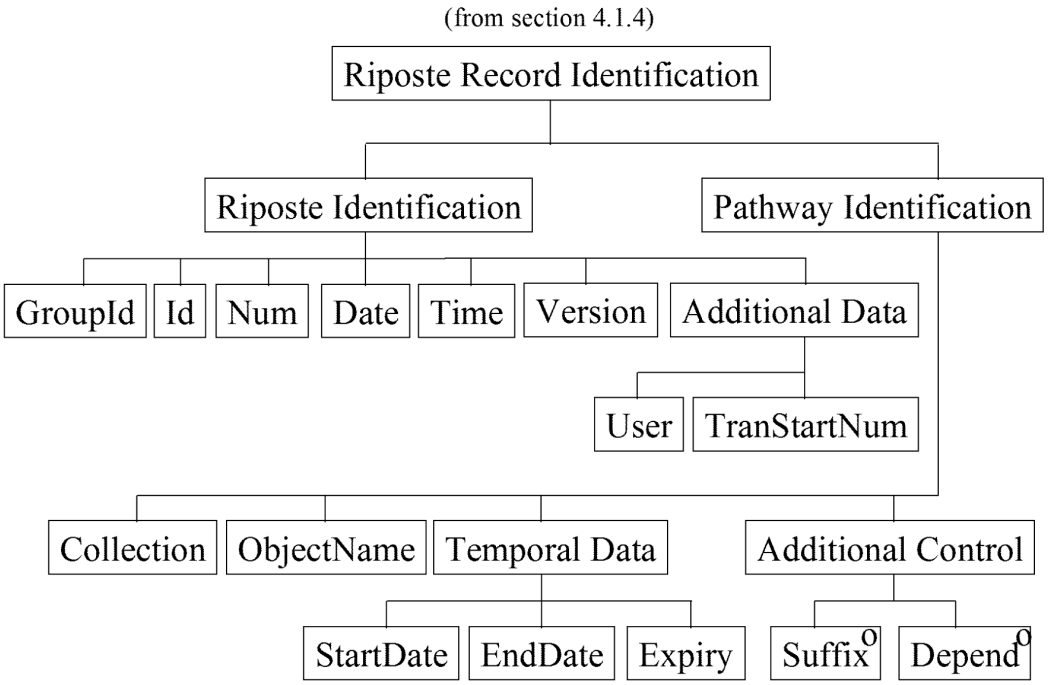


Figure 4-10 Persistent objects: Riposte level

FUJ00001368

4.1.4.1.1 Data names and values

Name	*Value*
Collection	Name of the collection of objects
Date	Date: dd-Mon-yyyy
Depend	Identifies whether or not the object applies in this outlet (optional). It is used to control non-core product usage; the Reference Data for a non-core item is sent to all outlets that have sold the product in the past.
EndDate	End date (dd-Mon-yyyy hh:mm:ss)
Expiry	Number of days (after Date) after which the message is considered for archiving and deletion
GroupId	FAD code (without the check digit)
Id	Counter node
Num	Sequence number of message
ObjectName	Name of the object within the collection.
StartDate	Start date (dd-Mon-yyyy hh:mm:ss)
Suffix	Supports multiple versions of temporal objects (optional)
Time	Time: hh:mm:ss UTC time (GMT)
TranStartNum	Always present but value not used by applications.
User	User ID. Only present if message written by a logged on user.
Version	Establishes current version of the persistent object.

4.1.4.1.2 Attribute grammar fragment

```
<GroupId:4038>
<Id:1>
<Num:18300>
<Date:23-Nov-1999>
<Time:15:36:00>
<Expiry:34>
<TranStartNum:18145>
<Collection:_EPOSSProducts>
<ObjectName:622_11>
<StartDate:01-Jan-1996 00:00:01>
<EndDate:>
<Version:3>
```

FUJ00001368

4.1.4.2 Reference Data types

The types of Reference Data that have a direct impact on application development are:

- Product Reference Data

  Defines the POCL products that can be transacted at the counter.

- Cash Account definitions

  Define the relationship between the products and Cash Account lines.

- Token definitions

  Define the structure and content of data on bar-codes, magnetic cards and smart cards.

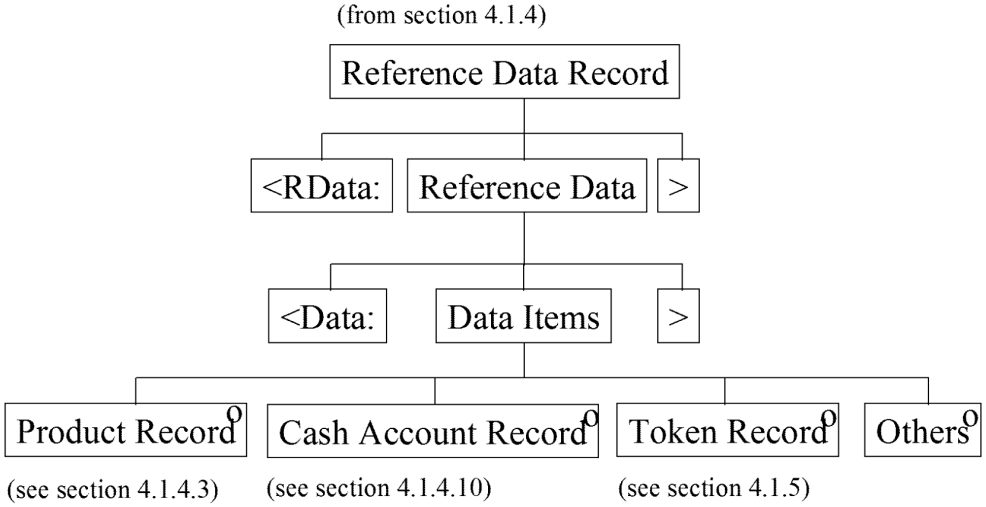


Figure 4-11 Persistent objects: Reference Data

The RData tag defines the start of the data that is passed down to Desktop applications. The Reference Data required to develop new applications includes product, cash account and token definitions. It is possible that other kinds of Reference Data may also be required for a new application, in which case such Reference Data will have its own data structures.

FUJ00001368

4.1.4.3 Product Reference Data

For product Reference Data, two persistent objects are of interest:

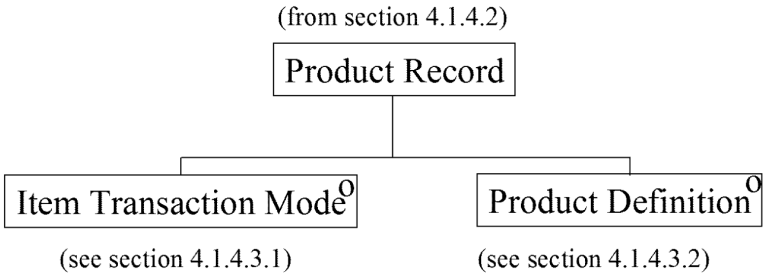


Figure 4-12 Persistent objects: Product record

4.1.4.3.1 Item Transaction Mode

The definition of Item Transaction Mode is as follows:

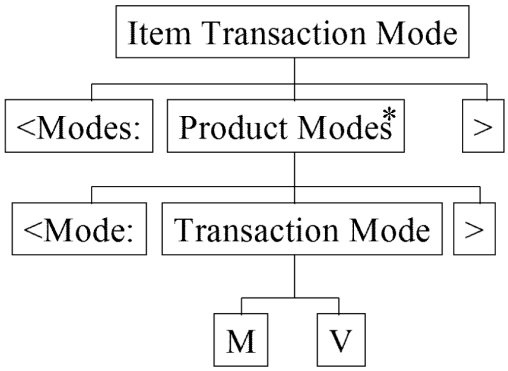


Figure 4-13 Persistent objects: item transaction mode

FUJ00001368

4.1.4.3.1.1 Data names and values

Mode identifies in which POCL-defined modes the product may be transacted. From the application point of view these are:

Mode	*Description*	*TIP usage*
M	Transaction mode: nnn	Transaction mode code: DDN = Declare Discrepancy Positive (Up) DDP = Declare Discrepancy Negative (Down) HK = Housekeeping NAD = Non Accounting Data PT = Parcel Traffic RD = Revaluation Down REC = Recovery RIAD = Remittance In (AutoDistribution) RICL = Remittance In (Client) RIOP = Remittance In (Other PO) RISD = Remittance In (Supply Division) ROAD = Remittance Out (AutoDistribution) ROCL = Remittance Out (Client) ROOP = Remittance Out (Other PO) RODC = Remittance Out (Data Centre) ROSD = Remittance Out (Supply Division) RU = Revaluation Up RV = Reversal in session SAN = Stock Adjustment Negative SAP = Stock Adjustment Positive SC = Serve Customer TI = Transfer In TO = Transfer Out
V	Version number	Version of item transaction mode (numeric)

See section 4.2.3 for details of reversals, refunds and voids.

4.1.4.3.1.2 Attribute grammar fragment

```
<Modes:
    <Mode:
        <M:REC>
        <V:9>
    >
>
```

FUJ00001368

4.1.4.3.2 Product definition

Any product definition identifies the product number (PN), the version (V) of the product and the pick list group (PG) it belongs to. It also defines the names for that product: short (SN), long (LN) and the name to be printed on receipts (RN). There may also be an instruction that has to displayed to the clerk when this product is transacted (SI).

An example of a product definition is given in 4.1.4.13.

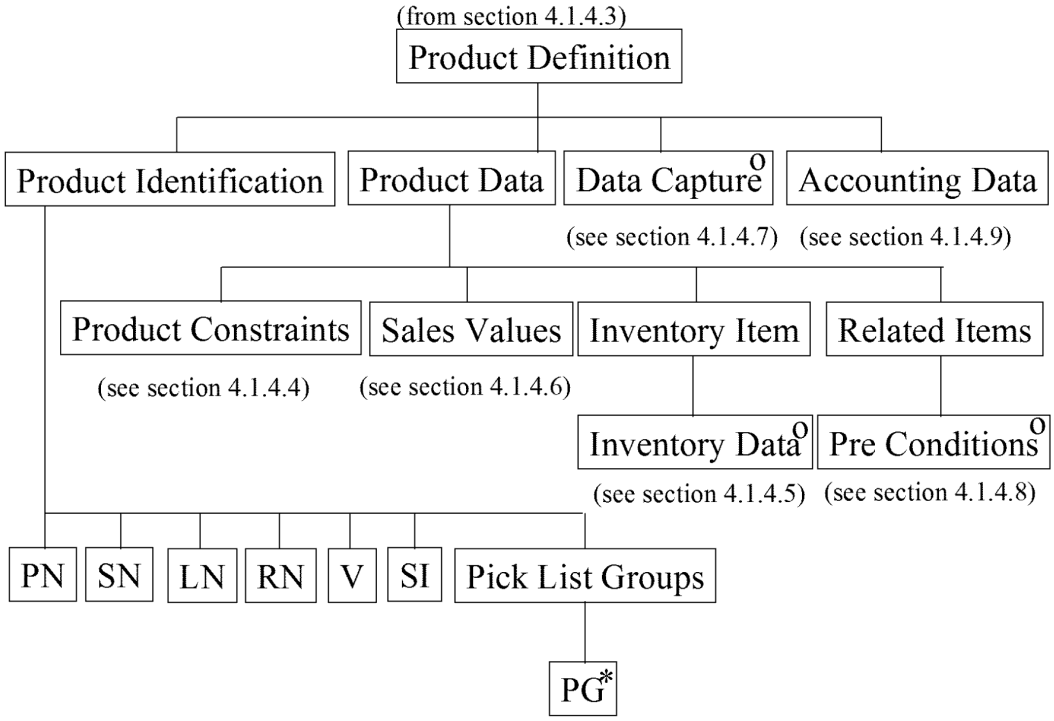


Figure 4-14 Persistent objects: product definition

4.1.4.3.3 Data names and values

Name	*Value*
LN	Long name (max 24 characters).
PG	Product group. Product number of pick list to which this product belongs on the menu hierarchy.
PN	The product number
RN	Name to be printed on receipts (medium name: max 16 characters).
SI	An instruction to be displayed to the clerk when this product is transacted. May be null.
SN	Short name (max 10 characters)

FUJ00001368

V	Version of the product

The screen font used is Arial Narrow and the maximum number of characters that may be used in the name fields depends upon which characters used, as shown in the following table:

Case and character size	*Letters and numbers*	*Number occupying equivalent space*
Lower case: small	l j I f t	18
Upper case: small	I J L F T	9
Lower case: standard	a b c d e g h k n o p q r s u v x y z 0 - 9	10
Upper case: standard	A B C D E G H K N O P Q R S U V X Y Z	7
Lower case: large	w m	7
Upper case: large	A B C D E G H K N O P Q R S U V X Y Z	6

4.1.4.3.4 Attribute grammar fragment

<LN :£5 Stamp>
<PG:21>
<PN:622>
<RN:£5 stmp>
<SN:£5 Stamp>
<SI:>
<V:11>

FUJ00001368

4.1.4.4 Product constraints

There can be constraints applied to transacting a product, such as the minimum (MnV) and maximum (MxV) value that can be sold in one transaction. Similarly, the minimum (MnQ) and maximum (MxQ) allowable quantity can be defined.

It may be mandatory for a receipt to be produced when the product is transacted (SR), in which case the receipt number (Rcptno) and receipt type (RT) can be specified.

Some products must be transacted together, in which case there is a mandatory product (MP) relationship. Some products can be reversed (RV) in which case a reversal authority may be required (RA). Other products can be voided (VO).

There may be a default service type (ST).

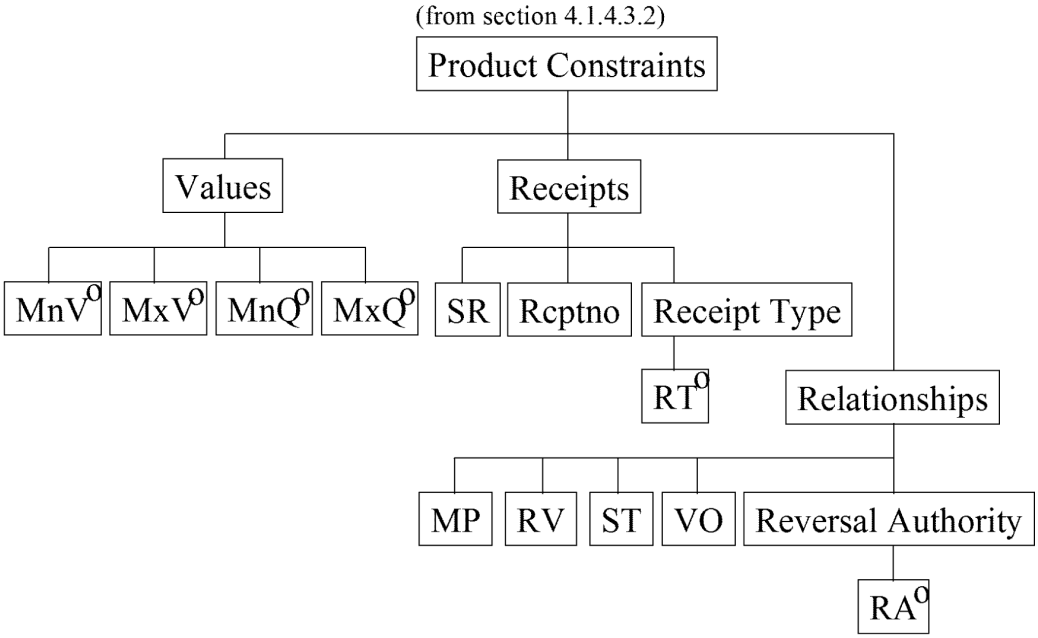


Figure 4-15 Persistent objects: product constraints

4.1.4.4.1 Data names and values

Name	*Value*
MnQ	Minimum allowable quantity (optional)
MxQ	Maximum allowable quantity (optional)
MnV	Minimum transaction value allowed - decimal pounds (optional)

FUJ00001368

MxV	Maximum transaction value allowed - decimal pounds (optional)
MP	Mandatory product. Indicates that the product with the product number stated must be transacted along with this product. It is a mandatory product relationship definition that is null if no such relationship applies.
Rcptno	Receipt number (if a receipt is mandatory). May be zero.
RA	Reversal Authority (optional). Identifies the application that is responsible for authorising a transaction reversal.
RT	Receipt type (optional). Identifies a Receipt Definition for this Product (Id of report definition)
RV	Indicates whether transactions for this product are reversible
SR	Mandatory for a receipt to be produced when the product is transacted
ST	POCL-defined Service Type code for any transactions for this product. May be null.
VO	Voidable: ‘True’ or ‘False’

4.1.4.4.2 Attribute grammar fragment

```
<MnQ:1>
<MxQ:99999>
<MnV:5>
<MxV:999999.99>
<MP:>
<RcptNo:0>
<RV:True>
<SR:False>
<ST:>
<VO:True>
```

FUJ00001368

4.1.4.5 Inventory items

If the product is an inventory item (I), and a logistics inventory item (LINVI), the logistics accounting item (LACCI) and its corresponding accounting item (ACCI) is provided:

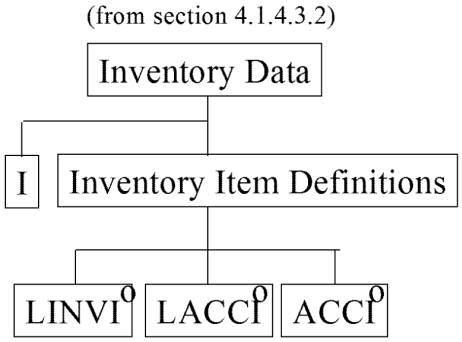


Figure 4-16 Persistent objects: Inventory Data

4.1.4.5.1 Data names and values

Name	*Value*
ACCI	Accounting item (optional). ACCI = product number If LACCI and LINVI both false, ACCI is null (<ACCI:>). If LACCI false and LINVI true, ACCI is the product number of the accounting item. If LACCI true and LINVI false, ACCI is the same as the product number.
I	Inventory item: 'True' or 'False'
LACCI	Logistics accounting item (optional): 'True' or 'False'
LINVI	Logistics inventory item (optional): 'True' or 'False'

4.1.4.5.2 Attribute grammar fragment

<I:False>

FUJ00001368

4.1.4.6 Sale values

Any product can be fixed price (FP) in which case it has a retail price (RP). It can be refundable (RF), can be staff (SDI) or customer discounted (CDI), and can be transacted in multiple units (MV).

The accounting sense (AS) is also defined, as is the effect it has when transacted in a session (SE).

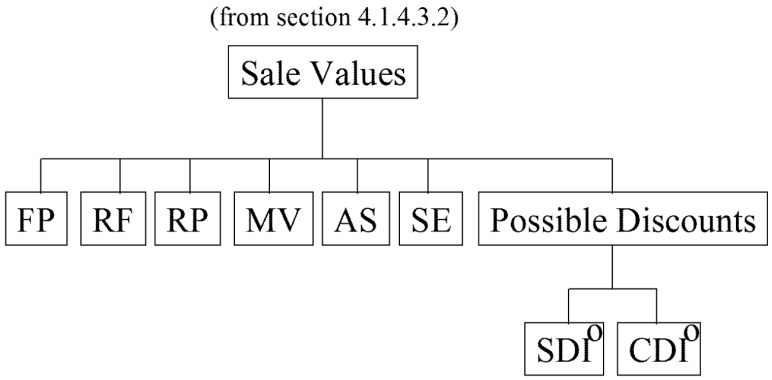


Figure 4-17 Persistent objects: sale values

4.1.4.6.1 Data names and values

Name	*Value*
AS	Adopt settlement sense. Indicates whether the value of this product when transacted adopts the sense (Positive / Negative) necessary to balance the current session. 'True' or 'False'.
CDI	Customer discounted (optional). 'True' or 'False'.
FP	Fixed price. 'True' or 'False'.
MV	Transacted in multiple units
RF	Refundable.
RP	Retail price.
SDI	Staff discounted (optional). 'True' or 'False'.
SE	Transacted in a session

4.1.4.6.2 Attribute grammar fragment

```
<AS:False>
<FP:True>
<MV:5>
<RF:False>
<RP:5>
<SE:In>
```

FUJ00001368

4.1.4.7 Data capture

If data has to be captured when the product is transacted, this data has to be defined item by item. For each item, the information displayed to the clerk is defined in terms of the prompt (P), the name of the item being captured (N), the caption used when it is captured (C), and a number associated with the order (O) in which the items are captured.

For validation purposes, the format (F) and data type (S), as well maximum (Max) and minimum (Min) values are defined. If the data fails validation, the message (VM) to be displayed is identified. If any action is allowed on the data (A) it is specified, as is the default (D) value presented to the clerk.

If options are to be displayed to the clerk, they are listed in terms of the text that identifies the option (Text) and the key value set of the option is selected (Key).

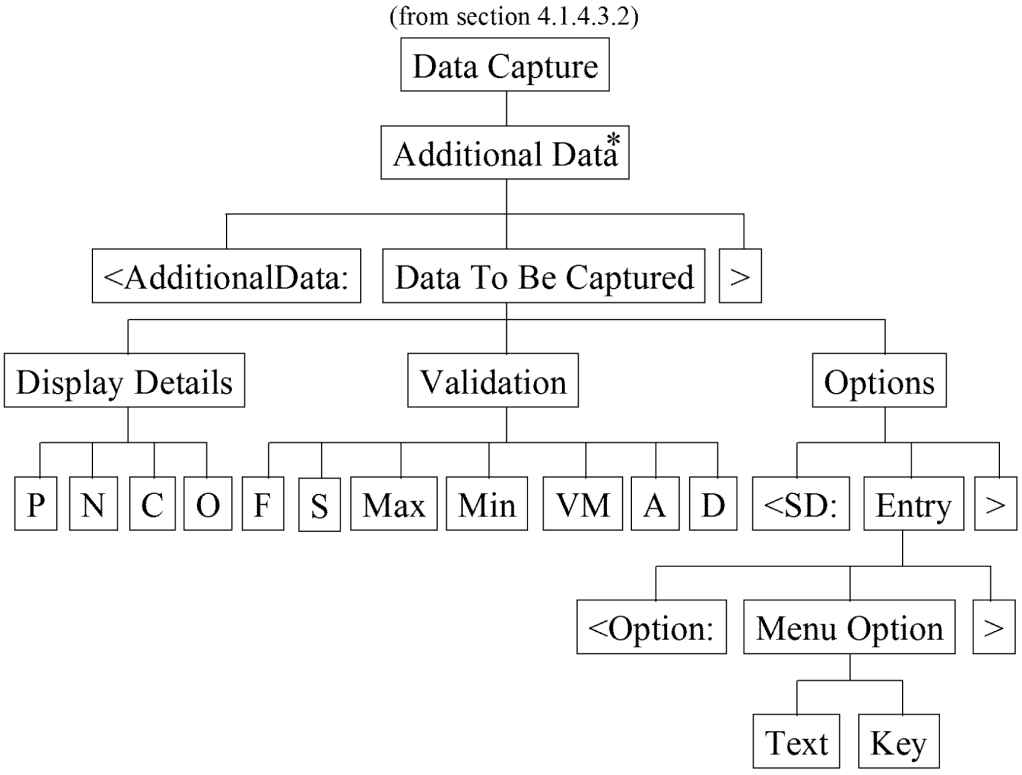


Figure 4-18 Persistent objects: data capture

FUJ00001368

4.1.4.7.1 Data names and values

Name	*Value*
A	Action on data
C	Caption
D	Default
F	Format
Key	Key to select option
Max	Maximum (Max) values
Min	Minimum (Min) values
N	Item name
O	Order
P	Prompt (maximum length: 30 characters) derived from Reference Data
S	Data type
Text	Text to display option
VM	Validation message

4.1.4.7.2 Attribute grammar fragment

```
<AdditionalData:
    <P:A/C NO>
    <F:*>
    <D:>
    <Max:12>
    <Min:0>
    <VM:Error>
    <N:ACC_NO12>
    <C:ACC_NO12>
    <O:51>
    <A:>
    <S:Alphanumeric>
    <SD:>
>
```

FUJ00001368

4.1.4.8 Pre-conditions

If any pre-conditions apply to the product being transacted, the product number of the product (ProductNo) and the product on which it depends (PCProdNo) are defined together with the message to be displayed to the clerk if the condition is not met (Msg).

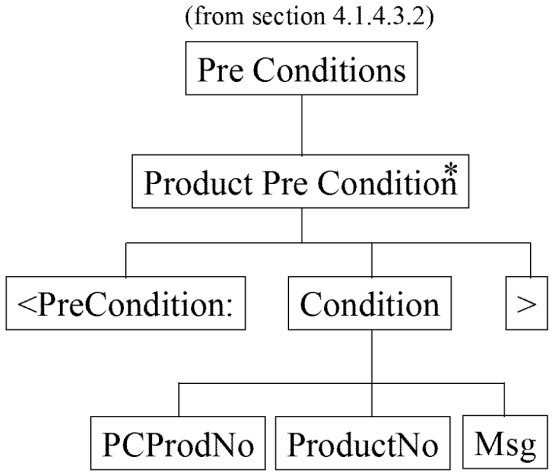


Figure 4-19 Persistent objects: product pre-conditions

4.1.4.8.1 Data names and values

Name	*Value*
Msg	Message to clerk
PCProdNo	Number of product on which current product depends
ProductNo	Product number

4.1.4.8.2 Attribute grammar fragment

<PCProdNo:2220>
<ProductNo:672>
<Msg:Transaction cannot be completed. Associated product has not been transacted.>

FUJ00001368

4.1.4.9 Accounting data

The Reference Data for a product identifies the primary mapping lines, one to five (L1, L2, L3, L4, L5). Levels 1 and 2 are owned by POCL. Levels 3, 4, and 5 are owned by ICL Pathway.

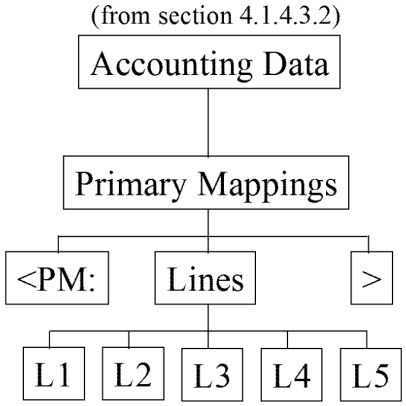


Figure 4-20 Persistent objects: primary mappings

4.1.4.9.1 Data names and values

All elements (L1, L2, L3, L4, L5) are present but may be null (<L1:>, for example).

Name	*Value*
L1	Primary mapping: line 1
L2	Primary mapping: line 2
L3	Primary mapping: line 3
L4	Primary mapping: line 4
L5	Primary mapping: line 5

4.1.4.9.2 Attribute grammar fragment

```
<PM:
    <L1:1702>
    <L2:2055>
    <L3:3007>
    <L4:3008>
    <L5:3017>
>
```

FUJ00001368

4.1.4.10 Cash Account information

The Cash Account record provides the details of the secondary mapping that must be recorded when the product is transacted. Secondary mappings are used to track transactions in modes other than Serve Customer, such as transfers and remittances.

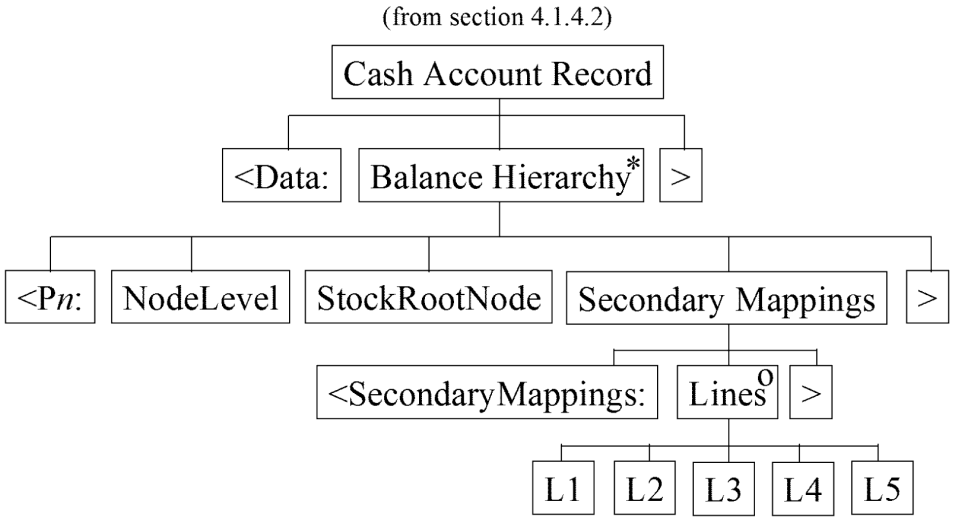


Figure 4-21 Cash Account: Secondary mappings

4.1.4.10.1 Data names and values

All elements (L1, L2, L3, L4, L5) are present but may be null (<L1:>, for example).

Name	*Value*
L1	Secondary mapping: line 1 (always present, but may be null)
L2	Secondary mapping: line 2 (always present, but may be null)
L3	Secondary mapping: line 3 (always present, but may be null)
L4	Secondary mapping: line 4 (always present, but may be null)
L5	Secondary mapping: line 5 (always present, but may be null)
n	Product number.
NodeLevel	Stock root node level
StockRootNode	Top-level node for Value Stock accumulation.

FUJ00001368

4.1.4.10.2 Attribute grammar fragment

```
<SecondaryMappings:
    <L1:>
    <L2:3035>
    <L3:3028>
    <L4:3027>
    <L5:3017>
>
```

FUJ00001368

4.1.4.11 Cash Account: office object

For the Cash Account itself, the Office level CAP record provides details of the start and end date of the current Cash Account Period for the office and the previous and next CAP.

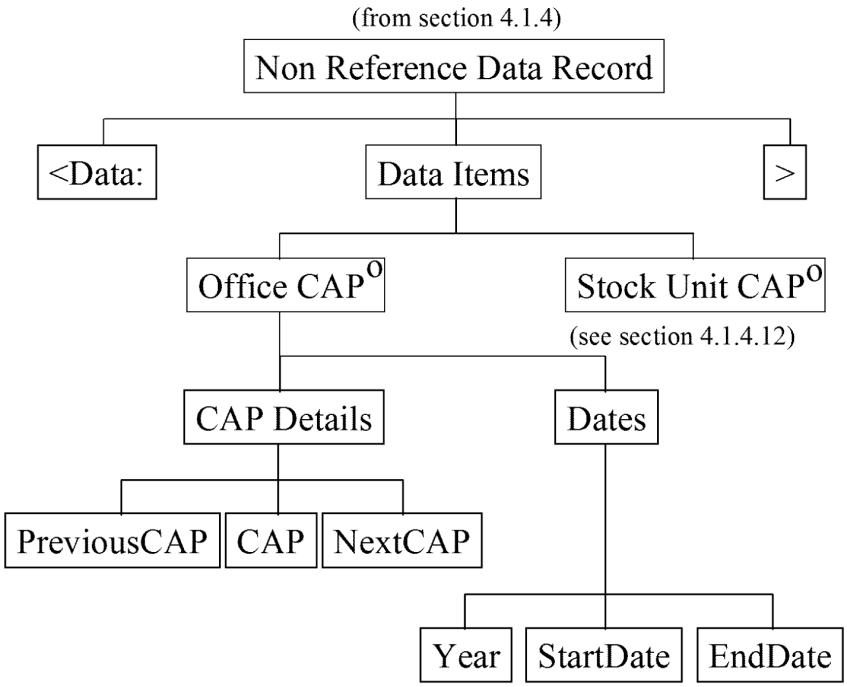


Figure 4-22 Office level Cash Account: Cash Account Period

4.1.4.11.1 Data names and values

Name	*Value*
CAP	Current Accounting Period
EndDate	Current Cash Account Period end date
NextCAP	Next CAP
PreviousCAP	Previous CAP
StartDate	Current Cash Account Period start date
Year	The year of the CAP

FUJ00001368

4.1.4.11.2 Attribute grammar fragment

```
<PreviousCAP:1>
<CAP:2>
<NextCAP:3>
<StartDate:06/01/2000>
<EndDate:12/01/2000>
<Year:2000>
```

4.1.4.12 Stock unit object

The equivalent record for each stock unit provides details of the Cash Account Period for that stock unit as well as identifying whether it is shared or locked, and if locked, by whom.

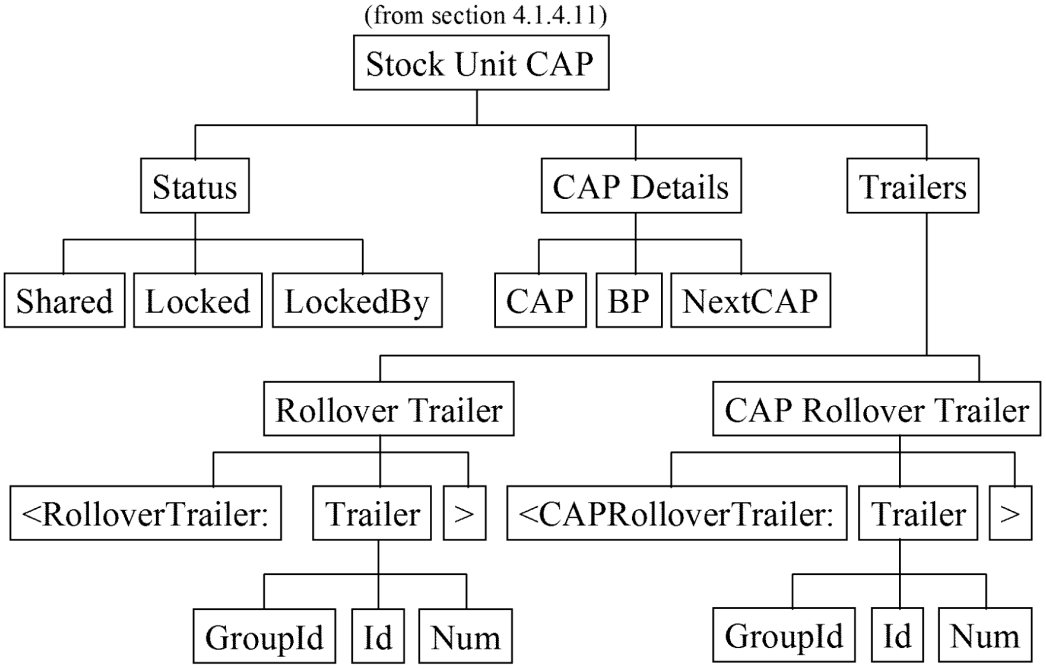


Figure 4-23 Cash Account: stock unit record

4.1.4.12.1 Data names and values

Name	*Value*
BP	Balance Period number
CAP	Current Accounting Period number
GroupId (CAPRolloverTrailer)	FAD code (without the check digit)
GroupId (Rollover)	FAD code (without the check digit)
Id (CAPRolloverTrailer)	Counter node

FUJ00001368

Id (Rollover)	Counter node
Locked	Stock Unit locked. ‘True’ or ‘False’
LockedBy	Locked by user: User ID
NextCAP	Next CAP
Num (CAPRolloverTrailer)	Sequence number of message
Num (Rollover)	Sequence number of message
Shared	Stock unit shared. ‘True’ or ‘False’

4.1.4.12.2 Attribute grammar fragment

```
<Data:
    <Shared:True>
    <Locked:False>
    <LockedBy:>
    <CAP:11>
    <BP:1>
    <NextCAP:12>
    <RolloverTrailer:
        <GroupId:666666>
        <Id:1>
        <Num:14504>
    >
    <CAPRolloverTrailer:
        <GroupId:666666>
        <Id:1>
        <Num:14504>
    >
>
```

FUJ00001368

4.1.4.13 Example product persistent object

```
<Message:
    <GroupId:4038>                          section 4.1.4.1
    <Id:1>                                  section 4.1.4.1
    <Num:18300>                             section 4.1.4.1
    <Date:23-Nov-1999>                      section 4.1.4.1
    <Time:15:36:00>                     section 4.1.4.1
    <Expiry:34>                         section 4.1.4.1
    <TranStartNum:18145>
    <Collection:_EPOSSProducts>             section 4.1.4.1
    <ObjectName:622_11>                     section 4.1.4.1
    <StartDate:01-Jan-1996 00:00:01>        section 4.1.4.1
    <EndDate:>                              section 4.1.4.1
    <RData:
        <Data:
            <PG:21>                         section 4.1.4.3
            <PN:622>                        section 4.1.4.3
            <SN:£5 Stamp>                   section 4.1.4.3
            <LN:£5 Stamp>                   section 4.1.4.3
            <RN:£5 stmp>                    section 4.1.4.3
            <V:11>                          section 4.1.4.3
            <SI:>                           section 4.1.4.3
            <MnV:5>                         section 4.1.4.4
            <MxV:999999.99>                 section 4.1.4.4
            <MnQ:1>                         section 4.1.4.4
            <MxQ:99999>                     section 4.1.4.4
            <SR:False>                      section 4.1.4.4
            <RcptNo:0>                      section 4.1.4.4
            <MP:>                           section 4.1.4.4
            <RV:True>                       section 4.1.4.4
            <VO:True>                       section 4.1.4.4
            <ST:>                           section 4.1.4.4
            <I:False>                       section 4.1.4.5
            <FP:True>                       section 4.1.4.6
            <RF:False>                      section 4.1.4.6
            <RP:5>                          section 4.1.4.6
            <MV:5>                          section 4.1.4.6
            <AS:False>                      section 4.1.4.6
            <SE:In>                     section 4.1.4.6
            <PM:                            section 4.1.4.9
                <L1:1702>                   section 4.1.4.9
                <L2:2055>                   section 4.1.4.9
                <L3:3007>                   section 4.1.4.9
                <L4:3008>                   section 4.1.4.9
                <L5:3017>                   section 4.1.4.9
            >
        >
    >
    <Version:1>                             section 4.1.4.1
    <CRC:9990C3AC>                          section 4.1.4
>
```

Figure 4-24 Attribute grammar for a product

FUJ00001368

4.1.5 Tokens

Tokens are a distinct subset of persistent objects and are described in this section.

There are currently three types of token:

- Bar-coded documents
- Magnetic stripe swipe cards used for automated payment
- Smart cards

For each of these token types, a different subset of information is present on the token. The data that is present on each token type and a list of the data specified in Reference Data for each token type are described in sections 4.1.5.1 – 4.1.5.4.

The definitions of tokens, as given in the collection established for each application, are used by their respective applications at start up each time the Desktop is reloaded to register with the Peripheral Server the tokens in which they have an interest. If a smart card is used, the Smart Card Manager is called. Unrecognised tokens are ignored.

Bar-codes can normally be scanned at any point within the Serve Customer hierarchy. The Peripheral Broker uses a Validation Object to check the definitions of the formats held in Reference Data to determine which application is interested in the bar-code. Failure to recognise a bar-code correctly can cause the wrong application to be invoked. This can occur if the definition of the characteristics of the bar-code are such that the definitions could apply to the bar-code produced for more than one application. Bar-codes, which are numeric, are normally identified by a combination of the following characteristics:

- Total number of digits that form the bar-code.
- Known values for specific digits within the bar-code.

If it is not possible to define the bar-code uniquely using these recognition parameters, as is the case for LFS, the application has to set the mode context prior to the bar-code being read in order to ensure that no other application can receive the token impulse.

For each token type, Reference Data defines:

- How the token is identified by its token identity (TID) and optional token name (TN).
- The data format on that token.
- Interface definitions to be presented to the application when the token is invoked.
- Validation that has to be applied to that token.

FUJ00001368

- The way the token data is sliced to allow identification of data and its validation.
- The modes in which the token may be used.

The reference data definitions received via POCL Reference Data at present define the bar-codes, magnetic swipe cards and smart cards that are needed to support Automated Payments via APS, Benefit Payments via OBCS and EPOSS functionality via EPOSS and LFS. If different tokens are required, the POCL Reference Data interface will have to be updated.

For further information on the technology used in magnetic cards and bar-codes please consult the appropriate *POCL Token Technology Specification*. These specifications define transaction data, the way the token is processed by the clerk and the use of check digit algorithms.

The examples used in the following sections are APS transactions, since they have the most complex set of attributes.

FUJ00001368

4.1.5.1 Generic token data

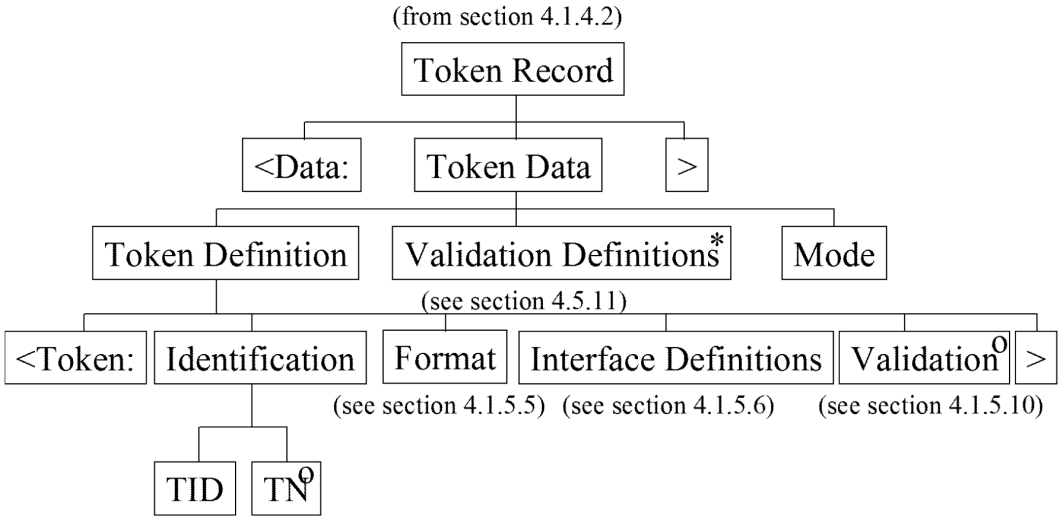


Figure 4-25 Persistent objects: token record

4.1.5.1.1 Data names and values

Name	*Value*
Mode	Transaction mode
TID	Token identity
TN	Token name (optional)

4.1.5.1.2 Attribute grammar fragment

```
<Mode:SC>
<TID:SPM1>
<TN:SPM1 Token>
```

FUJ00001368

4.1.5.2 Bar-coded token

4.1.5.2.1 Bar-coded token format

Numeric bar-coded tokens (currently OBCS, APS and LFS) contain the following information:

Field	*Value*
Client Reference Identifier	The first 6 to 12 characters of the bar-code. Uniquely identifies the Reference Data that defines the position of the other fields in the bar-code
Identification Number	Identifies the bar-code. Start position and length are defined in Reference Data. Maximum length is 24 characters Usually consists of Client Reference Identifier, CRN and optionally Service Code. The final digit of the Identification Number is a Luhn Check Digit, which is calculated over the whole of the Identification Number, using the algorithm defined in *ISO 7812 Identification cards - Numbering system and registration procedure for issuer identifiers*
Customer Reference Number (Optional)	Allocated by the Client to identify its customer. Start position and length are defined in Reference Data. Maximum length is 20 characters.
CRN Check Digit (Optional)	The CRN may or may not have a check digit(s) associated with it (they may or may not be part of the CRN). The start position, length and calculation algorithm are determined from Reference Data. Maximum length is 2 characters.
Service Code	Identifies the service for the Client. Used to determine Reference Data for the client/service. Start position and length are defined in Reference Data. Maximum length is 4 characters.
Amount (Optional)	The bar-code may contain an amount in pence. The start position is determined from Reference Data. Maximum length 9 characters. If Amount is invalid, the field is ignored during processing.
Amount Length (Optional)	The bar-code may contain the length of the Amount field.

FUJ00001368

4.1.5.2.2 Bar-coded token Reference Data

Service Reference Data

This data defines the processing parameters for each service of the client.

Bar-code Reference Data

This data define the content of each bar-code supported.

Customer Reference Check Digits

Most Customer References have or include check digits. The calculation algorithms for these check digits used for bar-codes are the same as those for magnetic cards. The parameters to determine which algorithm to use are held in the bar-code Reference Data for the Client Reference Identifier and Service Code.

4.1.5.3 Magnetic stripe card

4.1.5.3.1 Token format

Cards must conform to the magnetic stripe requirements for ID-1 cards as given in:

- *ISO 7813 Identification cards - Financial transaction cards*
- *ISO 7811 Identification cards - Recording technique:*

  7811/2 Part 2: Magnetic stripe

  7811/4 Part 4: Location of read-only magnetic tracks - Tracks 1 & 2

4.1.5.3.2 POCL Reference Data

Service data

This consists of the data to set up a terminal for an individual client, defining the processing parameters for each service of the client.

Card data

Card Reference Data consists of the parameters that define the content of each magnetic card supported.

Client check digit data

The type and method of check digit calculation to be used for a given card is determined from the magnetic card Reference Data for the Card ID and Service Code.

FUJ00001368

4.1.5.4 Smart card

4.1.5.4.1 Token format

The format of each token is different for each smart card application. Unlike other applications, data read from the smart card reader is handled not by the Riposte Desktop and the Peripheral Broker, but directly by the Smart Card Manager, SmartMan. SmartMan identifies the correct Card DLL for the inserted card and notifies any interested application that an impulse has been received.

4.1.5.4.2 POCL Reference Data

Persistent objects for each smart card application are held in Reference Data. More details of smart card applications and their architecture and interfaces are given in section 5.2.4 and Appendix A, *SmartMan Interfaces*.

FUJ00001368

4.1.5.5 Token format

The format of magnetic card and bar-code tokens is defined in terms of the length of fields (for magnetic cards and bar-codes), the security data on the token, the tracks that are used (for magnetic cards) and any definition of composite fields that need to be created from individual data items on the token. For smart cards, the name of the DLL to be called and the cash limit are specified.

Required length (Len) of the token, is specified if it is fixed length, or the minimum (MinLen) and maximum (MaxLen) lengths, if not. The length of the amount field (AMTLength) is also defined.

For smart cards, the definitions needed to complete the security checks are given. For lower level security requirements, the POCL-supplied check digit routine (CDig) is defined.

For card tokens, the track to be selected (TrkSel) is given if appropriate.

The composite fields definition identifies the elements to be included (Element).

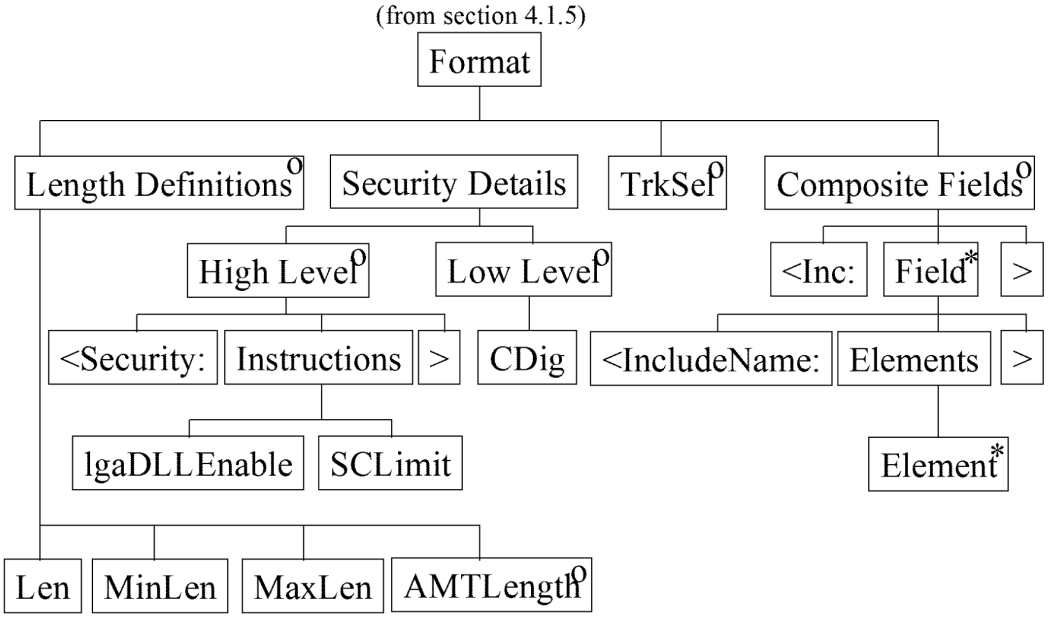


Figure 4-26 Persistent objects: Token Format

FUJ00001368

4.1.5.5.1 Data names and values

Name	*Value*
AMTLength	Amount field length (optional). If present it may contain one of the following values: A value to indicate that there is no Amount field present A value to indicate that the Amount field is of variable length The length of the Amount field
CDig	Check digit routine
Element	Elements to be included
Len	Length of the token
IgaDLLEnable	The DLL that is called to enable a smart card
MaxLen	Maximum length
MinLen	Minimum length
SCLimit	Session cash limit
TrkSel	Track

4.1.5.5.2 Attribute grammar fragment

Bar-code

<Len:0>
<MinLen:0>
<MaxLen:50>
<AMTLength:8>
<CDig:15>

Magnetic card

<Len:0>
<MinLen:19>
<MaxLen:37>
<TrkSel:2>

Smart card

<Security:
    <IgaDLLEnable:1>
    <SCLimit:1>
>

FUJ00001368

4.1.5.6 Interface definition

The interface definition identifies the application (*Application Name* and PropApp) to be called when the token is read, and the command to be passed to that application when invoked (Cmd). If there is a product associated with the token, it is specified (ProductNo).

Any data that is pre-defined for that token and needs to be passed to the application is also defined, as are the details needed to decode any check digits. If use of the token results in a transaction on the desktop stack, the definition of the button is given.

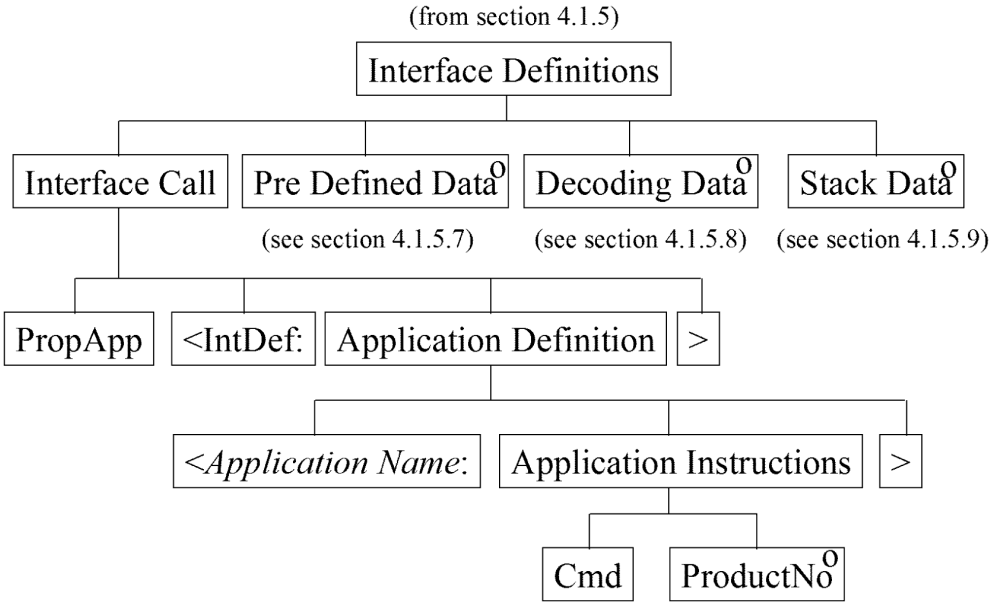


Figure 4-27 Persistent objects: Interface Definition

4.1.5.6.1 Data names and values

Name	*Vakue*
Cmd	The command to be passed to that application
ProductNo	Product associated with the token (optional)
PropApp	The name by which the application associated with the CardDLL is registered with the Riposte Desktop

4.1.5.6.2 Attribute grammar fragment

<IntDef:

FUJ00001368


```
        <AppXYZ:
            <Cmd:SellProduct>
            <ProductNo:698>
        >
>
<PropApp:AppXYZ>
```

FUJ00001368

4.1.5.7 Pre-defined data

Pre-defined data is used to define the data passed to the application across the interface:

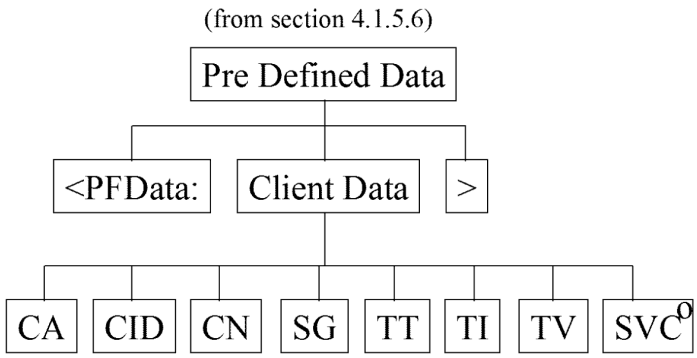


Figure 4-28 Persistent objects: Pre-defined Data

4.1.5.7.1 Data names and values

The following data is used to define **PFData** (pre-defined data):

Name	*Description*
CA	Client Account
CID	Client Identity
CN	Client Name
SG	Service Group
SVC	Service Code (optional)
TI	Token Identity
TT	Token Type
TV	Token Version

4.1.5.7.2 Attribute grammar fragment

```
<PFData:
    <CA:5555>
    <CID:88>
    <CN:Any Ltd>
    <SG:BD>
    <SVC:8>
    <TT:MC>
    <TI:TOKEN777>
    <TV:1>
>
```

FUJ00001368

4.1.5.8 Decoding check digits

To allow the application to decode check digits, the definition of the check digit array (CDs), the divisor (Div), the method of addition (Flg) and the weights array (W) are defined.

The minimum (MinLen) and maximum (MaxLen) of the item are given, as is the type of addition (AddType).

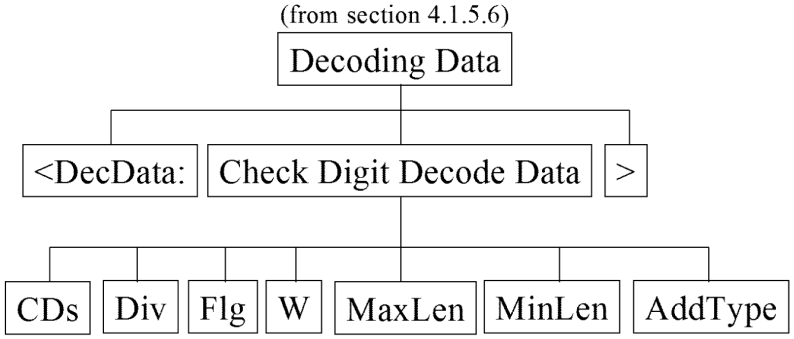


Figure 4-29 Persistent objects: Decoding Data

To allow the application to decode check digits, the definition of the check digit array (CDs), the divisor (Div), the method of addition (Flg) and the weights array (W) are defined.

The minimum (MinLen) and maximum (MaxLen) length of the item are given, as is the type of addition (AddType).

4.1.5.8.1 Data names and values

The following data is to define **DecData** (decoding data):

Name	*Description*
AddType	Type of addition: sum of products, or sum of digits of products
CDs	The check digit array
Div	Divisor
Flg	Method of addition: weighted or non-weighted.
MaxLen	Maximum length
MinLen	Minimum length
W	Weights array

FUJ00001368

4.1.5.8.2 Attribute grammar fragment

```
<DecData:
    <CDs:0,1,9,6,7,8,5,4,3,7,1,0,>
    <Div:11>
    <Flg:1>
    <W:0,0,0,0,0,0,0,0,0,7,8,4,6,3,5,2,1,0,0,0>
    <MaxLen:19>
    <MinLen:19>
    <AddType:1>
>
```

FUJ00001368

4.1.5.9 Stack items

For tokens that result in items being displayed on the desktop stack, the definition for the button includes the picture file from which the icon is extracted (StackPicFile) and the picture itself (StackPic). The caption on the button (StackCaption) is also defined.

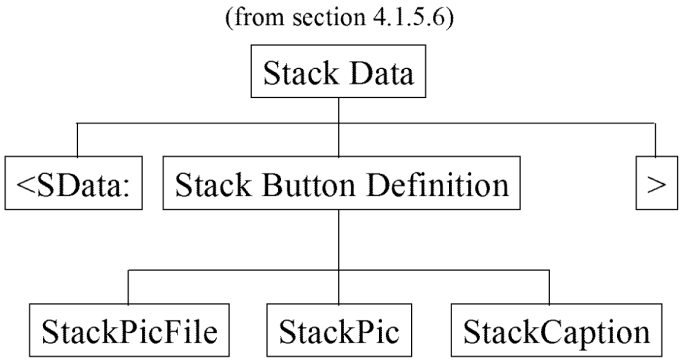


Figure 4-30 Persistent objects: Stack Item

4.1.5.9.1 Data names and values

The following data used to define **SData** (stack data):

Name	*Description*
StackCaption	Caption on the button
StackPic	Picture reference
StackPicFile	Picture file for the icon

4.1.5.9.2 Attribute grammar fragment

```
<SData:
    <StackPicFile:ANYicons>
    <StackPic:11>
    <StackCaption:Anyname>
>
```

FUJ00001368

4.1.5.10 Validation

The validation application to be called when the token is read is defined in terms of its name (ValApp), the command to be passed to it (ValCmd) and the name of the validation to be completed (ValName).

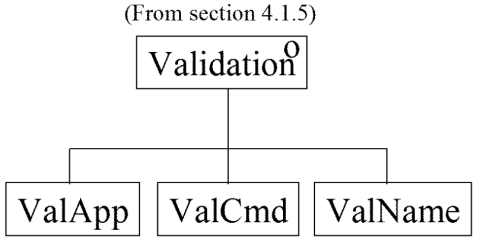


Figure 4-31 Persistent objects: Validation

4.1.5.10.1 Data names and values

The following data used to define the application to validate token data, its input and the validation to be performed:

Name	*Description*
ValApp	Validation application name
ValCmd	Command to pass to validation application
ValName	Name of the validation to be completed

4.1.5.10.2 Attribute grammar fragment

```
<ValApp:>
<ValCmd:>
<ValName:>
```

4.1.5.11 Validation definitions

The validation to be completed on each element of the data on a token is defined in terms if its element identity (EID), the name of the element (Name), given that it starts (Start) at a particular position and has a specific length (Len). The type of the data element (Pic) and the token identity to which it belongs (TID) are included, as is the start position (Start), type (Type) and length of the field to which it is applied (Length) and length of the check digit (Len).

FUJ00001368

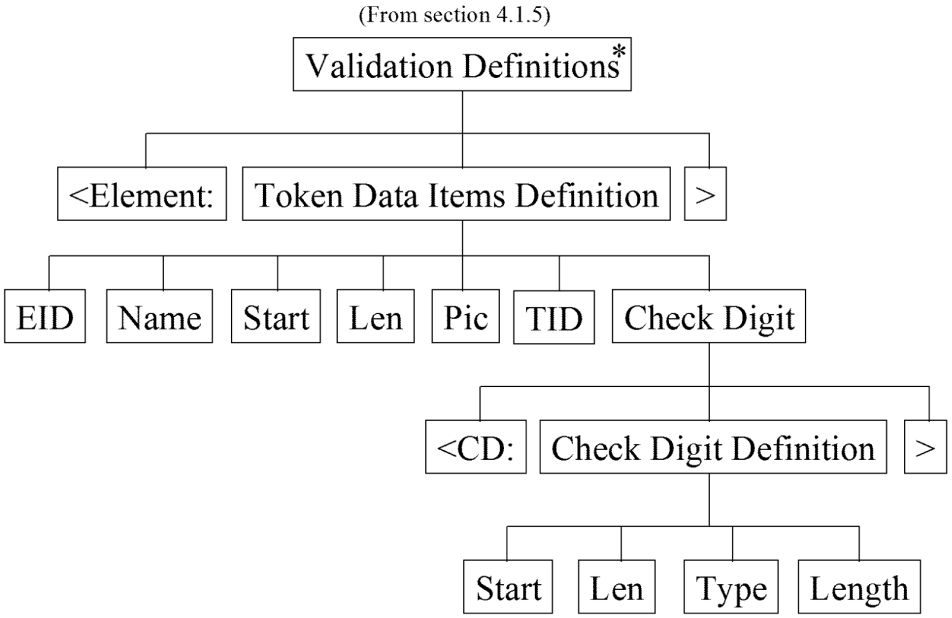


Figure 4-32 Persistent objects: Validation Definitions

4.1.5.11.1 Data names and values

Data for specifying **Element** (token data items) to be validated:

Name	*Description*
EID (Element)	Data element identity
Len (Element)	Data element length
Name (Element)	Data element name
Pic (Element)	Data element type
Start (Element)	Data element start
TID (Element)	Token identity

Data for specifying **CD** (check digits):

Name	*Description*
Len (CD)	Length of check digit
Length (CD)	Length of field to which it is applied
Start (CD)	Check digit start position
Type (CD)	Check digit type

FUJ00001368

4.1.5.11.2 Attribute grammar fragment

```
<Element:
    <EID:1>
    <Name:CIR>
    <Start:1>
    <Len:8>
    <Pic:"98341040">
    <TID:777>
    <CD:>
>
```

FUJ00001368

4.1.5.12 Example token definition: magnetic card

```
<Collection:TOKENXYZ>                                   section 4.1.4.1
<ObjectName:TOKEN777>                                       section 4.1.4.1
<StartDate:01-Jan-1996 00:00:00>                            section 4.1.4.1
<EndDate:>                                              section 4.1.4.1
<RData:                                                     section 4.1.4.2
    <Data:                                                  section 4.1.4.2
        <Token:                                             section 4.1.5
            <TID:777>                                       section 4.1.5.1
            <Len:0>                                     section 4.1.5.5
            <MinLen:19>                                 section 4.1.5.5
            <MaxLen:37>                                     section 4.1.5.5
            <TrkSel:2>                                      section 4.1.5.5
            <CDig:13>                                       section 4.1.5.5
            <TN:EXAMPLE1>                                   section 4.1.5.1
            <IntDef:                                        section 4.1.5.6
                <AppX:                                      section 4.1.5.6
                    <Cmd:SellProduct>                       section 4.1.5.6
                    <ProductNo:666>                         section 4.1.5.6
                >
            >
            <PFData:                                        section 4.1.5.7
                <CA:5555>                                   section 4.1.5.7
                <CN:Any Ltd>                                section 4.1.5.7
                <CID:88>                                    section 4.1.5.7
                <SG:BD>                                     section 4.1.5.7
                <TT:MC>                                     section 4.1.5.7
                <TI:TOKEN777>                           section 4.1.5.7
                <TV:1>                                      section 4.1.5.7
            >
            <DecData:                                       section 4.1.5.8
                <CDs:0,1,9,6,7,8,5,4,3,7,1,0,>              section 4.1.5.8
                <Div:11>                                    section 4.1.5.8
                <Flg:1>                                     section 4.1.5.8
                <W:0,0,0,0,0,0,0,0,0,7,8,4,6,3,5,2,1,0,0,0> section 4.1.5.8
                <MaxLen:19>                                 section 4.1.5.8
                <MinLen:19>                             section 4.1.5.8
                <AddType:1>                                 section 4.1.5.8
            >
            <SData:                                         section 4.1.5.9
                <StackPicFile:ANYicons>                     section 4.1.5.9
                <StackPic:11>                               section 4.1.5.9
                <StackCaption:Anyname>                      section 4.1.5.9
            >
            <ValApp:>                                       section 4.1.5.10
            <ValCmd:>                                       section 4.1.5.10
            <ValName:>                                      section 4.1.5.10
            <PropApp:XYZ>                                   section 4.1.5.6.1
        >
        <Element:                                           section 4.1.5.11
            <EID:1>                                     section 4.1.5.11
            <Name:CIR>                                  section 4.1.5.11
            <Start:1>                                       section 4.1.5.11
            <Len:8>                                     section 4.1.5.11
            <Pic:"98341040">                                section 4.1.5.11
            <TID:777>                                       section 4.1.5.11
            <CD:>                                           section 4.1.5.11
        >
        <Element:                                       section 4.1.5.11
            <EID:2>                                 section 4.1.5.11
            <Name:SVC>                                  section 4.1.5.11
```

FUJ00001368


```
                <Start:9>                       section 4.1.5.11
                <Len:1>                       section 4.1.5.11
                <Pic:"0">                       section 4.1.5.11
                <TID:777>                       section 4.1.5.11
                <CD:>                       section 4.1.5.11
            >
            <Mode:SC>                       section 4.1.5
        >
    >
```

FUJ00001368

4.2 Settlement

This section describes the following concepts:

- Settling a customer session
- Committing a customer session
- Refunds and reversals
- Swapping between customer sessions
- Transferring a customer session
- Sending settlement data to Clients

4.2.1 Settling a customer session

A customer session is settled when Methods of Payment are accepted and balance has been achieved. The Riposte Retail Broker Session Manager handles the settlement of transactions and is activated when the clerk touches the 'Finish' button. It is also invoked if a forced logout occurs (see section 4.2.2.2 for a description of the forced logout process).

A user application can be called by the settlement process. EPOSS uses this call to present the Method of Payment (MoP) menu and to ensure that the net value of the session is eventually settled to zero net value.

4.2.1.1 Methods of Payment

In general, the outstanding customer session balance can be settled by a single Method of Payment or a combination of two or more MoP (part cash and part cheque, for example).

The methods of payment for settlement are dependent upon the actual transactions in the session log. When the settlement function is invoked, the system presents the user with a list of available MoPs. Product pre-conditions determine which MoPs can be used to settle particular transactions (see section 4.1.1.3).

4.2.2 Committing a customer session

4.2.2.1 Session completion

When a session is completed and balance has been achieved, the details of the session are committed, a number of records are written to the message store and a receipt may be printed. Both sides of the transaction are always written to the message store: the sale of the product itself and the amount and type of the payment (unless the balance of the transaction was zero before settlement).

The number of records written depends upon the number of transactions performed within the session, the type of products being transacted and

FUJ00001368

the methods of payment being used. Except in the case of a zero-value transaction, which writes a single record to the message store, the sale of a product and the associated payment from the customer are recorded as two or more separate records. The example in Figure 4-33 shows three records written to the message store for the sale of a TV licence, settled with saving stamps and a cheque.

```
<Message:
    <GroupId:123456>
    <Id:1>
    <Num:43181>
    <Date:13-Dec-1999>
    <Time:08:13:27>

    <User:AMGR01>
    <Expiry:35>
    <TranStartNum:43181>
    <TxnData:
        <SessionId:44-123456-1-43180-1>
        <TxnId:44-123456-1-43180-2>
        <Container:A02>
        <Start:
            <Date:13-Dec-1999>
            <Time:08:13:02>
            <TF:9>
        >
        <End:
            <Date:13-Dec-1999>
            <Time:08:13:03>
            <TF:1>>
            <Mode:SC>
        >
        <Application:EPOSSAppMain>
        <EPOSSTransaction:
            <ProductNo:2220>                    TV licence
            <Qty:1>
            <PVer:17>
            <SaleValue:101>                     Price £101
            <BlackBoxData:
                <M:SC>
                <S:1>
            >
            <TranType:S>
            <PM:
                <L1:1061>
                <L2:2032>
                <L3:3006>
                <L4:3013>
                <L5:3017>
            >
            <SM:>
            <Discounted:False>
        >
        <Credit:10100>
        <CRC:79665F1A>
    >
>
<Message:
    <GroupId:123456>
    <Id:1>
    <Num:43182>
```

FUJ00001368


```
        <Date:13-Dec-1999>
        <Time:08:13:27>
        <User:AMGR01>
        <Expiry:35>
        <TranStartNum:43181>
        <TxnData:
            <SessionId:44-123456-1-43180-1>
            <TxnId:44-123456-1-43180-3>
            <Container:A02>
            <Start:
                <Date:13-Dec-1999>
                <Time:08:13:10>
                <TF:7>
            >
            <End:
                <Date:13-Dec-1999>
                <Time:08:13:18>
                <TF:4>
            >
            <Mode:SC>
        >
        <Application:EPOSSAppMain>
        <EPOSSTransaction:
            <ProductNo:199>                          TV saving stamp
            <Qty:-1>
            <PVer:15>
            <SaleValue:-50>
            <BlackBoxData:
                <M:SC>
                <UnitPrice:50>
                <S:1>
            >
            <TranType:S>
            <PM:
                <L1:1956>
                <L2:535>
                <L3:3005>
                <L4:3016>
                <L5:3017>
            >
            <SM:>
            <Discounted:False>
        >
        <Debit:5000>                                 £50
        <CRC:CA5CBF37>
    >

    <Message:
        <GroupId:123456>
        <Id:1>
        <Num:43183>
        <Date:13-Dec-1999>
        <Time:08:13:27>
        <User:AMGR01>
        <Expiry:35>
        <TranStartNum:43181>
            <TxnData:
                <SessionId:44-123456-1-43180-1>
                <TxnId:44-123456-1-43180-4>
                <Container:A02>
                <Start:
                    <Date:13-Dec-1999>
```

FUJ00001368


```
                    <Time:08:13:27>
                    <TF:3>
                >
                <End:
                    <Date:13-Dec-1999>
                    <Time:08:13:27>
                    <TF:4>
                >
                <Mode:SC>
            >
            <Application:EPOSSAppMain>
            <EPOSSTransaction:<ProductNo:2>                Cheque
            <Qty:-1>
            <PVer:9>
            <SaleValue:-51>
            <BlackBoxData:
                <M:SC>
                <UnitPrice:51>                             £51
                <S:1>
            >
            <TranType:S>
            <PM:
                <L1:1002>
                <L2:1000>
                <L3:3003>
                <L4:3008>
                <L5:3017>
            >
            <SM:>
        >
        <Debit:5100>
        <CRC:EFA8DFE2>
>
```

Figure 4-33 Attribute grammar: product sale and settlement records

All transactions written via the Retail Broker after settlement have the same SessionId and are treated as an atomic unit for the purposes of committal and replication. This is true whether the session completes normally, involves a session transfer or is terminated by the system, but it is the responsibility of the application to deal with recovery situations.

4.2.2.2 Temporary lock and forced logout

If a terminal is to be unattended for a period of time, the clerk can use the Temporary Lock facility to prevent anyone else from using it. When the clerk returns, the password has to be re-entered.

A session is timed out after a period of inactivity (when there has been no input from any of the peripherals) of a pre-configured length. After this time, a user logon screen is displayed saying that the session is currently locked by that user and that the user's password needs to be entered to unlock the screen.

If there is a further period of inactivity after a session has timed out, the system forces a permanent logout. This results in the user being logged out, and the Riposte logo appearing in the centre of the screen. If there is a customer session in progress (including a suspended session), it is

FUJ00001368

committed against cash and written to the message store. The Method of Payment in this case is recorded as cash even if there is a product pre-condition (see section 4.1.1.3).

The system does not produce a receipt. When a user logs on again after a forced logout, the Reprint Receipt function can be used to produce a session receipt showing the transactions committed.

4.2.3 Reversals, refunds and voids

There are different rules and procedures for reversals, refunds and voids, according to the token type and application involved. The business rules, defined by the Client, must be interpreted within the application code. These rules include whether or not reversals have to take place on the same day as the original transaction (prior to End of Day), and whether or not they are permitted if the counter position being used is not communicating with the rest of the positions in the outlet.

4.2.3.1 Reversals

Reversals and refunds may be performed if POCL's accounting and business rules, on behalf of the Client, allow them.

There are two types of reversal:

- A linked reversal

  The user must enter the transaction session number to initiate the reversal. The existing transaction is not deleted from the message store but an additional compensating transaction is created. This is the recommended reversal method. In line with the business rules, a linked reversal is compulsory for some transactions. A transaction may be reversed once only, using the same stock unit and during the same balance period and CAP as the original transaction. Some transaction types (AP, for example) cannot be reversed once End of Day has been completed.

- An unlinked reversal (refund)

  An unlinked reversal is performed if a customer wishes to obtain a refund for a purchase for which they have no receipt, or if it is necessary to perform a non-specific reversal for balancing purposes. Unlinked reversals must be in the same CAP but may be in a later balance period than the transaction being reversed; they are accounting corrections and do not require the original transaction to be identified. For an unlinked reversal, a negative sale is transacted and written to the message store.

FUJ00001368

4.2.3.2 Voids

A transaction can be voided (cancelled) within a customer session provided that the customer session has not been committed and that the Reference Data supporting the product allows that product to be voided. (To void a transaction, the user selects the Bin option and then selects the transaction in the stack.) When a transaction is voided in this way, no entry for it is written to the message store.

Once a product defined in Reference Data as being non-voidable is added to the stack, the reversal function is the only method available to cancel the transaction. If the user tries to void a non-voidable product, the system displays a message saying that voiding is not allowed. AP transactions cannot be voided once an amount has been entered and a receipt has been printed. If, for example, a customer wants the money back, the transaction must be completed and then reversed.

4.2.4 Swapping between customer sessions

Riposte allows the clerk to start a new customer session before completing the existing session. The existing session becomes the swapped session and allows for such situations as the customer needing to leave the counter to fill up a form before completing a transaction. The clerk is enabled to serve the next customer and then to revert to the swapped session when it becomes possible to complete it. No more than two sessions can be open simultaneously. The clerk is not permitted to log out if there is an incomplete (swapped) session. When applicable, the system still invokes a terminal inactivity timeout (see section 4.2.2.2) and then a forced logout. The forced logout commits the suspended session.

Swapping a customer session is logically similar to transferring a session (see section 4.2.5). There may be business and technical reasons why both swapping a session and transferring a session may be prevented, as described in section 4.2.5. The prohibitions so far identified are concerned with data integrity, as in the case of AP smart cards, and with volumes of data, as in the case of the display of large pick lists.

For full details of session swapping, refer to Escher's *Riposte Desktop Session Management.*

4.2.5 Transferring a customer session

The user can transfer a current session to another counter position by logging on at another terminal. The session transfer function allows the user to log on at another terminal, without having to log off at a previous one. Any activities that are being carried out appear on the new system in exactly the same state as on the previous system.

During a transfer session, the Desktop (Riposte) transfers all the properties of the OCXs and its own state; the states of variables within the

FUJ00001368

applications also need to be transferred. Two types of data can be transferred during a session transfer:

- String
- Unindexed Visual Basic collection of strings

To support session transfer, an application obtains a handle to the **TRState** object. During session transfer, **TRState** calls each application on the transfer-from PC via the **SaveState** method.

For full details of session transfer, refer to Escher's *Riposte Desktop Session Management.*

Session mobility depends upon the business rules and may be locked for some sorts of transactions. For example, AP smart card transaction sessions cannot be transferred, since removing the smart card completes the transaction and inserting the smart card into another reader starts a new transaction.

Interfaces for locking and unlocking are as follows:

- The **LockUser** interface is used to set a lock with user name and an EventType of 'TransferLock'.
- The **UnlockUser** interface is used to clear a lock with user name and an EventType of 'Unlock'.
- The **CheckUserLock** interface is used to find the status of a lock with user name. Both the LockStatus (0 if unlocked, 1 if locked) and the actual EventType is returned.

4.2.6 Sending settlement data to Clients

Transaction level reports by stock unit may be printed on Clients' stationery using the counter printer or, if the physical size or layout of the stationery demands it, the A4 outlet printer. The reports conform to POCL's specification on behalf of the Client and are produced at stock unit and office level as required.

The agent layer of TMS transforms Riposte messages from the Desktop into records in the Client Interface table, which are then available to Client servers. For a general description of this interface, refer to *TMS Architecture Specification*. For details of the agent interface, refer to section 7.

4.3 Stock unit management

The sale of each retail product is recorded as a separate transaction against a specific user and stock unit. The context that provides this is established automatically at logon and requires no action by applications.

FUJ00001368

Stock units are units of accountability that may relate to one individual or may be shared among a number of clerks, and contain stock: fixed price stock items, customer- and Client-specific tokens, retail stock items, cash and transaction vouchers. Each outlet has at least one stock unit.

The number of stock units and their set-up is under the control of the manager of the outlet, as part of the office administration functions.

4.3.1 Individual and shared stock units

Stock units may be defined as individual or sharable. Individual stock units allow for a single user at any one time. Sharable stock units may be in use by more than one user at any one time. In addition, each user is restricted to using only one stock unit at a time.

Users are always attached to a stock unit. When they are not attached to a serving stock unit, they are attached to the default stock unit, which limits the functions available to them.

There is no detach function. The equivalent is to attach to the default stock unit.

4.3.2 Stock unit balancing

Stock units can be balanced, as often as is required, within a Cash Account Period. Balancing involves the declaration of stock and cash by denomination. An office balance is produced at the end of the Cash Account Period by amalgamating the stock unit balances. The system provides an office level snapshot, at any time, of transactions, cash and stock levels, across all stock units. The same facility is available to provide an instant snapshot of the transactions and contents of any single stock unit.

For more details of stock unit balancing, see section 4.5.

4.4 Reporting

The interfaces that are available for applications to produce receipts and reports are as follows:

- Receipts

  A receipt generator, available via Cashier, which is supplied with Escher's Design Studio product.

- Reports

  Visual Basic custom code should be used to produce reports to be printed on A4 office printers.

FUJ00001368

4.4.1 Receipts

Escher's Design Studio product provides a set of templates and add-ins for common receipt layout types that can be adapted to the needs of a particular application. The receipt formats can be saved to the message store, or locally to a file, using the Build menu in Design Studio. They are printed using peripherals accessible from a Riposte node. For details of the Receipt Layout Editor, refer to Escher's *Design Studio Receipt Layout Editor User's Guide*.

4.4.2 Reports

All the reports and receipts produced by the Horizon system are specified in *Horizon OPS Reports and Receipts*. For historical reasons EPOSS uses a dedicated report generator package, which is not available to other applications and which will not be used in future developments. Other applications use the generic facilities provided by Riposte's Peripheral Broker (see section 5.2 for a summary). Full details are contained in *Riposte Peripheral Broker API Reference Manual* in Riposte's documentation set.

4.4.3 Cash Account Report

The Cash Account report comprises a set of tables each listing the accumulated totals of a set of contributing product transactions. The totals are either value or quantity, or both, depending on the product.

The mechanism used to track and record the transaction data is Reference Data. Reference Data is organised into a hierarchy of nodes.

Each level is 'owned' by POCL or by ICL Pathway. Levels 1 and 2 belong to POCL, and 3, 4, 5 to ICL Pathway. However, changes to Level 2 may affect report scripts, so all such changes can only be made in consultation with ICL Pathway.

Primary Mappings (see section 4.1.4.9) define how the transactions are summarised in Serve Customer mode.

FUJ00001368

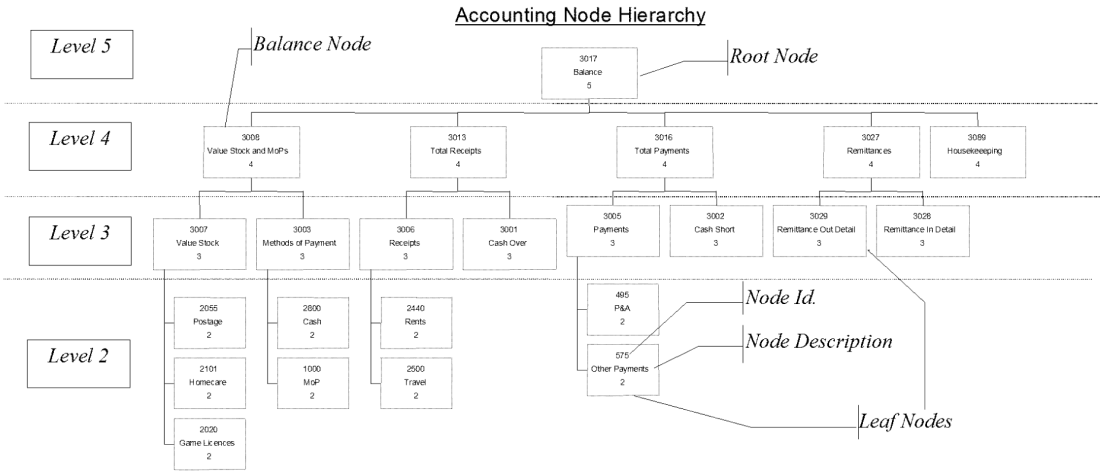


Figure 4-34 Account Node Hierarchy

Accumulating totals based on this view provides a product-based view.

To track transactions in other modes, such as transfers and remittances, there is a set of secondary mappings. In effect this supports the double-entry approach.

4.4.4 Cash Account Hierarchy

The Cash Account Hierarchy exists to collect the transactions correctly to compile the Cash Account Report.

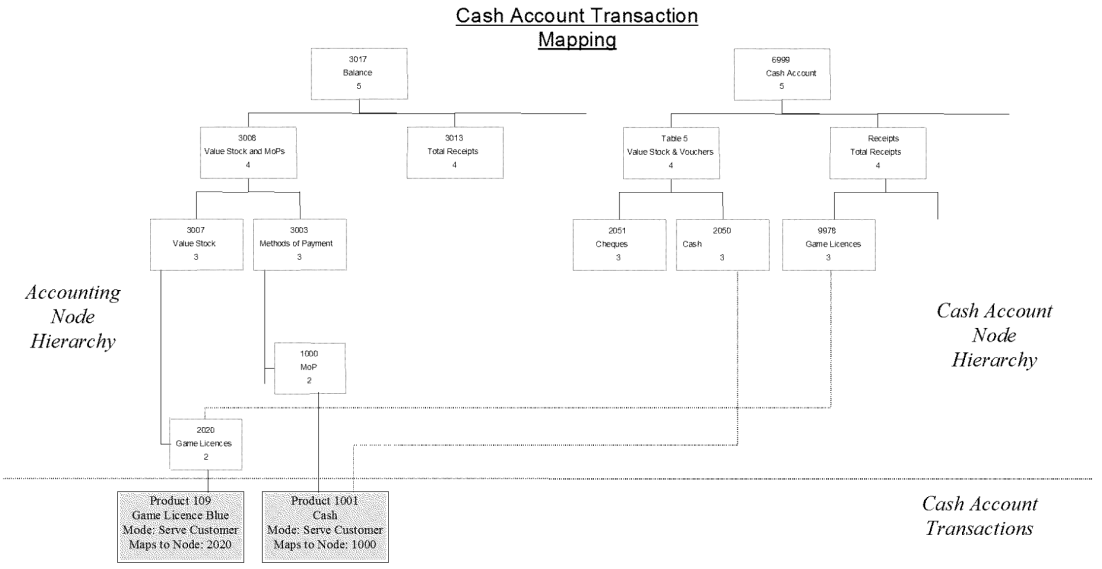


Figure 4-35 Cash Account Transaction mapping example

FUJ00001368

The Riposte Session Manager handles the transaction stack (as described in section 4.2.1). When the session is settled by the counter clerk, either by pressing 'Finish' and a suitable method of payment, or by swiping a card, the total set of transactions for that session is written to the message store as a single entity. This single entity provides the accumulation of transactions that are printed on a customer receipt, or session receipt.

The Cash Account Report is compiled after all value product transactions for each transaction mode for the period have been extracted from the message store. The sale of a blue game licence, shown in the diagram above as 'Serve Customer' mode, is summarised as part of value Stock and MoP under the accounting node hierarchy, both as a licence going out of the outlet, and cash coming into it. In the Cash Account hierarchy, the licence is reported as contributing to the Total Receipts table, and the cash as contributing to Value Stock and Vouchers: Table 5 in the Cash Account Report.

FUJ00001368

4.5 Stock unit balancing

Within EPOSS there are a several processes, invoked from the Riposte Desktop, that manage the administration, balancing, and reconciliation of stock units. These processes track the movement of stock into and out of the outlet (remittances) and the transfer of stock within the outlet.

Each clerk is attached to a stock unit, which must be balanced, according to the business rules, by the end of the Cash Account Period (CAP). All active stock units are balanced and rolled forward into the next Cash Account Period (CAP); any inactive stock units must be rolled forward into the next CAP. An inactive stock unit is reactivated and needs to be balanced if it has been revalued during the CAP.

The Cash Account Period can be extended if necessary, in which case the Cash Account Period becomes the week number to which it has been extended and will be out of step with current week number.

The sum of all the stock units' balances allows the outlet Cash Account to be derived, by working forward from the last balance. When the balance is taken and the outlet rolls into the next CAP, the Riposte message store is marked, and a clean start is made using the brought forward balances.

Horizon OPS Reports and Receipts gives details of the contents of the Cash Account Report and how it is derived.

FUJ00001368

5 Application Functions

The architecture of OPS and TMS and their relationship to counter applications are described in detail in section 3 of *OPS Architecture Specification.* A summary of the information is included in this section, together with a description of the following components:

- Peripheral Broker Interfaces
- Retail Server and Broker Interfaces
- Desktop Interfaces
- Riposte Functions

5.1 Architecture

Counter applications operate within the framework of Riposte and support EPOSS. They have access to various common Riposte functions for recording transaction details, settlement, user access control and session management.

Counter applications make use of Reference Data to control various detailed aspects of their operation. Reference Data is distributed from a central services Reference Data Management Centre (RDMC). The data is stored as persistent message data within the message stores on each counter PC at each outlet, where it is accessible to applications using the normal message store API.

FUJ00001368

Figure 5-1 shows the interfaces between counter applications and other functions and components of the OPS layer. Smart card applications have a different interface, which is described in section 5.2.4.

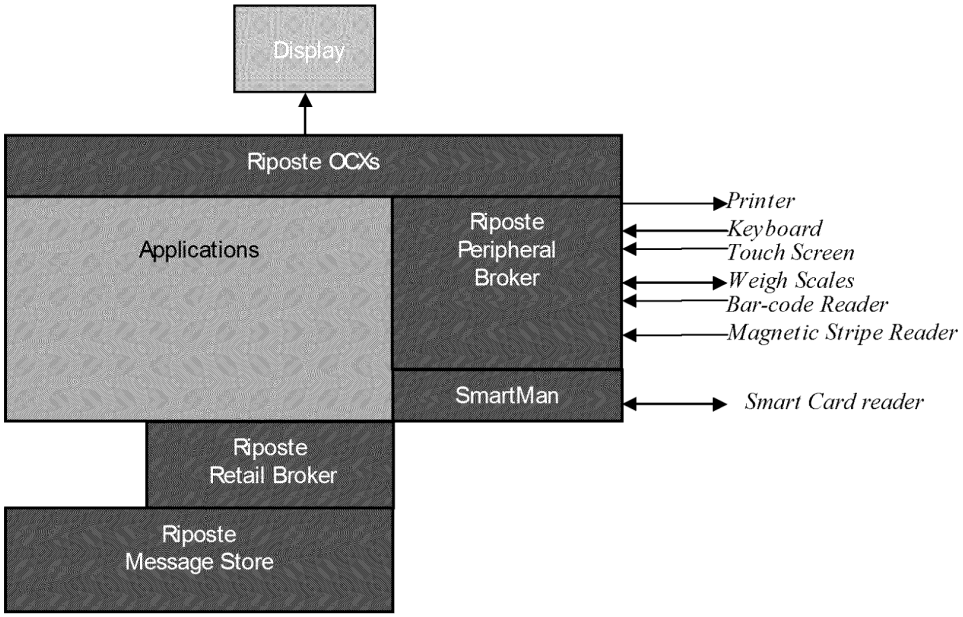


Figure 5-1 Counter Applications Environment

5.2 The Peripheral Broker

The Peripheral Broker (also known as the Peripheral Server) provides a common OLE interface to all Riposte applications. It supports the use of input and output devices and manages spooling and queuing. The Peripheral Broker hides the details of the interfaces to all peripherals connected to the counter PCs, and yet makes the facilities of these available to all terminals.

The peripherals supported include:

- Bar-code readers
- Magnetic card stripe reader
- Smart card reader/encoder (via the SmartMan interface: refer to section 5.2.4)
- Scales

  Scales are shared between terminals, although each has its own direct connection (for more details of the Scales interface, refer to *OPS Architecture Specification*).
- Keyboard

FUJ00001368

- Touch screen
- Counter printers (printing on A4 printers is controlled by standard NT services)

Desktop applications make use of a Peripheral Broker API to obtain input and direct output, without needing to know about the capabilities of individual devices. If changes are made to the configuration of devices supported by the Desktop, the Desktop must be restarted (see also section 6.3.4).

The overall architecture is shown in Figure 5-2.

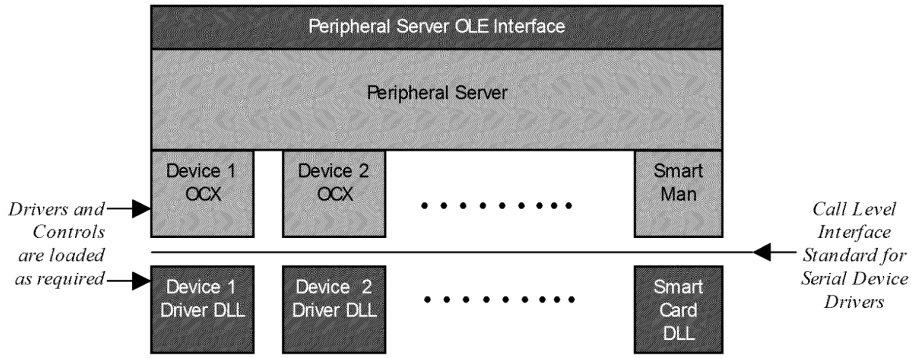


Figure 5-2 Peripheral Broker architecture

The Peripheral Broker supports the following features of the Riposte Desktop:

- An input event, such as a bar-code scan, can activate an application automatically.
- Riposte applications can share input and output devices transparently across the network.
- An input event can be sent to multiple applications simultaneously, allowing the first available operator to manage the transaction. For example, if two counters share a set of scales, the placement of a package on the scales can be handled by either counter position, depending on which is available.
- Output to printers can be precisely laid out and previewed on-line using an HTML browser.
- It is possible to queue output to A4 printers asynchronously, so that applications are available for continued use.

FUJ00001368

There are primarily two functions that the Peripheral Broker provides for user applications:

- It accepts and routes input events from a number of attached input devices.
- It processes data to be written to devices such as printers or smart cards.

The Peripheral Broker's functions are described in detail in Escher's *Riposte Peripheral Broker API Reference Manual*. The names of the methods described in this manual are, in most cases, the same as the Riposte 32 API function with the word 'Riposte' removed; exceptions are listed in the manual. This manual identifies those calls that have been superseded and gives guidance on the methods to use.

5.2.1 Peripheral Broker output

The following sequence of methods is used within applications to ensure that standard output jobs are handled correctly (for a description of the methods used, see Table 5-1):

1. **BeginJob** is called to create a job entry in the Peripheral Broker. This method returns the job ID that must be passed in all subsequent calls to the Peripheral Broker. The **BeginJob** method takes a number of parameters, including the Application Callback Object, the application's name, the job type (which is used to assign the correct device to the job), or the specific device name, and finally whether the job is to be synchronous or asynchronous.
2. For every line of data to be sent to the output device, the **WriteData** method of the Peripheral Broker is called, passing it the unique job ID returned from the **BeginJob** method.
3. When all the data has been sent to the output device using the **WriteData** method, the **EndJob** method is called, using the unique job ID returned from the **BeginJob** method.

5.2.1.1 Error handling

If an error occurs during processing of an output job, an application may be notified as follows:

- If the job is asynchronous, a call is made to the application's **DeviceErrorHandler** method. The call includes the error details.
- If the job is synchronous, the return value of each function call to the Peripheral Server must be checked for the value 'False'. If an error occurs, a call to the **ErrorDescription** property of the Peripheral Server returns the error that occurred.

FUJ00001368

If an error occurs, the current job can be cancelled using **CancelJob,** or started again at the exact point where the error occurred using **ResumeJob**. Table 5-1 gives details of these methods.

5.2.2 Peripheral Broker input

The Peripheral Broker monitors input devices for data and routes data to the correct application. When data is read from one of the input devices, the Peripheral Broker first passes that raw data to the Validation Object (see section 4.1.5.10), which matches it to a token definition that has been registered with it by the application. The Validation Object will then return the event name and the raw data in a parsed form to the Peripheral Broker (for further details of the Validation Object, refer to Escher's *Riposte Validation Object Developer's Guide*). The Peripheral Broker also searches its list of registered events for a match. Once found, the application associated with that event is sent the data via the application's **CallInterface**.

So that the Peripheral Broker can associate data with a particular application, the application must register information about the event as follows:

- Register the event's data and its format with the Validation Object.
- Register the event's name and pass its call-back object to the Peripheral Broker via the **RegisterEvents** method.

5.2.3 Peripheral Broker interfaces

This section briefly describes the programming interface of the Peripheral Broker DLL object. For full details of this interface, refer to Escher's documentation *Riposte Peripheral Broker API Reference Manual*.

The Peripheral Broker's methods and properties are accessed by using the **EGPeripheralBrokerLib** external type library interface. Object methods in this library accept as arguments both values, and attribute grammar messages supplied by applications.

The Peripheral Broker also implements the **EGBroker** interface, which is described in section 5.3.

The objects listed in Table 5-1 reside in the **EGPeripheralBrokerLib** type library.

FUJ00001368

EGPBrokerAdmin Administrative functions.	
Admin	Sends an administration command to a device.
GetActiveDevices	Returns a collection of descriptions of currently active devices.
GetActiveJobs	Returns a collection of descriptions of currently active input and output jobs.
GetActivePaperTypes	Returns a collection of descriptions of active paper types.
GetDeviceStatus	Gets the status of a specified device.
GetJobDetails	Gets information about a specified job.
LockDevice	Locks the specified device for a designated length of time.
Reset	Reloads the Peripheral Broker configuration from the message store.
UnlockDevice	Unlocks the device locked by a **LockDevice** call.
EGPBrokerCallBack Callback functions to inform the application of events.	
CallInterface	Called by the Peripheral Broker to inform an application of events.
DeviceErrorHandler	Called by the Peripheral Broker to handle device-related errors.

Table 5-1 Peripheral Broker Library interfaces

FUJ00001368

EGPBrokerClient Client functions for particular events in the Peripheral Broker.	
OnDeviceError	Called when a device error occurs.
OnError	Called when an internal error occurs in the Peripheral Broker.
OnInputEvent	Called when a device receives an input event.
OnOrphanImpulse	Called when an Orphan Impulse fires.
OnOrphanImpulseLost	Called when an Orphan Impulse is selected by another node.
OnUnknownData	Called when the Peripheral Broker cannot identify data sent to a device.
EGPBrokerInput Input events.	
CreateImpulse	Generates an impulse based on data.
MonitorEvents	Reads and registers or unregisters an Event Collection.
NotifyOnEvent	Notifies the application of specific event types.
ReadBlob	Reads an amount of binary data.
ReadData	Reads a message from a device.
RegisterEvents	Registers input events with the Peripheral Broker.
RemoveNotifyOnEvent	Removes events from notification.
UnRegisterEvents	Unregisters registered events.
EGPBrokerOrphanImpulse Orphan impulse events.	
AcceptImpulse	Attempts to accept a specified impulse.
GetImpulseDetails	Obtains information about an impulse.
RejectImpulse	Refuses to accept a specified impulse.

Table 5-1 Peripheral Broker Library interfaces (continued)

FUJ00001368

EGPBrokerOutput Output events.	
BeginJob	Marks the beginning of a job and provides job information.
CancelJob	Cancels a job.
EndJob	Marks the end of a job.
EndPage	Marks the completion of a page image. Should only be called after a call to **StartPage**.
NewPage	Marks a new page.
PauseJob	Pauses a job.
ResumeJob	Resumes a paused job, or a job that was interrupted during a **WriteData** call.
SetJobName	Names a job.
StartPage	Marks the start of a page.
WriteBlob	Writes a blob of data to a device.
WriteData	Writes formatted output to a device.

Table 5-1 Peripheral Broker Library interfaces (continued)

The Riposte Broker can also be used to access the message store without the use of the Retail Broker or Desktop. For details, refer to section 5.5.1 and *Riposte Broker API Reference Manual*.

FUJ00001368

5.2.4 Smart card system architecture and Interfaces

The software functions that support the handling of smart cards consist of a number of layers as shown in Figure 5-3:

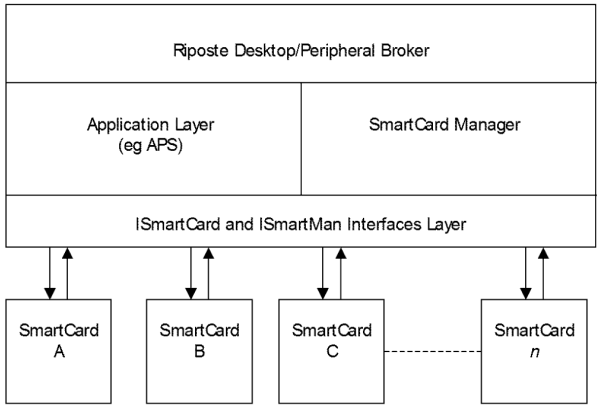


Figure 5-3 Smart card architecture

Smart Card Manager handles the result of all smart card impulses. It is responsible for identifying the CardDLL associated with the inserted card and notifying any interested application that an impulse has been received. The Smart Card Manager software acts as a buffer between the smart card reader, the Peripheral Server, the application and the CardDLL.

The Smart Card Manager, SmartMan, provides the technology-independent functions common to all card types such as retrieval of Reference Data definitions and access to the smart card itself.

The technology-dependent DLLs perform those functions that are specific to a particular type of card, such as updating credit details, extracting meter readings and encryption. There is an interface defined for ISmartCard that must be implemented by each CardDLL. Each CardDLL communicates with SmartMan through a separate ISmartMan interface.

The application level interfaces cover:

- Identification of the CardDLL.
- Identification of the Card Technology.
- Identification of the Token Types.
- Definition of the action to be taken by the clerk. This may be to abandon the transaction.

The definition of these interfaces are given in Appendix A, *SmartMan Interfaces*.

FUJ00001368

5.3 The Retail Broker

The Retail Broker co-ordinates the various applications in terms of a customer session and adds contextual information, such as stock unit names, to messages. It supports the concept of a mode and may make certain Desktop functions unavailable; for example, the Logout function is not available while the counter PC is in Serve Customer mode.

This section lists the interfaces that are available to add the sale of a product as a transaction to the stack, to cancel transactions, and to deal with any additional processing required at the point when a session is settled. It includes others used to access the message store for other reasons: scanning, writing, and manipulating persistent objects.

The Retail Broker object implements three interfaces:

- **EGBroker**

  EGBroker is an abstract class that is implemented within each broker. Its implementation is specific to each broker. It provides mechanisms to initialise, terminate and check the initialisation status of objects.

- **EGRetailBroker**

  EGRetailBroker is an interface that provides mechanisms for managing transactions as they are written to the message store.

- **EGRetailBrokerClient**

  EGRetailBrokerClient is an interface that provides the mechanism for settling transactions and for writing the transaction details to the message store. It assigns a user-defined transaction type and a unique transaction key to each transaction.

The **Connect** method lets an application connect to an instance of the Riposte Broker. It must be called first in an application before any other Riposte Broker method can be called.

5.3.1 Retail Broker interfaces

Table 5-2 lists the Retail Broker Library interfaces.

EGBroker initialises and terminates broker objects.

FUJ00001368

Initialize	Initialises a broker object.
Terminate	Closes and deallocates references held by a broker object.
EGBrokerStates	Read-only property. Set to an initial value when an **EGBroker** object is created.

EGRetailBroker provides the mechanisms for managing transactions as they are written to the message store

AddTransaction	Adds the transaction to the stack.
BalanceSession	Balances the session prior to committing it to the message store.
CancelSession	Removes all transactions. **(Not to be used by applications in shared sessions.)**
CancelTransaction	Cancels the specified transaction and removes it from the transaction stack.
ComputeCurrentCurrency	Performs arithmetic on the supplied values.
CurrencyValue	Returns the value of a currency string in the specified currency unit.
FormatCurrency	Formats and converts a currency description to the current currency. The returned value can be in any currency.
FormatCurrentCurrency	Formats and converts a currency description to the current currency. The returned value is in the current currency.
GetTransaction	Retrieves a transaction that matches a provided key.
NextCurrency	Gets next in set of defined currencies. Only to be used in accordance with system defined rules.
NextTransaction	Retrieves a transaction that matches a provided filter string.
SettleSession	Commits transactions to the message store **(Not to be used by applications in shared sessions.)**
StartTransaction	Records the starting time of a transaction, and return a handle for use when committing the transaction.

Table 5-2 Retail Broker Library interfaces

EGRetailBrokerClient provides the mechanism for settling transactions and for writing the transaction details to the message store.

FUJ00001368

OnAddTransaction	Called when a transaction is created. The information passed is the same as that passed to the **EGRetailBroker.AddTransaction** method.
OnCancelSession	Called when all transactions are removed from the transaction stack by the **EGRetailBroker**.**CancelSession** method.
OnCancelTransaction	Called when a transaction is cancelled. The information passed is the same as that passed to the **EGRetailBroker.AddTransaction** method.
OnCurrentContainerChange	Called when the current stock unit changes.
OnSettleSession	Called when a session is settled.
OnStartTransaction	Called when the **EGRetailBroker.StartTransaction** method is called.
PreSettleTransaction	Returns an Attribute Value String that describes the result of a call to SettleSession.
SystemBroker	A reference to the System Broker object. This object references other broker objects in the Riposte system

Table 5-2 Retail Broker Library interfaces (continued)

5.4 Desktop interfaces

The desktop interface is populated using a standard set of OCXs, which ensures that its consistent appearance and behaviour is maintained. OCXs are provided for application developers to use in building their applications; they are not accessed directly, but via the Desktop, using Riposte scripts. This restricts developers to the Riposte OCX collection and rules out the common Visual Basic development practice of using custom-designed OCXs. It is possible to modify an existing OCX by specifying different values for some of it is attributes such as panel style and field length.

A number of standard OCX actions control common features of the Desktop:

- Desktop Button

  The Desktop Button moves the user back to the main menu (for the current mode).

FUJ00001368

- Prev Button

  The Prev(ious) Button returns the user to the next screen up in the hierarchy, either the menu screen from which the transaction was selected, or the previous transaction screen in the case of an application that has a dialogue with the user.

- Bin Button

  The Bin Button takes the user into 'Remove Mode'. If it is allowed under the application's rules, an item subsequently selected can be removed (from the stack).

- Info Button

  The Info Button takes the user into 'Help Mode'. (Help for some OCXs is completely managed by the Desktop.) Note that Escher's HTML help, indicated by the ⓘ symbol, is not implemented in the Horizon system; because of the complexity of counter procedures, help is provided in a set of documents that is available to all counter clerks. The use of HTML help has performance implications, which are discussed in Appendix C, *System Management*.

Application developers select the OCXs that are appropriate for each application, via scripts and screen dialogues, and Riposte controls others that maintain the consistent behaviour and appearance of the Desktop. The available OCX types are as follows:

- The Calendar and Clock OCX displays the date and time in the stack estate when the stack does not contain any transactions. Application developers cannot specify its use, as it is entirely controlled by Riposte.
- The Help OCX provides information about any screen object that the user selects. Information is displayed in a speech bubble with a yellow background. Application developers supply the text and the Help OCX controls how and where on the screen it is displayed.
- Card and Button OCXs are used to offer options to the user. The Card OCX is a simple button that the user touches to make a selection. A Button OCX has an identical function but has a more complicated format with a tabbed caption.
- The Trans OCX is used for all panels that display information to the user, but cannot be altered or selected. It is normally displayed with a Card OCX to enable the user to react to the displayed information.
- The Script OCX is a panel that tells the user what data to enter and how to proceed. It is always displayed with one of the input OCX types: Calculator, Options or Message/Error.
- The Calculator OCX is a standard display format used to enter alphanumeric data.

FUJ00001368

- The Options OCX enables the user to make a selection from a list of alternatives.
- The Message/Error OCX gives information about a transaction or an error, and offers one or two courses of action.
- The List OCX provides a structured list format with navigation and search facilities. It has a number of uses, including the provision of Product Look-Up and Declaration lists.

Other aspects of the design of the man-machine interface (MMI) are described in the *Horizon Office Platform Service Style Guide*.

FUJ00001368

5.5 Riposte functions

Riposte supports a number of message-handling APIs that are implemented via Remote Procedure Call (RPC) mechanisms. They support natural C or C++ interfaces to a local or remote application. Functions are implemented as OCXs that can be multi-threaded within the Windows NT environment and which support Visual Basic error-handing mechanisms so that they can be used by desktop applications that are coded in Visual Basic.

There are three layers of APIs:

- The Message Layer, which includes standard facilities for creating and manipulating messages.
- The Persistent Object Layer, which includes additional facilities for creating, updating, searching for and selecting persistent objects.
- The Reference Data Layer, which provides additional facilities to select records that are effective and applicable to the caller. Reference Data is a particular example of persistent objects, and is used to pass parameters to applications and to make applications aware of changing data. Reference Data messages can be created with an 'effective date' in the future, and the Reference Data interface, if called in an outlet, will not return the record until that date. Similarly, some Reference Data is only applicable to some outlets (for example, not all outlets sell Vehicle Excise Licences). Data is only returned if it is relevant.

The three layers are shown in Figure 5-3. An implication of this diagram is that Reference Data is accessible through the persistent object APIs as well as its own APIs, and everything is accessible via the Messaging APIs.

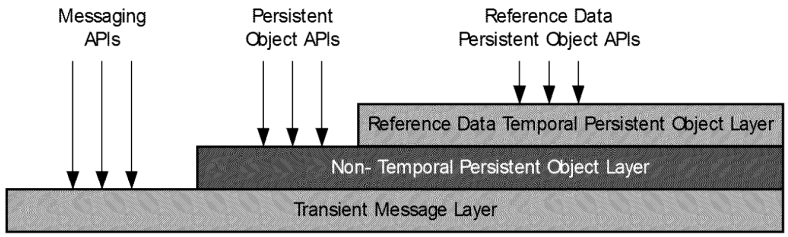


Figure 5-3 Messages, Persistent Objects and Reference Data

Reference Data is intended for use by counter applications, and a set of Riposte OCXs is provided to access it.

FUJ00001368

5.5.1 Messages

All information used by Riposte is held as messages in attribute grammar format (see section 4.1.2.1). Most messages are created as a consequence of information being passed down to counter applications, or as a result of customer transactions at the counter. The methods that are available for generating and retrieving messages, some of which are referred to in this section, are documented in Escher's *Riposte Broker API Reference Manual*.

5.5.1.1 Replication

Messages contained in the message store of any counter PC are replicated to all other counter PC nodes within the outlet and to the central Correspondence Servers. Data replication into and from this local message store is essentially transparent to local applications running on the counter PC, which access the message store using the Riposte API to create, retrieve or parse messages independently of any replication activity.

5.5.1.2 Message types

Messages are used for the following purposes:

- Transactions

  When a customer transaction is committed, a set of messages is written to the message store. If a system failure occurs before the transaction is committed, any messages created within the transaction are rolled back.

- Enquiries and responses

  For example, a request for messages that satisfies a set of criteria (see also section 5.5.1.5 for information about ports).

- Audit and monitoring information

- Authorisations and instructions

  For example, OBCS foreign encashment authorisation.

- To define the text of instructions to counter clerks.

- To maintain a session context

- To record system events such as a user login or use of a card.

All messages have a unique message ID and record the group at which it was created and the node within the group. The group is derived from the Financial Accounting Division (FAD) code by dropping the check digit character. The node is the machine number within the group, and sequence number is within that node. Riposte does not store a message it receives from another node unless it has the preceding message from that node. If it has not, it requests the missing messages from the other node.

FUJ00001368

5.5.1.3 Markers

The sequence number field within a message identifies its position in the sequence of messages generated by that system. Although there is no relationship between the sequence number fields of messages from different systems, Riposte supports the concept of a marker. A marker is a string that holds a series of the latest sequence numbers for each system in the group, so it is possible to state that a particular message precedes or follows a given marker.

Markers can be used to delineate a cut-off point for a set of messages. For example, when it is necessary to balance the outlet at the end of the day, a marker can be used to decide which messages are balanced, and which are held over to the next day.

Markers are also used by Riposte to co-ordinate message stores in the event of failures.

5.5.1.4 Checkpoints

Checkpoints apply to a whole message store (to all Correspondence servers in a cluster) and are used by Agents to control the recovery of information after a failure. Refer to *TMS Architecture Specification* for information about clusters.

5.5.1.5 Message ports

If a counter application creates a message that contains an enquiry or update, it creates a message port waiting for a response, using the key of the message ID of the enquiry or update. The message port is set to time out after a time specified by a configuration parameter set in a persistent object.

Real Time Message Ports may be used by Agents to ensure that priority messages can be processed immediately by the message port, without waiting for the message to arrive in sequence. Note that delivery through a Real Time Message Port cannot be guaranteed, and duplicates may also arise under certain recovery scenarios.

5.5.2 Persistent objects

Persistent objects provide a means of permanent storage of standard values and other data. They have additional properties over and above normal messages, of which the two most important are:

- The content of a persistent object can be changed by the creation of a new version; Riposte always returns the latest version.
- The latest version of a persistent object is never purged. Earlier versions are purged on the same basis as normal messages.

FUJ00001368

For details of the structure of persistent objects, refer to section 4.1.4. It should be noted that persistent objects are used only for Reference Data or for messages to the clerk.

Use of persistent objects for any other reason can have significant performance implications. The rules that define their usage and the way performance implications are defined and modelled are stated in Appendix C, *System Management*.

5.5.2.1 Persistent object collections

Persistent object collections provide a means of grouping like persistent objects together. The persistent object store corresponds in many ways to a database table. The collection name corresponds to a table name, and the object name to a prime key. Where more than one message has the same collection name and object name, the one with the highest version number is returned, unless the 'deleted' flag is set in the highest-numbered version, in which case no record is returned.

5.5.2.2 Reference Data

The counter applications are heavily dependent on Reference Data needed to instantiate them and to feed other components of the system with information generated by POCL or its Clients.

Reference Data is stored within Riposte as persistent objects. Each item of Reference Data has an effective date associated with it, which allows the data to be downloaded to the counters before it is needed, and to become available automatically on the effective date. The data is not read directly from the persistent objects by the counter applications. Instead, there is an effective date algorithm that is used to provide the calling counter application with the appropriate value for the Reference Data it requires, based on today's date.

Business Reference Data (Type A and Type B Reference Data) becomes available whenever it is read by an application on the effective date; all times used are GMT. For Client Reference Data the same applies. Implementation Reference Data (Type C and Type D Reference Data) becomes active on the effective date, the first time the Desktop is loaded on that day.

5.5.3 Message-handling interfaces

The Riposte Broker provides methods to retrieve messages, to parse messages, and retrieve attribute names and values. Similarly, there are methods to enable an application developer to construct a Riposte message and define its attributes. (Refer to section 4.1.2.1 for a description of attribute grammar structures.) The following sections group the methods according to their use and give a brief description of each. For full details of these and for additional methods, refer to Escher's *Riposte Broker API Reference Manual*.

FUJ00001368

5.5.3.1 Transactions

Function	*Action*
StartTransaction	Starts a transaction and returns a handle to it.
EndTransaction	Commits the transaction, frees server resources associated with the transaction handle.
UndoTransaction	Undoes the transaction. Rolls back any associated messages. Frees server resources associated with the transaction handle.

Table 5-3 Riposte Broker Library interfaces: transactions

5.5.3.2 Messages

Function	*Action*
CreateMessage	Creates a message.
GetMessage	Retrieves a message from the message store (using key).
CreateIndex	Creates an index on the specified attribute for message retrieval. **(See note below.)**
DestroyIndex	Destroys an index generated by **CreateIndex**. **(See note below.)**
MessageFileTime	Retrieves the creation time of a message in FILETIME format (in Co-ordinated Universal Time).
MessageGroupID	Retrieves the group identifier (FAD code) of a message.
MessageID	Retrieves the message identifier (FAD code, node ID, sequence number).
MessageNodeID	Retrieves the node identifier of a message.
MessageNum	Retrieves the sequence number of a message.
MessageSystemTime	Retrieves the creation time of a message in SYSTEMTIME format (in Co-ordinated Universal Time).
NextMessage	Retrieves the next message in the message store that matches specified criteria.
ScanMessage	Retrieves the next message in the message store, within a specified range, that matches specified criteria.

Table 5-4 Riposte Broker Library interfaces: messages

Note: Refer to Appendix C, *System Management,* for information about the performance implications of creating and destroying indexes.

FUJ00001368

5.5.3.3 Queries

Function	*Action*
AccessQuery	Accesses a message retrieved by a query.
CreateQuery	Creates a query to retrieve messages satisfying selection criteria.
DeleteQuery	Deletes a query created by **CreateQuery.** Frees associated resources.
QueryStatus	Retrieves status information for a query.

Table 5-5 Riposte Broker Library interfaces: queries

5.5.3.4 Markers and checkpoints

Function	*Action*
CreateCheckpoint	Saves the current checkpoint of the host message server. Only for use by agents. Counter applications should use a marker instead.
DestroyCheckpoint	Destroys a checkpoint saved using **CreateCheckpoint.** Only for use by agents. Counter applications should use a marker instead.
GetMarker	Returns a pointer to a marker structure.

Table 5-6 Riposte Broker Library interfaces: markers and checkpoints

5.5.3.5 Manipulating persistent objects

Function	*Action*
CheckObjectAccess	Verifies user access rights to a persistent object.
GetObject	Retrieves the current version of a persistent object.
NextObject	Enumerates persistent objects that belong to the specified collection and the specified attribute value list.
RemoveObject	Writes a persistent object version that identifies it as deleted.
PutObject/PutObjectSec	Creates or updates a persistent object.

Table 5-7 Riposte Broker Library interfaces: persistent objects

FUJ00001368

5.5.3.6 Message ports

Function	*Action*
CreateCheckpointedMessagePort	Creates a checkpointed message port for an Agent.
CheckpointMessagePort	Saves the internal checkpoint of a checkpointed message port for an agent.
WaitForMessagesWithTimeout	Creates a non-checkpointed message port.

Table 5-8 Riposte Broker Library interfaces: message ports

5.5.3.7 Parsing and manipulating attribute grammar objects

Riposte provides the following functions for parsing objects:

Function	*Action*
ObjMake	Creates an object.
ObjLen	Returns the length of an object.
ObjName	Returns the name of an object.
ObjValue	Returns the value of an object.
ObjAttribute	Returns object's specified attribute and its value.
ObjAttributeValue	Returns value of specified attribute.
ObjAttributeComplement	Removes the specified attribute.
ObjAttributeUpdate	Replaces attribute value with new value.
ObjAppendAttribute	Appends an attribute to a specified attribute grammar object.
ObjGetValue	Retrieves value of specified attribute grammar object.
ObjListFind	Finds list of attribute grammar objects.
ObjRemove Attribute	Removes an attribute from the specified attribute grammar object.

Table 5-9 Riposte Broker Library interfaces: attribute grammar objects

FUJ00001368

6 Other Functions

This section describes the interfaces that support the following features of the system:

- Administration
- Security
- Availability
- Usability
- Performance
- Resilience

6.1 Administration

6.1.1 User administration

The functions that are available for managing users and security are as supplied by Riposte, and are not configurable by individual applications.

Riposte provides facilities for users in outlets who have the system role 'Manager' to define users, to change their passwords, and to delete users. Other facilities enable the manager to specify:

- That a password must be changed when a user next logs on
- A user's roles

A user can have one or more than one role. The functions available on the menus to a particular user are determined by the access rights available to the role(s) selected. Users who have more than one role on the system, have all the access rights that are defined for each of those roles.

The system supports the following roles, known as 'user groups':

- ENGINEER: This role has only a hardware diagnostic capability.
- AUDITOR: This role gives access to the limited functionality that supports the auditing process.
- AUDITOR E: This role has full manager rights, and is only used in emergency situations if another employee needs to take over the post office.
- SUPPORT: This role has only administrative functionality to create users. For instance, it is used if a manager forgets his or her password.
- MANAGERS: This role allows all access to all functions. User administration is restricted to this group.

FUJ00001368

- SUPERVISORS: This role has access to all counter and some administrative functionality.
- CLERK: This role has access to all the counter functionality.
- SETUP: This role operates only on initialisation of the system. The only rights this group has are administrative ones in order to create a manager user. Once this has been completed, the group must be deleted.

Groups not appropriate to outlet users (MIGRATE, ENGINEER, AUDITOR, AUDITOR E, SUPPORT and SETUP) cannot be allocated to new or existing users. These roles can be assigned only by the system start-up process for agreed user names.

FUJ00001368

6.2 Security

System security is achieved in a number of ways:

- By controlling access to desktop menus and applications
- By the use of encryption
- By the use of digital signatures

6.2.1 Users

Access to the Horizon desktop is controlled by the user administration functions described in section 6.1.

6.2.2 Menus and applications

Part of the data held within Reference Data defines the mappings from buttons or impulses to applications and to counter users. It also determines which component of the application handles the button or impulse. Not all applications and their associated buttons and impulses are available to all users. Some are reserved to the outlet manager and others to auditors. The access controls must be defined and set up as part of the application design. They include the user-to-application privileges associated with each of the user groups. New applications need to define their own controls.

6.2.2.1 Access Control Lists

Access Control Lists (ACLs) are defined, like everything else in Riposte, in an attribute grammar syntax (see Appendix C, *System Management*). They provide an ‘allowed list’ and a ‘denied list’, with the latter taking precedence.

Some messages (or parts of messages) need to be signed. The Access Controls define which these are, and also allow the application to verify the signature on signed messages that it receives.

6.2.2.2 The MenuSecurity collection

The **MenuSecurity** collection in Riposte Reference Data defines each menu in terms of user access; each button on a menu may be specified separately for access purposes, or the same access rights may apply to an entire menu screen.

FUJ00001368

6.2.3 Digital signing

For integrity-critical products, such as APS, standard simple public key technology is used for digital signing.

Under public key technology, protected messages are digitally signed by a private key and validated using the private key's matching public key.

Transactions are digitally signed at the post office and then signature-verified at the harvesting agent that takes the transactions from the correspondence servers, or vice versa. Complete files of transactions are signed before transmission to Client hosts.

Applications and agents use cryptographic functions in order to sign, verify, encrypt or decrypt data or files. The cryptographic functions shield the application from requiring knowledge of the Key Management Service's (KMS) managed keys used to perform these operations. For details of the functions available for cryptographic operations, refer to Appendix B, *Cryptography and Key Management*.

The harvester or application verifies that the data in each transaction is valid by checking the digital signature added by the counter. Any invalid transactions generate an error event in the NT Event Log. These failing transactions are investigated manually.

6.3 Availability

Optimum availability is achieved through a combination of factors:

- Reliable components, both hardware and software.
- Detectable failures, so that a failed component is immediately detected and notified to the support organisation.
- Resilience, so that major components are replicated or include retries and error processing to minimise the impact of failure.
- Recovery and repair processes that can restore a failed component or system to normal operation speedily.
- Distribution, installation and activation processes that can be used to install and configure components remotely to fix an identified problem, or to minimise the disruption when a key component is out of service.
- Measurement techniques that may be used to determine the overall availability of the services provided by the information system and whether these conform to agreed service levels.

The availability requirements for the Horizon system are specified in a number of Service Level Agreements (SLAs). The Horizon system is required to be available to the clerk for normal counter operations during working hours. It also runs a number of unrelated applications that have to be scheduled in such a way that they allow ICL Pathway to keep within the

FUJ00001368

terms of the Service Level Agreements and so that they do not interfere with each other.

6.3.1 End of Day

Applications that produce external data need to be synchronised in their view of 'End of Day'. If this is not achieved, reconciliation processes between external bodies will throw up anomalies. End of Day in this context means the identification of the transactions that count as belonging to the current day's transactions and those that will be included in tomorrow's.

End of Day is indicated by the arrival of an End of Day (EoD) marker for each outlet. However, the production of an End of Day (EoD) marker is derived from POCL outlet Reference Data that defines the opening and closing times of each outlet. If the outlet opens on a particular day, the time at which the processing of EoD is initiated, and the marker written, is set to closing time plus 30 minutes, unless this would result in a time later than 7pm, in which case 7pm is used. If the outlet does not open on a particular day, EoD processing is initiated at 7pm.

The operation here is driven by the requirement that the same data is sent to all Clients and POCL via TIP. Where an outlet does not report End of Day by the time the latest time for this signal, no data is sent to any Client or TIP. If a counter is offline, the End of Day processing does not occur.

A generic harvester reads these EoD markers. For each one that it receives, it updates a persistent object in a virtual group that is used to delineate which outlets may be harvested for the day. All other harvesters only harvest messages from outlets that are recorded within the virtual group.

The EoD marker is written to the mainstream message store, rather than to the active message store (which may be the training one). There are two types of EoD marker, private and public. It is the public markers that are used for harvesting. Each counter position creates its own private EoD marker and it is the responsibility of the Gateway terminal to create the public marker and mediate between private markers in recovery situations. If a counter position is unavailable at End of Day, the EoD marker is created when the position becomes available again. One EoD marker is written for each 'missing' day.

Transactions added to the message store after the EoD marker has been written to it appear as 'tomorrow's' rather than 'today's' transactions in TIP.

6.3.2 Disconnection

6.3.2.1 Disconnected outlets

If an outlet becomes disconnected because of a communications failure, it can continue to be used for the majority of applications. Some transactions

FUJ00001368

cannot be performed because of particular business rules. Transactions that require on-line queries or verification can be performed by using a Helpdesk telephone connection. When the communications link is restored, Riposte updates the correspondence server with all the messages that have been written to the message store since the last connection was made.

The rules about which transactions can be conducted when there is a communications failure are business-specific.

6.3.2.2 Disconnected counters

Many outlets contain multiple counter systems, and so the loss of one (for example through power supply failure) does not prevent the use of other counters.

Riposte ensures that transactions completed at a counter are replicated both on the other counter PCs within the outlet, and on a number of correspondence servers in the data centres. It provides for automatic recovery of transaction data. If a counter PC fails, its transaction can be completed at another counter PC within the outlet.

An application can determine the status of all nodes that belong to the group by use of two Retail Broker commands:

- **NextNode** makes it possible to establish all the neighbours in a group.
- **GetConnections** makes it possible to establish whether or not any data has been received from a node within two Riposte marker exchange periods. Exchanging markers is the method used by Riposte to synchronise neighbours.

The rules about which transactions can be conducted when the counter is disconnected are business-specific.

6.3.3 Reboot

The POLO process is invoked to unlock the counter PC's filestore during its boot process. Most of these services will fail if they run on a locked filestore, and so they cannot be started automatically by Windows NT. Instead, they are defined as ‘started manually’, and it is the POLO process that starts them. They are recorded in a registry entry that is read by POLO. This mechanism ensures that no user-accessible desktop functions can run until the Post Office Manager has successfully executed a POLO process.

6.3.4 Reload

In the Horizon environment, Windows NT reloads the Riposte Desktop overnight. If the counter is in use, the reload is postponed. This reload is necessary to ensure that new versions of persistent objects are picked up.

FUJ00001368

The persistent objects are used to configure some applications, and to specify the menu buttons and their meanings.

A utility called *PathwayLoad* is used to start (and stop) the Riposte Desktop. When it is necessary to update the restartable software on any counter PC overnight, the following sequence is automatically executed:

- Call *PathwayLoad* to close down the desktop (it uses APIs provided by Riposte)
- Shut down the Riposte-related services
- Shut down the Riposte Message Server(s)
- Update the necessary software
- Restart the Riposte Message Service(s)
- Restart the Riposte related services
- Call *PathwayLoad* to start the desktop

FUJ00001368

6.4 Usability

The different functionality of the component Horizon applications is blended into a single user interface, where consistency is achieved through using the defined set of facilities provided by the Riposte software. Certain conventions ensure that similar actions performed in different applications produce similar effects. For example, a numeric value that is entered incorrectly produces an error message worded similarly and in the same position on the screen, whether it is the sale of a stamp or the payment of a savings account withdrawal.

The system must be intuitive and discoverable to the user. Intuitive in this sense means that without guidance the user can deduce from the system's responses what to do next. Discoverable means that where any action is not obvious, it can be deduced. In neither case should the user be led to an irrecoverable position.

In order to achieve this, the following guidelines are followed:

- The system identifies at each point in a dialogue all the options open to the user. This is achieved using a combination of displayed instructions and prompts in text and pictorial form.
- Similar user interface styles are used in similar situations.
- Users have options available that enable them to change their minds.
- Prompts only require confirmation when the next action cannot be undone.

The *Horizon Office Platform Service Style Guide* defines the interface standards as described in the following sections.

6.4.1 Screen types

The screens within the Horizon system are of three main kinds:

- Menu screens from which to navigate to the required function or task.
- Transaction screens to input data or perform a specific task.
- Report screens for selecting, previewing and printing reports.

The menu screens present the user with a maximum of sixteen buttons (in addition to navigation buttons) with which to select products, applications and lower-level menus.

The transaction screens present a combination of buttons, information panels and data input panels.

The buttons that enable the user to choose an option or take a particular action are activated via the touch-sensitive screen or the keyboard. The data input panels allow the entry of data using the on-screen keypad or

FUJ00001368

the keyboard, and the information panels give instructions to the user and information about the status of the transaction. Some applications may also be activated by swiping a magnetic card or via a bar-code reader.

The consistency of the appearance of both menu and transaction screens is controlled by the use of the standard Riposte Desktop application interface.

The elements of screen layout that it controls are as follows:

- The shape and position of navigation and function buttons.
- The shape and position of the buttons on menu screens.
- The shape and position of data and information panels on input screens.

The following aspects can be varied under the control of the application programmer:

- The types of panels and buttons that are used.
- Particular effects using text colour and non-default panels.

Global visual attributes such as background colour are system-wide and are not under the control of application programmers.

6.4.2 Desktop components

The Desktop is laid out using a number of buttons, panels and scripts, which are assembled into named software component types known as OCXs. Some of these types are entirely under the control of the Riposte software and others are available to programmers to use in their applications.

6.4.3 Panel and button styles

Each OCX type has a number of different properties, including the style of the panels and buttons that are used and the text that is displayed on them.

Panel and button styles used by the Desktop are either as supplied by the Riposte Desktop application interface or are customised versions of them.

6.4.4 Data types

Within the Horizon system, data input is handled by Riposte. Formats remain consistent, irrespective of the source of the data.

Depending upon the task, data can be input using one or more of three methods:

- Using a scanning device such as a bar-code reader, smart card reader or a magnetic card swipe.
- Selecting options using the touch-sensitive screen.

FUJ00001368

- Using the keyboard.

6.4.5 Navigation

Navigation between screens is achieved by selecting an item from a menu or by using general navigation facilities to skip menu levels or return to a specific level. The options depend upon what mode the user is in. In 'Serve Customer' mode, for example, the user is returned to the 'Serve Customer' menu when the Home key is selected. In other modes, the user may be returned to the 'Transactions' menu.

External impulses such as bar-code scans navigate to the correct screen for processing the input. For example, if the bar-code of a Child Benefit order book is scanned, the appropriate transaction screen is displayed.

General navigation facilities are provided by the Riposte Desktop application interface. The application programmer, however, can modify navigation requests within an application by trapping them and processing them as necessary.

6.4.6 Functions

Among the generalised controls that are provided are a 'Bin' facility that enables the user to remove a transaction from the stack if the business rules allow it, and a 'Suspend' facility that suspends the current customer session so that a new one may be started.

6.4.7 Menus

Menu screens are used as a means of offering the user a set of options. The user selects an option and is then presented with a screen for performing a task, or with a further menu screen from which to make a selection.

Riposte controls the overall appearance of menus and the absolute positions of the buttons. The application interface enables the programmer to specify the relative position of each option button so that they can be arranged in a way that is most convenient for the user.

6.4.8 Messages and help text

Information and error messages are handled in a consistent way using the Message/Error OCX. A number of guidelines are defined in *Horizon Office Platform Service Style Guide* for the consistency of messages.

6.4.9 Icons and colour

Icons are used to assist in the rapid identification of products and functions, as well as adding to the visual appeal of the Desktop.

FUJ00001368

Guidelines are provided in *Horizon Office Platform Service Style Guide* for the design of icons and for the use of colour in icons, backdrops, panels, buttons and scripts.

6.4.10 Reports and Receipts

The designs of reports, till receipts and slips, depend on their contents and usage as well as the stationery on which they are printed. Current definitions are included in *OPS Reports and Receipts*. The *Horizon Office Platform Service Style Guide* describes the style generally adopted for reports and receipts.

6.5 Performance

Performance is a measure of the rate at which a system is able to perform useful work, its throughput and the responsiveness with which the results are presented. There are a number of factors to be considered when designing new applications, in order to minimise the impact on the existing OPS application services.

6.5.1 Service Level Agreements

As for availability, performance criteria are specified in a series of Service Level Agreements (SLAs). The three main areas covered by SLAs for performance are illustrated in the following examples:

- On-Line Transactions

  A complex set of maximum transaction times is defined. They range between 20 and 33 seconds, depending on the type of transaction.

- Transaction Information Processing

  Files must be delivered to POCL's Transaction Information Processing Application (TIP), which handles transaction data returned from Horizon, by 03:00 on the day following that to which they refer.

- Reference Data

  Reference Data that is authorised and delivered to ICL Pathway by 20:00 on a particular day (day 1) must be available to the following percentages of outlets by the start of core days 2, 3 and 4 as follows:

Type	*Day 2*	*Day 3*	*Day 4*
IMMEDIATE	97%	99%	100%
ADVANCE	---	99%	100%

FUJ00001368

6.5.2 Performance monitoring and measurement

Monitoring mechanisms allow performance parameters and volumetric data to be extracted. They vary from platform to platform. Where it is possible to monitor behaviour against predetermined thresholds, any instances in which the thresholds of key measurements are exceeded, are notified as events to the Tivoli Systems Management software.

Various monitoring mechanisms are available as follows:

- Windows NT

  Use is made of the Windows NT Performance Monitor. This is especially useful in measuring the performance of the Riposte system on counter systems and correspondence servers.

- Riposte

  All messages generated by Riposte are time-stamped. As well as helping with message synchronisation, these timestamps are gathered by the Data Warehouse and are used to check ICL Pathway's conformance to the agreed SLAs. Riposte itself maintains over 100 statistics that can be accessed via the Windows NT Performance Monitor.

6.5.3 Performance modelling

The loading on all the key components of the system are modelled by ICL Pathway as the workload grows and as new services come on-line. This is achieved by the construction of a resource model that predicts future component loadings based on volumetric information and system performance parameters.

The output from this model is fed into the design and development processes and helps to identify where performance shortfalls are likely to occur. Early warning of performance issues is necessary to enable designers and implementers to review and modify designs and implementations (where appropriate) or to recommend changes to the system configuration before performance is endangered.

The model's algorithms can be tested by comparing the predictions, once calibrated, with the system's real behaviour.

A number of performance models are used and focus on components of the system that are heavily used, or where there is a requirement to process on-line or batch workloads within the SLA constraints.

A sizing activity estimates the resources required to process each of the main workload components. The resource loading is generated by applying the resource estimates to the throughput model. The model can be used to identify performance deficiencies within the system. It can also be used to evaluate alternative solutions.

FUJ00001368

A system model is used to link together the component resource models and to allow the impact of one component on another to be assessed. It enables questions about the effect of changes in transaction mix and volumes to be answered across the system.

The volume of counter and batch transactions to be generated by a new business application must be forecast accurately so that its effect on system loading and throughput can be assessed before development is started.

Designers are required to provide information about new applications to ICL Pathway. ICL Pathway can then model and assess the application's impact.

6.6 Resilience

6.6.1 Message replication

Riposte ensures that transactions completed at a counter are replicated both on the other counter PCs within the outlet, and on a number of correspondence servers in the campuses. It provides for automatic recovery of transaction data. If a counter PC fails, its transaction can be completed at another counter PC within the outlet.

6.6.2 Agent managing of message recovery

Agents are responsible for managing recovery and message re-synchronisation between the host layer and the counter layer following a failure.

Riposte contains transaction features that allow multiple messages to be committed as one atomic unit. These ensure that messages are replicated as a unit, and another processor cannot read one message in the message run until all the messages in it have been replicated to that processor.

Equivalent facilities are available at the host layer within SQL, and must be designed into other application types as appropriate.

Bulk agents must mark their work chunks as 'in progress' as soon as they start to process them. If the agent fails, it is detected by a bulk monitor running on the host central server. The bulk monitor initiates a recovery process that recovers the failed work chunks.

Enquiry agents and interactive agents run as Windows NT Services. Failure of one of these is detected by the Tivoli Systems Management software which then restarts the agent.

The general strategy towards message recovery is:

FUJ00001368

- Applications must be able to cater with the same message content appearing several times (repeated message insertion may occur under failure conditions) typically by ignoring any duplicates.
- Agents must guarantee not to lose messages, though they can duplicate them.

FUJ00001368

7 Agent Interfaces

This section describes the generic structure of agents used by ICL Pathway and the constraints that must be recognised in the design of their interfaces. It is recommended that all agents follow this strategy.

7.1 Introduction to agents

A new application is likely to need a new agent component to extract the transaction data it needs for a Client. Agents are unique to applications and do not interact with agents belonging to other applications. The structure of agents is shown in Figure 7-1.

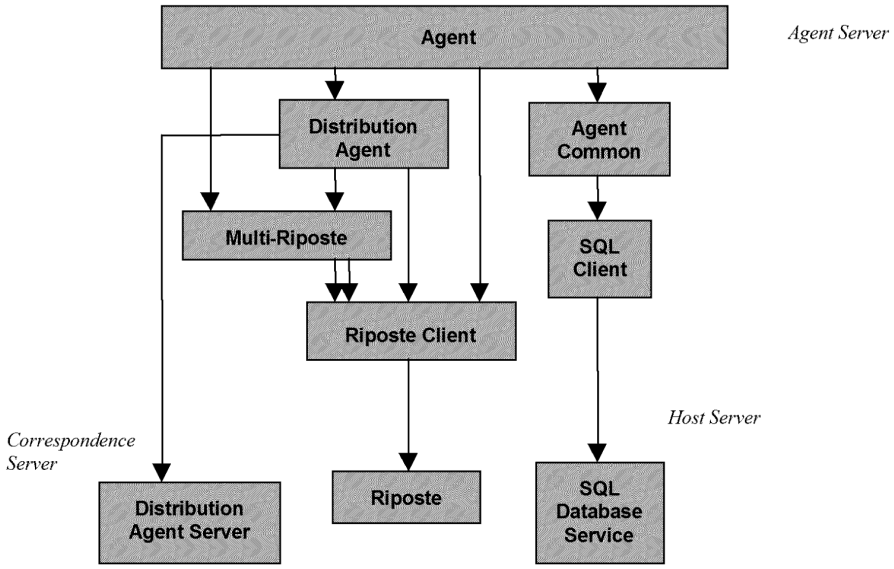


Figure 7-1 Agent structure

Agents are responsible for transforming the format of transaction data messages between that required by the Host and that required by the counter application. The transformation can take place in both directions.

Riposte holds the data in 'unnormalised', attribute grammar format, whereas the Host application can hold the data in a variety of formats, commonly as database tables, such as Oracle, or flat file structure. Figure 7-2 shows the relationship between data associated with a banking

FUJ00001368

transaction at the Host and at the counter. In this case the Host uses two database tables.

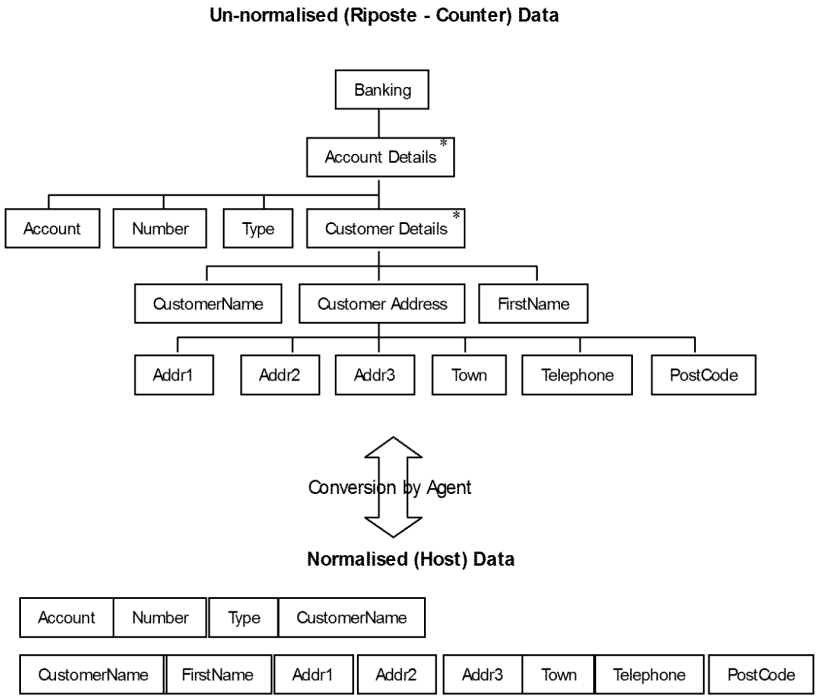


Figure 7-2 Data transformation

Agents can process messages either individually, or in bulk. Agents are therefore designed specifically to use either one method or the other.

Typically, agents are written in C.

7.1.1 Flat File format

For the purposes of this document, a flat file is one that is manipulated by a File Management Service, such as NTFS in Windows NT, as opposed to a database that is manipulated by a database management system such as Oracle. ICL Pathway normally expects that flat files be encoded using ISO 8859-1.

The format of flat files may be dictated by other suppliers, such as Microsoft. If the files are under the control of ICL Pathway, it advised that:

- Files include both a header and a trailer.
- Files include a record count, and sub-record counts, where appropriate.

FUJ00001368

7.2 Types of agents

Agents are grouped in three categories:

- Bulk agents: Inbound or harvester, and outbound or loader. Bulk agents process bulk messages or records.
- Interactive agents: Inbound or harvester, and outbound or loader. These process single messages or records (events).
- Enquiry agents: These process messages requiring authorisation or acceptability.

The structure of agents depends on whether or not the Host application uses an SQL database, as this interposes SQL client and server components into the data path. Such components are not present in the Flat File model.

There are also agents that ICL Pathway uses to manage aspects of messages, such as the General Acknowledgement Agent, described below.

Note: In the diagrams in this section solid lines indicate the flow of data, while the broken lines indicate a call.

7.2.1 General Acknowledgement Agent

The General Acknowledgement Agent serves two purposes:

- It demonstrates the success and timeliness of the arrival of data from the Data Centre to the counter.
- It can trigger an additional process if a message does not reach its destination within a predefined period of time.

As shown in the diagram below, there are two variants of this agent:

- On one of the counter PCs at an outlet

  This acknowledges the receipt of regular data, including Reference Data, from the Correspondence Server in the Data Centre.

- In the Data Centre

  This acknowledges the receipt of messages from the counters.

FUJ00001368

The messages issued by these variants of the General Acknowledgement Agent are shown in the following table:

Action	*Messages*
Acknowledge a message This provides for the measurement of the time between despatch and arrival, and thus the performance in delivering the message. This action causes two messages to be issued:	**Acknowledgement Request –** by the external agent.
	Acknowledgement Result – by the Acknowledgement Agent.
Alert if a message is not received within a specified period of time. This action causes the same two messages to be issued, together with an optional third message: **It is recommended that this mechanism is not used and that such checking is done in Host applications not Agents.**	**Acknowledgement Request –** by the external agent.
	Acknowledgement Result – by the Acknowledgement Agent.
	Acknowledgement Timeout - only generated if a **Request** cannot be matched with a corresponding **Result.** An event is written to the Event Log.

If a Request and a Result can be matched as a pair, then the Timeout is not issued.

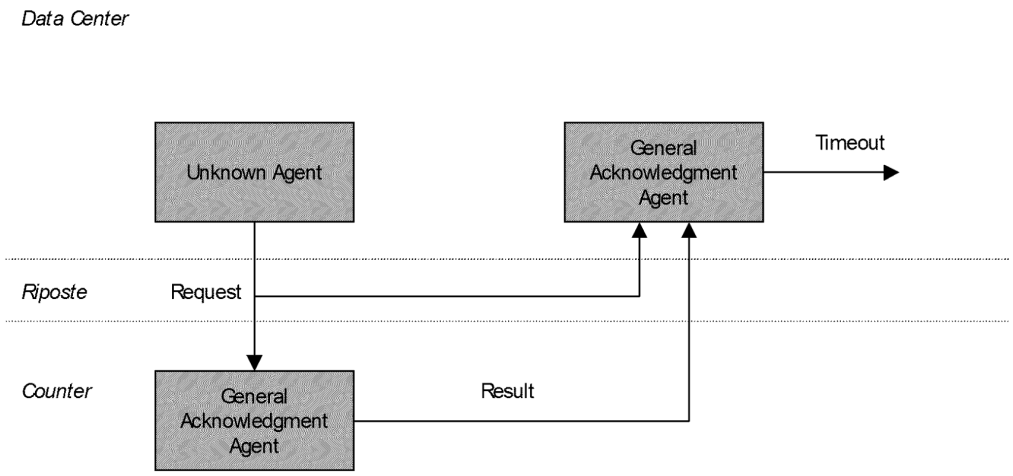


Figure 7-3 General Acknowledgement Agent

The set of messages handled can be extended, and the way they are handled can be tailored. There is a standard persistent object for each message type; and default handling styles for particular message types.

7.2.2 Bulk agents

Bulk agents operate on all messages of any specified type held in the Riposte message store.

FUJ00001368

7.2.2.1 Bulk harvester agents

These have similar structure to the interactive harvester agents, but they are started by a scheduler (Maestro) to harvest all the messages that have accumulated in the Riposte message store, rather than waiting on a message port for the arrival of a message and processing it as soon as it arrives.

Bulk harvesters run at defined periods, depending on a combination of performance requirements, SLAs and data volumes. Maestro schedules such agents automatically, but the agents must have their own control mechanisms to identify the outlets to be harvested.

For SQL-based bulk harvester agents, the structure is shown in the Figure 7-4.

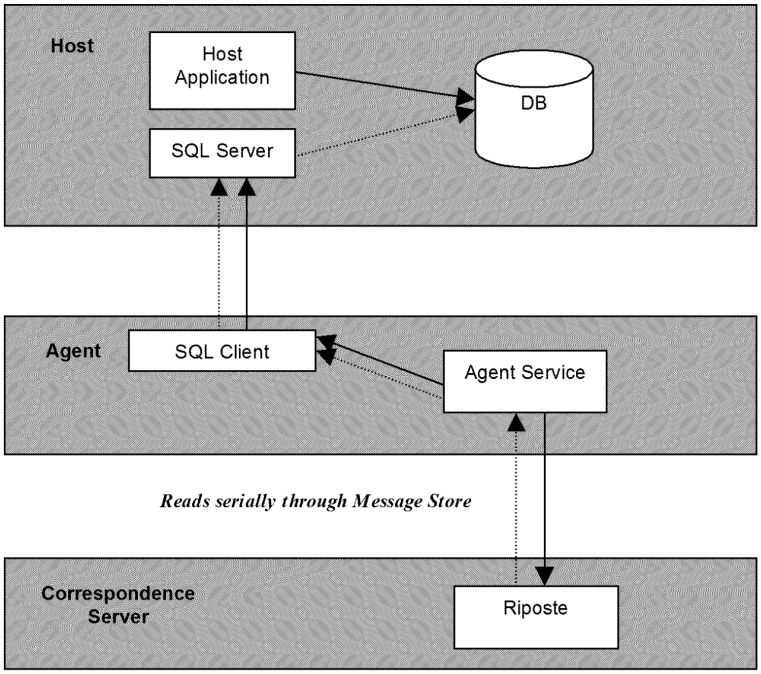


Figure 7-4 Structure of a bulk harvester agent

Typically each instance of such harvester agents process a sub-set of the outstanding messages on an outlet by outlet basis. harvested records are re-formatted into the format needed by the Host. When all outlets have been harvested the harvester exits.

Non-SQL (Flat File) bulk harvesters generate a set of flat files. The Host application may manipulate the flat files directly, or it may bulk load them into a database for manipulation (using the SQL flat file load option). These harvesters should operate between markers in the Riposte message store. (More than one flat file may be needed.)

Non-SQL bulk harvesters have the structure shown in Figure 7-5.

FUJ00001368

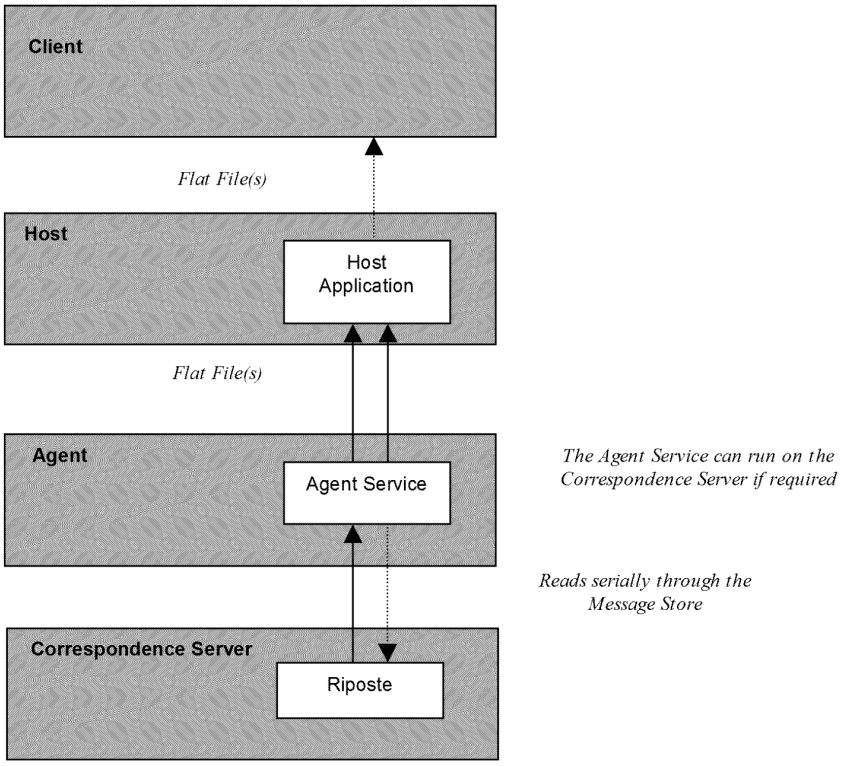


Figure 7-5 Flat File bulk harvester agents

7.2.2.2 Bulk loader agents

The structure of these depends on the complexity of the Host application, and whether or not it uses an SQL database.

Simple SQL-based bulk loader agents have the same structure as interactive loaders. The Host passes records to the agent and the agent passes them to the Riposte message store.

A bulk loader agent where the Host uses an SQL database is shown in Figure 7-6.

FUJ00001368

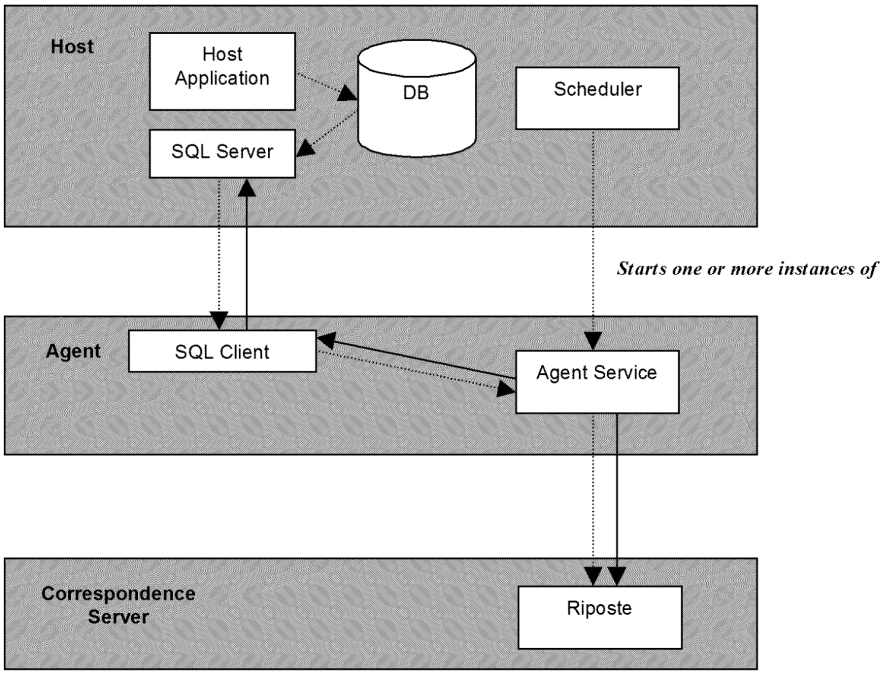


Figure 7-6 Structure of a bulk loader agent for an SQL Host

The Host assembles the records to be loaded and a number of agents are then started in parallel by the scheduler. When all the agents have completed their processing, a check process verifies that all the records have been successfully input to the Riposte message store.

Flat File bulk loader agents receive the flat files from the Host application and reads the records serially, passing them to Riposte. The structure is shown in Figure 7-7.

FUJ00001368

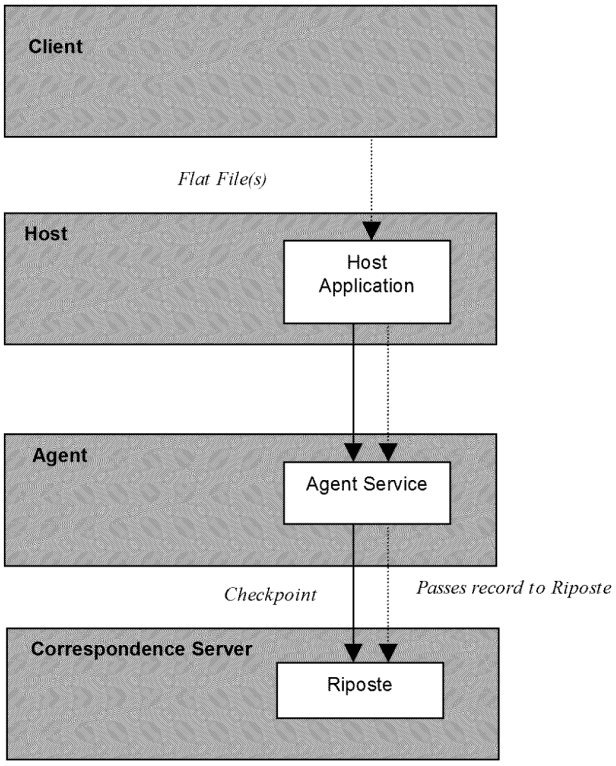


Figure 7-7 Flat File loader agents

7.2.3 Interactive agents

Interactive agents react to the appearance of a particular type of message.

7.2.3.1 Interactive loader agents

Non-SQL Hosts make a direct call to the agent as shown in Figure 7-8.

FUJ00001368

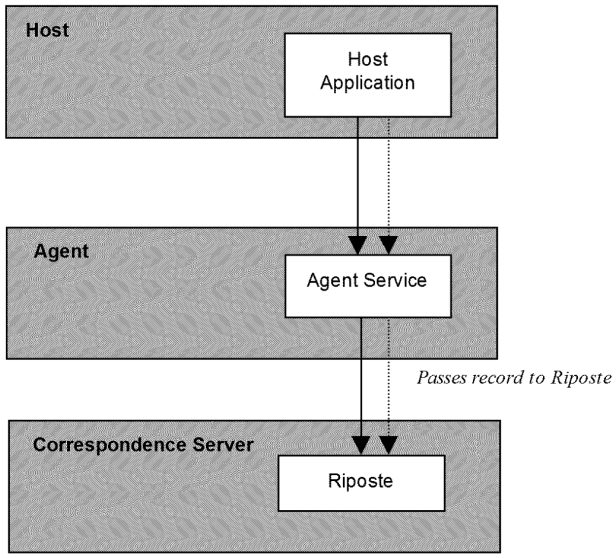


Figure 7-8 Non-SQL loader agents

If the Host application is implemented on an SQL database, and the triggering event is the application writing a record to the database, a mechanism like the one shown in Figure 7-9 is used.

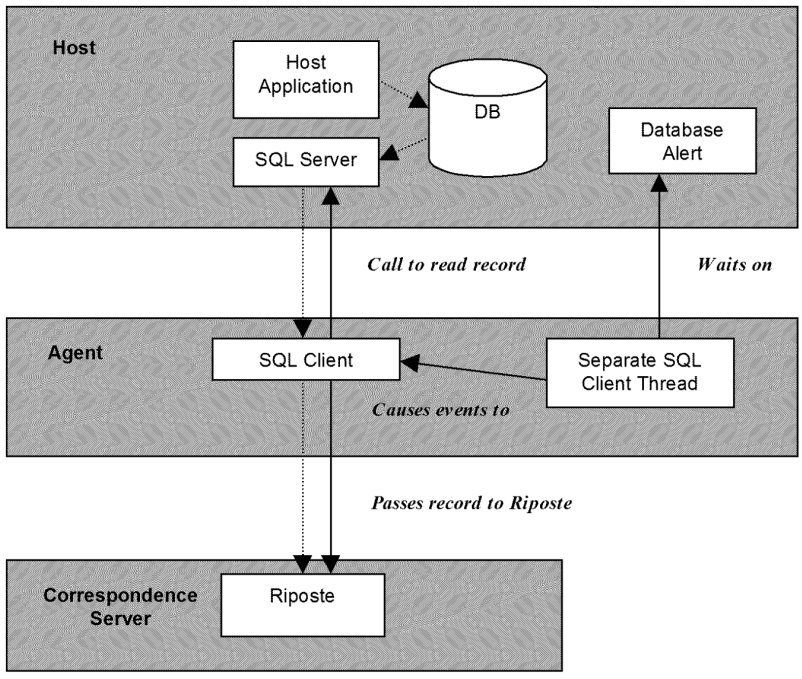


Figure 7-9 SQL Driven interactive loader agents

When the application writes a record to a table in the Host application's database, a notification is generated. This notification causes an event to

FUJ00001368

the SQL client, causing the client to read the record from the SQL server. The interactive agent calls the SQL client to read the record and process it, including passing it to the Riposte message store using the Riposte Application Programming Interface.

7.2.3.2 Interactive harvester agents

An SQL based harvester is shown in Figure 7-10. It runs at designated times of the working day, listening for the arrival of Riposte messages of the type it is designed to harvest. Such messages are then processed and the results written to the SQL database's Client Interface Table by a direct call to the SQL Client.

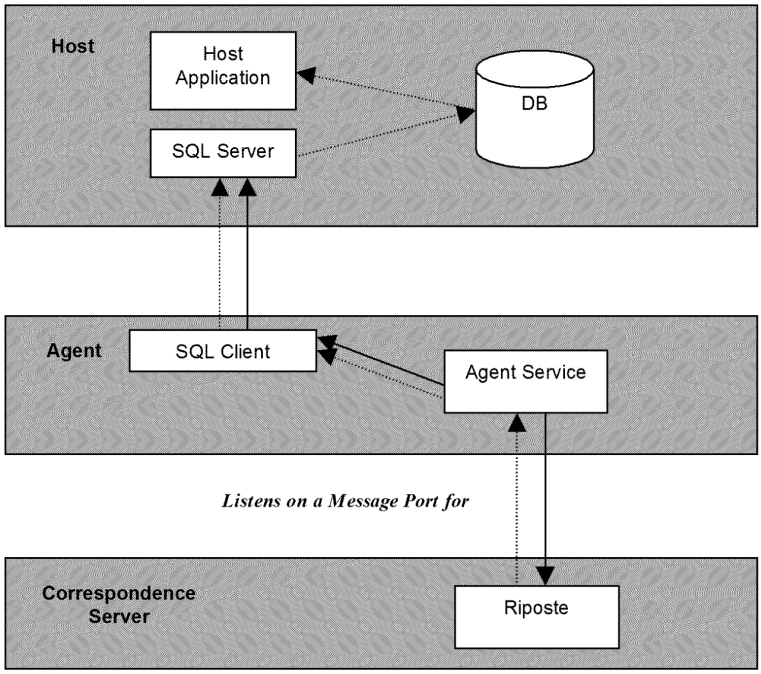


Figure 7-10 Structure of an SQL-based interactive harvester agent

For a non-SQL harvester the agent makes direct calls to the Host, as shown in Figure 7-11.

FUJ00001368

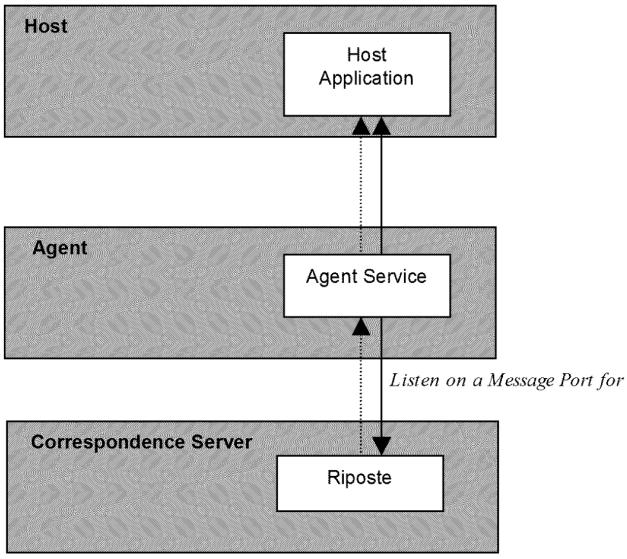


Figure 7-11 Non-SQL interactive harvester agents

7.2.4 Enquiry agents

Enquiry agents are invoked by a counter application and acquire data and pass it back to the application.

The counter application creates a Riposte Priority Message that includes the enquiry (or update) and its message ID. It next creates a message port waiting for a response. The message port time out should be set by a configuration parameter set in a persistent object. Real Time Message Ports allow Priority Messages to be processed immediately rather than waiting for the message to arrive in sequence. Delivery through a Real Time Message Port is not guaranteed, and duplicates may occur, so agents using Real Time Message Ports must be designed with this in mind.

7.3 Interface constraints

The constraints that need to be accommodated in the design of an Agent are presented in the following sections, the general constraints first then those more specific to particular types of Agent: Bulk, Interactive and Enquiry.

7.3.1 General

1. Transactional Integrity must be maintained. This is the preservation of all messages that comprise a single transaction as a single entity for processing. Agents have to ensure that Riposte messages are never lost, though they may be duplicated.

FUJ00001368

2. Agents must support multi-threading or multiple instances to optimise the use of the limited number of agent servers.
3. If two or more agents require access to the same set of transactions within Riposte, they must obey the principle of Vertical Application separation, and each must retrieve the transactions for their own use.
4. Data can be harvested twice by the same application.
5. All agents must have an application interface specification, defining the nature of interfaces both in attribute grammar and host syntax
6. If a new application adds a new attribute to any message type that is harvested by the TPS harvester agent for input to TIP and the Data Warehouse, the impact of adding the new attribute must be highlighted. The TIP AIS is described in *Pathway to TIP AIS.*
7. Monitoring and managing progress when processing SQL records: Either by marking individual or groups of rows as processed in the Client Interface Tables; or by maintaining a 'progress marker' in separate control tables. The progress information must be updated before issuing a 'Commit'. Riposte transactions must also be completed before the 'Commit'.
8. Fetch units: (these are collections of records retrieved by an SQL*net call. The size of a fetch unit depends on the optimum number of records that can be retrieved and the associated memory available in the Agent Server, and is an example of the kind of parameter that must be specified in the Application's AIS.
9. Checkpoints: these are collections of markers applied to the whole message store (to all Correspondence servers in a cluster). An agent that fails is restarted by Tivoli – the agent must restart from the latest message store Checkpoint – thus ensuring that all relevant messages are processed.

   Note that there are some conditions that Tivoli cannot detect (a looping agent, for example).

   Checkpoints are expensive to write and so should not be written too frequently.
10. Recovery – agents must manage the recovery and message resynchronisation between the Host layer and the Counter Layer following a failure.

    Riposte contains transaction features that allow multiple messages to be committed as one atomic unit – thus the messages are replicated as a unit and cannot be processed individually until they have all been replicated.

FUJ00001368

11. General Recovery rules:
    - Applications must be able to detect duplicate messages and ignore the duplicate.
    - Applications must guarantee not to lose messages, though they can duplicate them.

7.3.2 Bulk agent constraints

1. Bulk agents run continuously and Maestro is configured to initiate their processing when needed.
2. Bulk agents mark work chunks as 'in progress' as soon as they start to process them – if there is a failure, the bulk Monitor detects it and initiates a recovery process to recover the failed work chunks.

7.3.3 Interactive agent constraints

1. For SQL driven interactive loader agents, alerts may be lost occasionally, but to ensure that no records are missed, the SQL Client also reads the table on a regular time schedule.
2. Non-SQL interactive harvester agents: because clusters are essentially discrete message stores, one such harvester must run in each cluster. Clusters are groups of correspondence servers, organised in this way for performance reasons. Normally only one instance of a harvester can run in a cluster at a time. For more information about clusters in Horizon, see *TMS Architecture Specification*.

7.3.4 Enquiry agent constraints

1. Enquiry agents: if these agents need to go outside to an external source to gather data, they must set a time-out period that is just longer than the SLA agreed with the service provider. In order to handle line failures, the timeout must be significantly less than that of the counter.
2. The current ICL Pathway policy is that Enquiry agents run under Windows NT; their failure is detected by Tivoli, which will restart them.

FUJ00001368

8 Systems Management

This section describes the following topics:

- **Reference Data:** this section describes how Reference Data is accessed, the temporal nature of such Reference Data and the process used to maintain such data.
- **Event reporting**: this section describes the application interfaces to be used for event reporting, including the reporting of exception conditions.
- **Software packaging**: this section describes how software comprising new applications is to be handed over to ICL Pathway for system integration testing, and the documentation needed to support such handovers. It gives an overview of the system and integration process and subsequent processes leading to implementation of the new application.
- **Diagnostics:** this section describes the requirement to make diagnostic information available so that errors can be interrogated centrally.

8.1 Reference Data

Reference Data can be received from POCL in the form of a Reference Data element following an associated Operational Business Change (OBC) Form, which gives prior notice formally authorising the change. Note that those changes classed as 'Basic' do not require prior authorisation via an OBC form; examples of basic changes are changes to product prices, product names, and certain business rules (whether a product is voidable or reversible, for example).

Type A Reference Data is received from the POCL Reference Data System (RDS) as a data file in a format conforming to the *POCL Application Interface Specification Reference Data to Pathway*, and is automatically input to the ICL Pathway Reference Data Management Centre (RDMC).

Type B Reference Data follows a similar route, but is subject to manual intervention before it is released into the live estate.

Types C and D Implementation Reference Data are dealt with as Work Packages.

An explanation of the various types of Reference Data, including the types required to support an application, is given in section 3.1.2.

For further details see Appendix C, *System Management*.

FUJ00001368

8.1.1 Change Control

Most of the change control process is outside RDMC with Customer Services entering details of change requests into RDMC once a change has been agreed. The actual change reference is generated outside RDMC. Reference Data can then be associated with the change reference. This Reference Data is held within RDMC and is not visible to RDDS until Customer Services release the change reference into the RDDS environments.

8.1.2 Classes of changes

The Reference Data changes are classified as follows:

1. Class 1 Standard Reference Data changes – These changes require only data validation. They do not invoke the Change Control process and thus can be implemented rapidly.
2. Class 2 Business Rules changes – These are subject to Change Control as they affect ICL Pathway's levels of service. They are subject to data validation.
3. Class 3 Logistically Constrained changes – These are subject to Change Control since they incur logistical constraints such as installation of equipment.
4. Class 4 Extended Functionality – These changes are subject to Change Control, data validation and testing.
5. Class 5 New Functionality - These changes are subject to Change Control and code changes as well as data validation and testing.

Class 1 changes are handled on their own, whereas all others have to be associated with the relevant Reference Data Change Request that came from POCL. This association is performed by the POCL Reformat process, which inserts the change reference and data class into each data line requiring them.

8.2 Event reporting

Tivoli Event Management is used to gather information about the past and present behaviour of the managed components, and pass this information back to the systems management centre. It has two principal components: *Event Server* that collects and assesses events, and *Event Console*, which displays the status of monitored nodes to an operator.

Within the OPS/TMS environment, events originate from one of two sources:

- NT Event Log (from all NT processors)
- Tivoli Distributed Monitoring (this is a sentry program which monitors managed nodes, including counter PCs).

FUJ00001368

All events are archived.

8.3 Software packaging

ICL Pathway software is assembled using a hierarchy of components defined by the Product Breakdown Structure. In ascending order in the hierarchy, these are:

- Configuration items, items, are the lowest level component
- Design parts
- Workset
- Baseline
- Release

There is a fuller explanation in Appendix C, *System Management*.

8.3.1 Outline

An application can comprise elements that are:

- Bespoke software
- Tailored commercial off the shelf products
- Untailored commercial off the shelf products

These last two items are included as 'Common Product Set' in Figure 8-1.

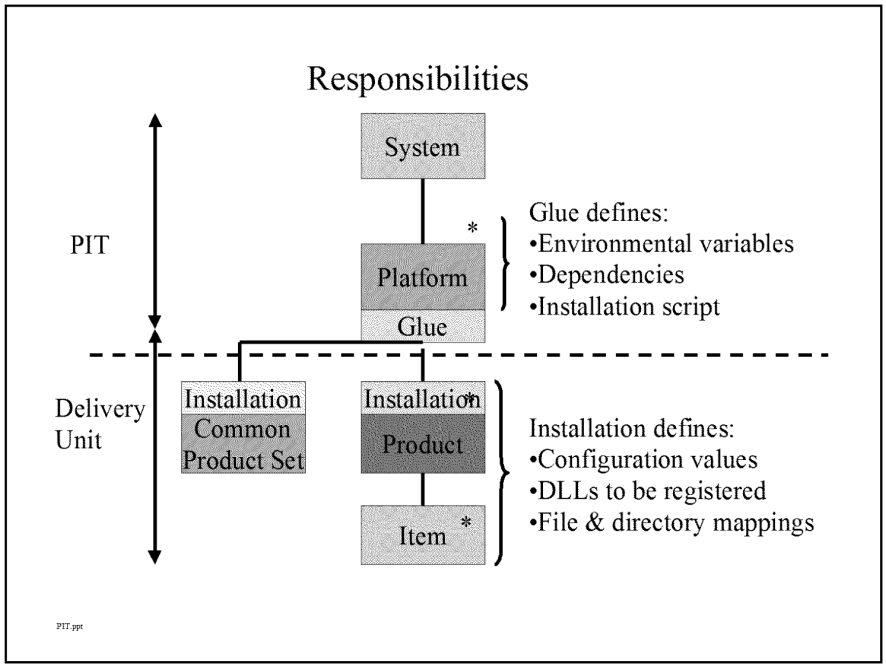


Figure 8-1 Application build process

FUJ00001368

As illustrated in the above diagram, it is the responsibility of the delivery unit, whether internal to ICL Pathway or external, to supply not only the items grouped into products, but all the information and values required. This is the 'installation' in the diagram, and the tools and processes of Configuration Management, CM, enforce its provision. The main CM tool is PVCS, from Merant.

FUJ00001368

8.3.2 Development steps

The following is necessary in order for a development group to deliver software to ICL Pathway:

- Register Configuration Items in the PBS

  For a new product on a new platform that has not already been set up within PVCS, there must be an appropriate Physical Design document that defines all the physical components involved. Physical Design documents cover hardware in terms of the capacity of CPU, memory and discs, and additional components such as Ethernet cards. Software is defined in terms of the release of NT software required, the Service Packs that apply, any drivers required for additional cards and the dependencies on security software, Riposte, Tivoli and auditing and archive.

- Submit Configuration Items to PVCS

  The files placed in PVCS must have attributes entered correctly; where material is supplied to reproduce an executable, it must be supplied in the form of a ZIP or TAR file.

  Each delivery from development to PIT must detail the content of the delivery and any dependencies together with installation instructions.

  The files are actioned to ensure they are ready for delivery to PIT, and are captured in the product baseline.

- Create product baseline

  PIT requires a full product release, containing the full set of files required for the product. A baseline is produced from within the appropriate workset, and against the design part for the product.

  During the creation of a product baseline the attributes must be set to indicate the target release for the product, and to identify to which platforms the new version of the product should be applied.

- Handover to PIT

  Once the product baseline is created, all suitable attributes are set, and the contents verified, the baseline is actioned to the next state. This automatically places the baseline in the pending list for each PIT user.

- Handover to live

  This is addressed in Appendix C: *System Management.*

FUJ00001368

8.4 Diagnostics

It is necessary to make diagnostic information available for interrogation, wherever an error occurs on the distributed estate. The information created by the application to aid diagnosis must be recorded on the node on which the error occurs, in such a way that it can be interrogated centrally. For example, it is important that each DLL records its version number when it is loaded, as it is possible that not all outlets, or nodes within an outlet, will be running the same software version.

How diagnostic information should be provided is described in Appendix C, *System Management.*

FUJ00001368

9 Naming Standards

This section defines the naming standards for the following entities:

- Applications and their components
- Worksets
- Configuration items
- Transient messages
- Persistent objects
- Attributes
- Events

9.1 Basic rules

The name of any entity must be unique; if an application uses persistent objects, it must use a unique set of collection names.

A balance must be maintained between performance, including storage space occupied, and comprehension of a name. Names should be as short as possible while preserving clarity. For this reason ICL Pathway requires that application names be abbreviated to three characters and that ICL Pathway allocates them.

Capitals are used for application names, with the next level of entity being expressed in mixed case.

9.2 Applications and components

From a configuration management viewpoint, an application and its components are collected into *worksets*, allowing for multiple development streams, each workset holding a complete set of the items needed for a release.

As development proceeds, the contents of the workset are *baselined*, to give a frozen 'snapshot' of the state of the items at a point in time. Figure 9-1, below, shows the cycle in which items from development are incorporated into a baseline, which is then tested. As development proceeds and further items are added, the previous baseline is exported to a new workset, forming a new baseline which is tested.

When development of the application has reached a suitable stage, after testing the baseline a release is built which can be deployed operationally. Figure 9-2 below shows this part of the process.

FUJ00001368

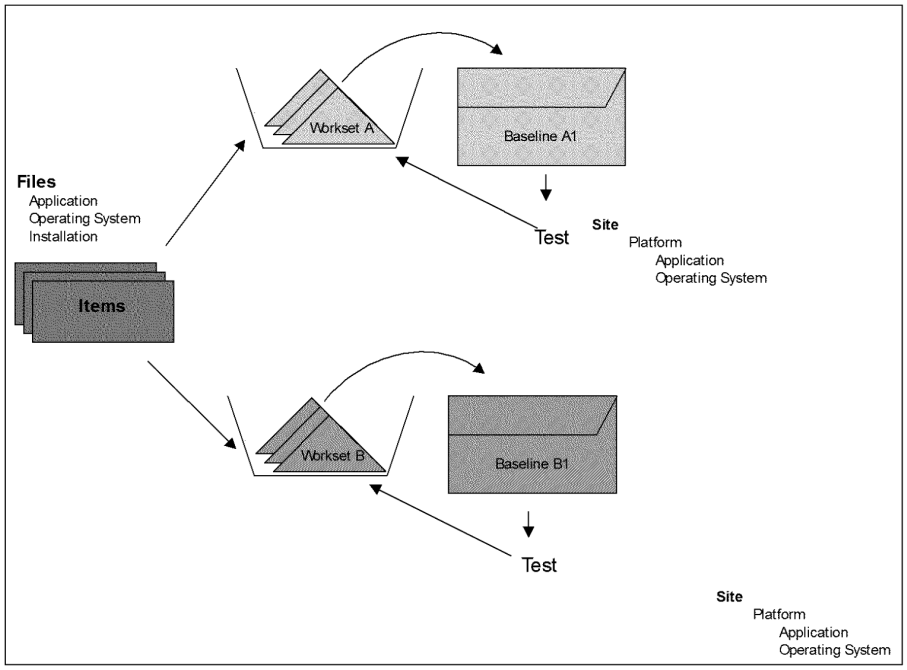


Figure 9-1 Worksets, baseline, baseline test cycle

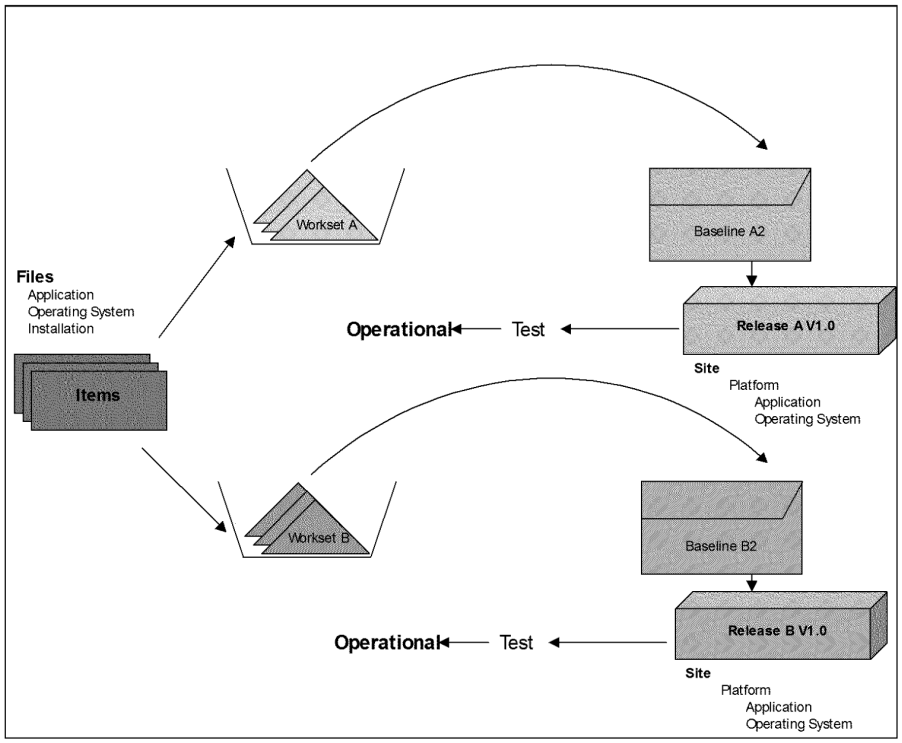


Figure 9-2 Workset, baseline, release to Operational process

FUJ00001368

9.3 Worksets

Work sets have the following naming conventions.

Workset	*Definition*	*Example*
Application	*Application short name_target version*	ABC_1_6
Platform Build: Normal development/ testing stream	*Platform_Platform Version_Bnnn*	COUNTER_2_0_B001
Branched development/ testing streams	*Platform_Platform Version_Bnnn*	COUNTER_MR2_B001
Fastrack: Normal development/ testing stream	*FSTK_ReleaseNos_WPnnnn*	FSTK_2_0_WP1992
Branched development/ testing streams	*BranchID_WPnnnn*	TEST1_WP1992

Table 9-1 Workset names

nnn or *nnnn* are unique numbers.

Platform Build worksets are collections of the components for the build of a particular platform, and the Fastrack worksets are used for urgent changes.

FUJ00001368

9.4 Naming major configuration items

This table shows, for each entity, the format of the ID, fuller ID or Part Variant, and the full description of the entity. For each entity the first row shows the general case, and the second an example.

For any Release, there can be more that one Site specific component, and within a Site there may be more than one Platform (PC or server).

Within a Platform the components are typically Applications and Operating Systems.

For each entity, such as a Release, there is an ID, for which the Part Variant is a more specific attribute, such as V2.0 in the example shown.

Entity	*Part ID*	*Part Variant*	*Part Description*	*PCS*	*Father Variant*
Release	*Release*	ICL Pathway Release Version Number	Full meaningful description of the part, including the version number	Left at the value of 1	Left at the value of A
Release Example	RELEASE	2_0	Release 2.0	1	
Site	*Site name*	ICL Pathway site Version Number	Full meaningful description of the part, including the version number	Left at the value of 1	Left at the value of A
Site example	POST_OFFICE	1	Post Office Single Counter Version 1	1	A
Platform	*Platform name*	Platform Version Number	Full meaningful description of the part, including the version number	Left at the value of 1	Left at the value of A
Platform example	PO_MULTI_COUNTER	1	Post Office Multi Counter Version 1	1	A

FUJ00001368

Application	*Application short name*	Target Version Number	Full meaningful description of the application, including the version number	Left at the value of 1	Left at the value of A
Application example	ABC	1_5	ABC Application Version 1.5	1	A
Operating System	*Operating System_Version*	Used to denote service or maintenance packs applied	Full meaningful description of the part, including the version number	Left at the value of 1	Left at the value of A
Operating System example	NT_4_0	1	NT 4.0 - no service packs applied	1	A

Table 9-2 Major configuration items: entity names

For items, there are other components of the name: Type, Library filename, User filename and Workset filename.

	Item ID	*Item Description*	*Type*	*Variant*	*Library filename*	*User filename*	*Workset filename*
Item	*Short name_nnnn,* including leading zeroes	Full meaningful description of the item, excluding the version number	Select from pre-defined list	Left at the value of A	**Product_Real file name** (all in lower case)	real file name (in required case)	relative directory for the file when extracted plus the filename (in required case) Note: the lack of leading /
Item example	ABC_0004	ABC account verification routine	INSTOBJECT	Left at the value of A	**abc_abcveri.dll or abc/abcveri.dll**	AbcVeri.dll	abc/AbcVeri.dll

Table 9-2 Major configuration items: item names

FUJ00001368

9.4.1 Defining an application

An application, developed by either ICL Pathway or by an external supplier, must adhere to the standards specified in Table 9-2.

The target version number is the number allocated to the specific version of an application. That version of the application is delivered on to a specific version of a platform for a given release of ICL Pathway. Thus the application is delivered against the target version number. This allows a release to be defined from the outset.

There is an attribute that can denote a project team's internal application version number or a supplier version number.

9.5 Transient messages

These are data records created to support EPOSS and TIP and any other Client. They are archived after an expiry period, specified by the Expiry attribute, and determined by the Clients business rules. Transient Messages have the following generic structure:

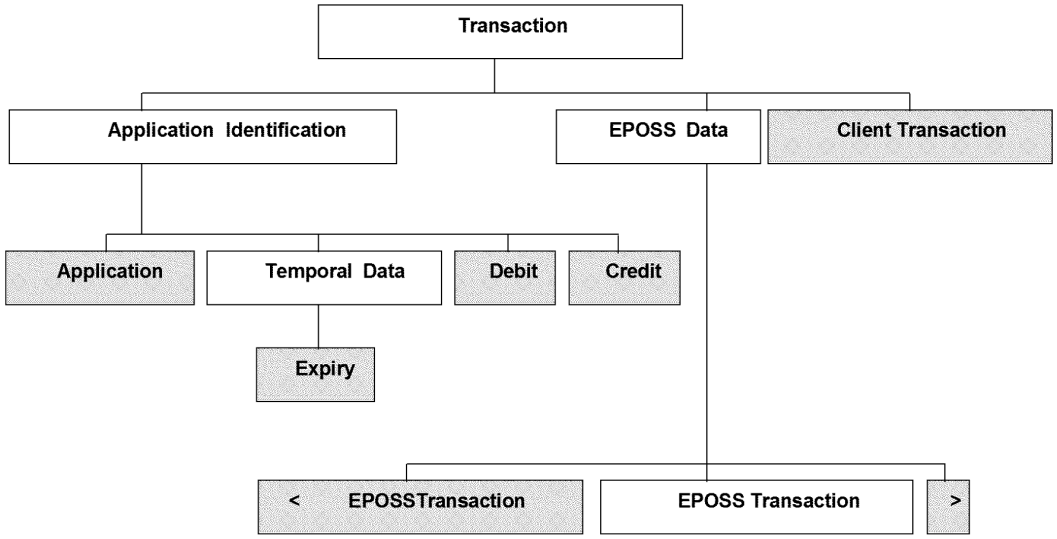


Figure 9-3 Transient message

The naming convention for the Application Identification is to use three upper case characters: ABC or APS, for example. Note however, that although historically some applications have four characters, the rule is now to use three.

FUJ00001368

There are essentially three classes of Transient Messages:

- **Transactions** – monetary
- **Administration** – Non-monetary, office facing that must be recorded. An example is the receipt by the office of a benefit order book, or cheque book.
- **Event** – Non-monetary (with respect to a Stock Unit), customer facing that must be recorded. An example might be a payment split across several Client accounts.

Type	*Generic name*	*Example*
Transaction	*APPTransaction*	OBCSTransaction
Administration	*APPAdmin*	APSAdmin
Event	*APPEvent*	ABCEvent

9.6 Persistent objects

These are records that have a start date and an end date. The number of persistent objects that an application uses should be kept to a minimum. This is because the more persistent objects, the more adverse the effect on the performance of the system.

9.6.1 Collection name

The main collections that are used by an application are:

- Products
- Error Messages
- Tokens
- Training Data
- Training Messages

Training data and messages need to be considered separately. ICL Pathway should be consulted for the rules covering their creation.

The collection name is formed from the application name, abbreviated to three upper case characters, followed by an abbreviated form of the type of collection. Example: ABCTkns.

Table 9-3 lists the main types of collections, giving an example collection name together with a definition of the identifier for the items that make up the collection.

FUJ00001368

Type	*Generic name*	*Example*	*Item Id*	*Definition owner*
Products	*APPProducts*	APSProducts	*nnnnn*	ICL Pathway defined
Error Messages	*APPErrMsgs*	ABCErrMsgs	Text, abbreviation of message content	Application
Tokens	*APPTkns*	ABCTkns	Text, abbreviation of token description	ICL Pathway defined
Training Data	*APPTData*	ABCTData	Text, abbreviation of data description	Application
Training Messages	*APPTMsgs*	ABCTMsgs	Text, abbreviation of message content	Application
Reports	*APPReports*	ABCReports	*nnnnn*	ICL Pathway defined 5-digit number
Cash Account Mappings	*CAMappings*			
Interface Items	*InterfaceItems*			

Table 9-3 Collection names

There are also system collections such as events, which defines the content of the Event Log reports (for details see *Horizon OPS Reports and Receipts*), and various EPOSS collections defining the use of such things as Scales, and the Impulses from input devices.

9.7 Attributes

The naming of attributes within messages has to reflect two conflicting requirements:

- The need to be able to easily identify the attribute when diagnosing errors
- The size overhead implications of long attribute names.

The conventions for assigning attribute names can be summarised as follows:

- Names are made up of one or more elements, where each element supports the qualification of the identification of the element. For example, the Locked attribute identifies whether or not a stock unit is locked and LockedBy identifies the user who locked it.
- Capitals are used at the start of each element of the name (LockedBy and StockRootNode, for example).

FUJ00001368

- Abbreviations are used where they are meaningful (Txn in TxnId and TxnData, for example).
- Consistency of abbreviations should be maintained, as shown in the previous example.
- Obvious abbreviations, such as TID for token identity, should be used where possible, to keep names short. In these abbreviations, all characters are capitalised.

9.8 Events

All applications events must be recorded in the NT Application Event Log. Recorded details about the event must include:

- Its severity.
- The name of the application that generated it.
- The version of the application code involved.

It is also necessary for each application to be capable of being managed remotely in terms of restart after an error, or installation of a new version of the application.

More information on how events are recorded, the constraints imposed on event management by a distributed remote estate, and the rules associated with restarting applications is given in Appendix C, *System Management*.