

SMART

Business Objects Developer's Guide



Foreword

STATEMENT OF CONFIDENTIALITY

CSC has prepared this document in good faith. Many factors outside CSC's current knowledge or control affect the recipient's needs and project plans and errors in the document will be corrected once discovered by CSC. The responsibility lies with the recipient to evaluate the document for applicability.

The information in this document is proprietary, confidential and an unpublished work and is provided upon recipient's covenant to keep such information confidential. Personal Data supplied in this document may not be used for any purpose other than for which it was supplied. Personal Data may not be transferred to other parties without the prior consent of CSC. In no event may this information be supplied to third parties without CSC's consent.

The following notice shall be reproduced on any copies permitted to be made:

© CSC Corporation Limited 2013 All rights reserved

Any request for further information concerning this document should be addressed to:

Industry Software & Solutions (IS&S) Helpdesk
Royal Pavilion
Wellesley Road, Aldershot
Hampshire, GU11 1PZ
United Kingdom

Direct Line: +44 (0)1252 536333

Fax: +44 (0)1252 534022

Email: fssupport@csc.com

CSC Corporation Limited is registered in England under number 1812179, registered office Royal Pavilion, Wellesley Road, Aldershot, Hampshire, GU11 1PZ

Table of Contents

RELATED PUBLICATIONS	1
TRADEMARK ACKNOWLEDGEMENTS.....	1
PUBLICATION HISTORY	1
1. PURPOSE OF THIS DOCUMENT	2
2. FUNCTIONAL OVERVIEW	4
2.1. SMART RELEASE 9509	4
2.1.1. Design Requirements.....	4
2.1.2. Screen IO Module Inversion	4
2.2. DEFINITIONS	10
2.2.1. Client.....	10
2.2.2. Server.....	10
2.2.3. Business Object.....	10
2.2.4. Business Method	11
2.2.5. Business Object Handler - BOH.....	11
2.2.6. Request message.....	12
2.2.7. Response message.....	12
2.2.8. Message Identifier	12
2.2.9. Application Software	13
2.3. STRUCTURES	14
2.3.1. Leader Header Information	14
2.3.2. Message Header.....	16
2.3.3. Message Data.....	17
2.4. BUSINESS OBJECT CODE STRUCTURE.....	18
2.5. ERROR HANDLING	18
2.5.1. Handling System Errors	18
2.5.2. Handling Communications Errors.....	18
2.5.3. SFERRREC - Fatal Error.....	19
2.5.4. Handling Editing Errors.....	20
2.5.5. BOVERRREC - Validation Error.....	20
2.5.6. Handling Escape Messages	21
2.5.7. Paging and Scrolling	21
2.6. SMART BUSINESS OBJECTS.....	22
2.6.1. Session Object: SESSION	22
2.6.2. Fatal Error Object: SFERR	24
2.6.3. Validation Error Object: SVERR.....	25
2.7. NOTES ON THE BUSINESS OBJECT HANDLER	26
2.7.1. Linkage	26
2.7.2. Starting a Session	26
3. THE BUSINESS OBJECT GENERATOR.....	28
3.1. DSNBOBJ - DESIGN BUSINESS OBJECT	29
3.1.1. Parameters	29
3.1.2. S0200 – Work With Business Object Definitions.....	29
3.1.3. S0201 – B/O Definition (Maintenance).....	30
3.1.4. S0205 – Delete B/O Definition	31
3.1.5. S0202 – Work With B/O Method Definitions.....	33
3.1.6. S0203 – B/O Method Definitions Program List.....	34
3.1.7. S0204 – B/O Method Definitions Field List	36
3.1.8. S0638 – B/O Method Definitions Additional Copybook Fields.....	38
3.1.9. S0634 – B/O Method Definition - Field Mapping.....	40
3.1.10. S0206 – B/O Definition – Insert Source.....	41
3.1.11. S0207 – B/O Definition – Insert Copybook Source.....	43

3.1.12.	S0211 – Auto Tag Inserted Source	44
3.1.13.	S0212 – Delete Inserted Source	46
3.2.	CRTBOBJ - CREATE BUSINESS OBJECT.....	47
3.2.1.	Parameters	47
3.2.2.	Source Members Generated	49
3.2.3.	Software Management.....	51
3.2.4.	Summary	51
3.2.5.	Generated Business Object Example	53
3.2.6.	Development Utilities – Field / XML Tag Mapping	70
3.2.7.	Data Model.....	72
3.3.	TSTBOBJ.....	73
3.3.1.	Parameters	73
3.3.2.	Input and Output files used by TSTBOBJ.....	74
4.	BUSINESS OBJECT DESIGN/BUILD CONSIDERATIONS	76
4.1.	CONVERSION.....	76
4.2.	DRIVER RUN-TIME SUPPORT	78
4.3.	STANDARDS	78
4.4.	USING THE BUSINESS OBJECT GENERATOR	79
4.5.	THE FRONT-END SYSTEM	80
4.6.	BUSINESS OBJECTS	81
4.6.1.	Standard Business Object	81
4.6.2.	Super Business Objects	81
4.7.	WHAT TO AVOID.....	82
4.8.	NOTES ON SUBFILE PROCESSING	83
4.8.1.	Retrieving Data from a Subfile Screen.....	83
4.8.2.	Updating a Subfile Screen.....	83
4.9.	NOTES ON ERROR TRAPPING/HANDLING	84
4.9.1.	Retrieving Subfile Errors from a Business Object.....	84
4.9.2.	Use of ERROR-INDICATORS OF VALIDATION-ERROR	84
4.9.3.	Use of SVERR	84
4.10.	NOTES ON THE USE OF BUSINESS OBJECTS IN A DIARY SETTING	85
4.11.	DATA CONSIDERATIONS.....	85
4.11.1.	Front-end Database.....	85
4.11.2.	Server Database	85
4.12.	PROCESSING INBOUND MESSAGES OVER 32K LONG	86
4.12.1.	Overview.....	86
4.12.2.	Implementation	86
4.12.3.	BOHIO	87
4.13.	MISCELLANEOUS.....	89
4.13.1.	WSSP-COMMON-AREA.....	89
5.	BUSINESS OBJECT TEST TOOL	90
5.1.	BOTT INTRODUCTION.....	90
5.2.	OVERVIEW	91
5.2.1.	Online Subsystem	91
5.2.2.	Business Object Test Data Admin Submenu	92
5.2.3.	Work with Business Object Test Data.....	93
5.2.4.	Create a Business Object Test Member	94
5.2.5.	Modify a Business Object Test Member	97
5.2.6.	Copy a Business Object Test Member.....	100
5.2.7.	Delete a Business Object Test Member	102
5.2.8.	Enquire on a Business Object Test Member	103
5.2.9.	Print a Business Object Test Member.....	103
5.2.10.	Run a Business Object Member Test	106
5.2.11.	View Business Object Test Output	108
5.2.12.	Print Business Object Test Output	109
5.2.13.	View the Business Object Member Copybook	111
5.2.14.	Print Business Object Member Copybook.....	111

5.2.15.	<i>Print the Business Object Member Test Data</i>	<i>113</i>
5.3.	CONFIGURATION	115
5.3.1.	<i>Software Defaults</i>	<i>115</i>
5.3.2.	<i>File Definitions</i>	<i>115</i>
6.	APPENDIX 1 - BOHMQMON	116
6.1.	EXCHANGE OF MESSAGES	116
6.1.1.	<i>Messages Life Cycle</i>	<i>116</i>
6.1.2.	<i>Explanation</i>	<i>117</i>
6.1.3.	<i>BOHMQMON in Detail</i>	<i>118</i>
6.2.	MQSERIES FACILITIES USED	122
6.2.1.	<i>Reply-to-queue Mechanism</i>	<i>122</i>
6.2.2.	<i>MQPUT1 versus MQPUT</i>	<i>122</i>
6.2.3.	<i>Request/Reply Message Correlation</i>	<i>122</i>
6.3.	OTHER ISSUES	123
6.3.1.	<i>Commitment Control</i>	<i>123</i>
7.	APPENDIX 2 – ONLBOH AND BCHBOH	125
7.1.	BUSINESS OBJECT HANDLER FOR ONLINE	125
7.2.	BUSINESS OBJECT HANDLER FOR BATCH	125
7.3.	TYPICAL USAGE OF ONLBOH AND BCHBOH	126

Related Publications

- SMART Developer's Guide

Trademark Acknowledgements

See <http://midrange.csc.com/trademarks.html>

Publication History

Date	Comments
April 2004	Initial Version
April 2005	Revised
July 2005	Revised
April 2006	Revised
April 2007	Revised and updated CSC and IBM product names
April 2008	Revised for SMART 0804. Business handlers for online and batch
Oct 2008	Revised for 0810
March 2009	Replaced term 'Attribute' with 'XML Tag'
April 2010	Revised for 1004
April 2011	Revised for 1104
April 2012	Revised for 1204
Aug 2012	Additional comments on the BOTT for company tests
June 2013	Revised for SMART R13.0

1. Purpose of this Document

This document describes how SMART-based applications may be enabled to operate within a Client-Server environment.

This could be in conjunction with a front-end GUI such as CSC's GenApp Framework or with a customer specific system, using the applications in an Intranet/Internet solution. Further, the Client-Server enablement also allows the applications to be used in a "services"-oriented structure, typically in conjunction with CSC's MSP host integration component, where the applications perform the role of service providers to the wider organisation.

This document aims to inform designers/developers of how Client-Server enabled SMART-based applications work, the tools available to perform the enablement and the best practises to be used in this process.

- This page is intentionally left blank –

2. Functional Overview

2.1. SMART Release 9509

The 9509 release of SMART was the first release to provide the option to use the SMART-based applications like POLISY and LiFE in a Client-Server environment. From then on a number of enhancements have been made available both in the running of the Client-Server enabled applications as well as the toolset to perform the enablement.

This chapter will explain how the Client-Server environment works for the SMART-based applications.

2.1.1. Design Requirements

In designing this solution, the following requirements have been taken into consideration:

- CSC wishes to retain and build on the investment made to date in its intellectual property.
- CSC wants to ensure that its Customers will also be able to retain the investment they have made in their versions of CSC software.
- CSC understands that business applications are no longer used as stand-alone solutions. More and more they are used in a "componentised" way where a variety of applications provide the necessary functionality to cover the business needs. To participate in this open environment, business applications like POLISY and LiFE need to become open systems by publishing a standard set of service requests and responses.
- CSC understands that third parties will develop or have already developed Client software - also known as Front End software - using a variety of tools and platforms. This architecture must be able to support various front-end applications.

2.1.2. Screen IO Module Inversion

Developers using SMART will understand that on-line transactions have always had a standard structure that separated the various phases of processing.

There are database IO modules that externalise database manipulation and therefore insulate the application from the selected DBMS.

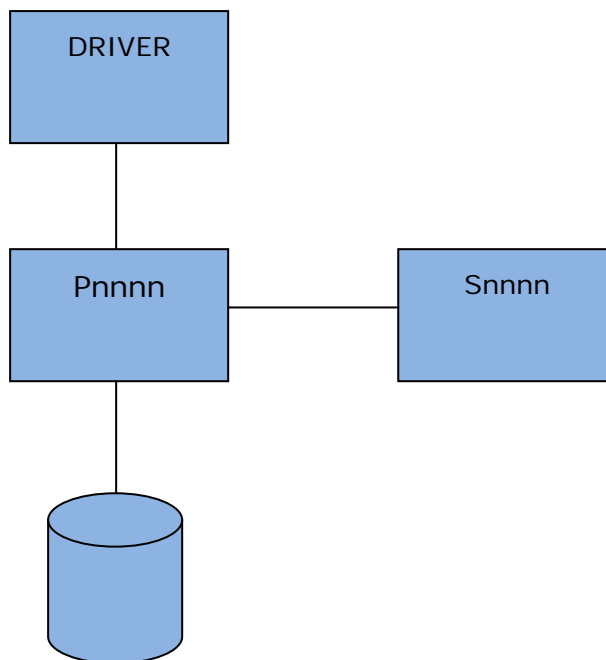
There are also screen IO modules that have served to insulate the application from the details of the screen presentation handling software.

Program mainlines are of a standard structure which can allow logic to be located in certain sections of the program.

Applications that follow these SMART standards are well poised to take advantage of fundamental changes to database, presentation and logic separation.

One such change is the Screen IO Module Inversion possibility that was introduced by the SMART 9509 Client-Server development.

To understand this inversion, an explanation is required of the way programs and screens traditionally interact with each other in the SMART-based applications. The following diagram shows this.

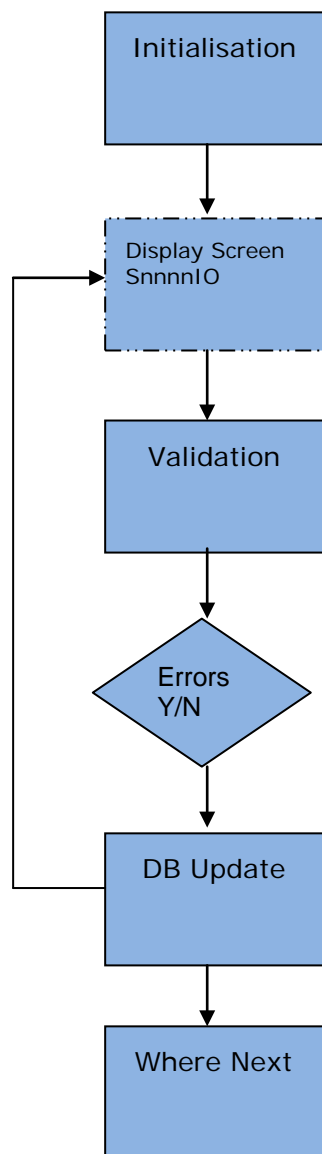


In this traditional scenario it works as follows when a program is called.

The user enters the system via the DRIVER command. From there, an online Pnnnn program is invoked, which will then display screen Snnnn to the user. The program pauses while the user completes a screen. Once finished, the user presses a function key or the <ENTER> key at which point the program resumes processing. A typical transaction in the system consists of a number of these programs & screens which the user will visit sequentially.

The MAINF copybook that is present in all Pnnnn programs directs the program flow. It is this copybook that dictates at what point screen Snnnn is displayed and what happens next.

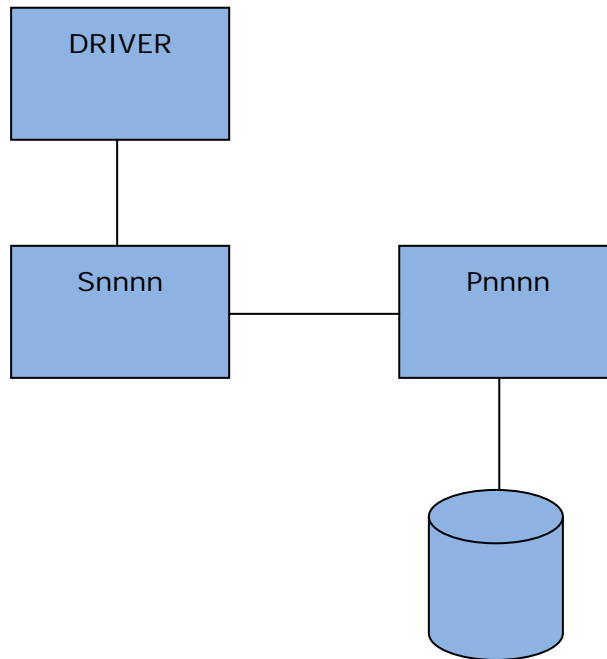
In short, MAINF ensures that the following happens in the Pnnnn program:



Programmers will recognise the 1000-, 2000-, 3000-, 4000- section of the online Pnnnn programs in the above diagram.

In a Client-Server environment, some external process or front-end system will invoke a program, possibly from a different machine as well. Therefore, it is impossible for the program to conduct a conversation in the way as described above.

The Screen IO Module Inversion is the first step in allowing another process to invoke a SMART based application program. As far as the traditional interface to the system is concerned the user will not see any difference. In diagrammatic form:



The inversion means that the program no longer manages the dialogue, as it is now the screen IO module that calls the mainline program. When using the system, the end user will see no difference, as the screen IO module will still display the screen to the user and once a function key or <ENTER> is used, the program will resume processing.

In the Client-Server enabled environment the MAINF copybook is no longer present in the Pnnnn program, instead a new MAING copybook drives the functionality.

MAING has a very simple structure, and it does the following:

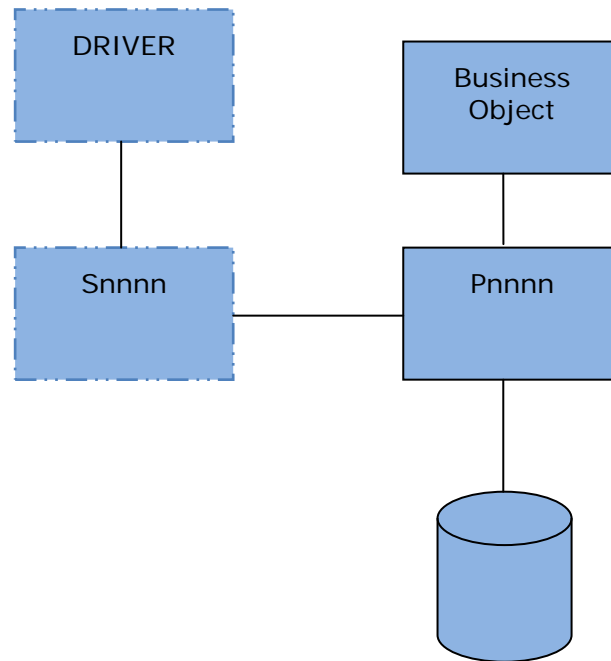
- if requested to perform the initialisation, it will perform the initialisation
- if requested to perform the validation, it will perform the validation
- if requested to perform the DB Update, it will perform the DB update
- if requested to perform the go to the next program, it will go to the next program.

Instead of being present in the Pnnnn program, the MAINF copybook is now located in the Snnnn screen IO module. This ensures that the program flow as described earlier is still followed when using the system through DRIVER, as MAINF in the Screen IO module will ensure that Pnnnn is called with the relevant function at the right time.

In short, inversion means that the program no longer calls the screen IO module, the screen IO module now calls the program!

In a Client-Server enabled environment this inversion is achieved by recompiling the Snnnn screen and associated Pnnnn. The replacement of MAINF by MAING is done automatically during the compile.

In itself this inversion doesn't have many benefits when used in the traditional way through DRIVER, however, the inversion allows the Snnnn programs to be replaced by a different driving process, called "Business Object" as shown below.



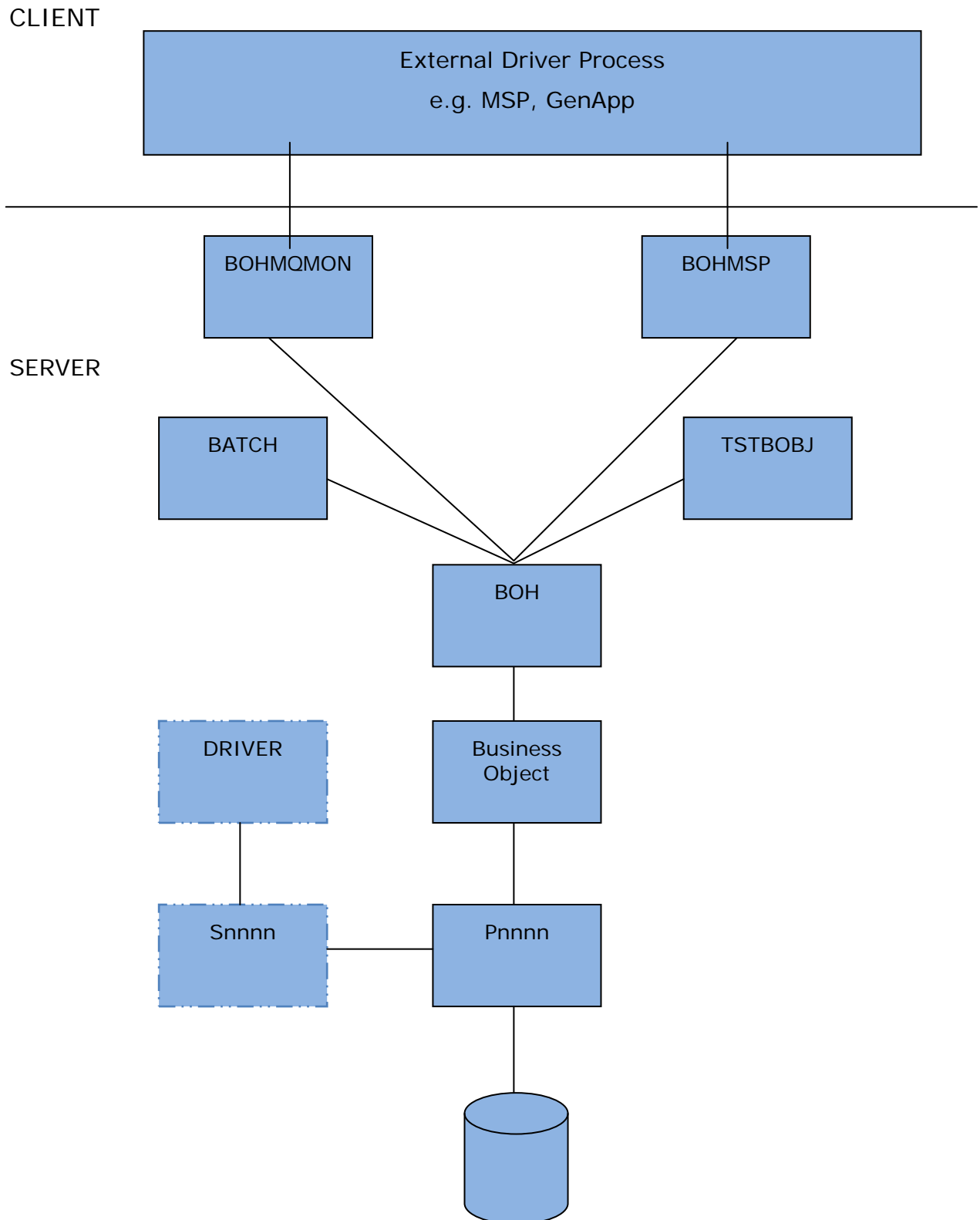
The main feature of this is that anything can now drive existing code. Possible 'drivers' are:

- GUI front-end
- Batch program
- DIARY
- Internet
- Intranet
- Extranet
- Agent laptop
- Interface from external system.

Examples of Business Objects used through such drivers are:

- Client functions from centralised CRM system
- Proposal creation from Quotation system
- Receipt creation from centralised receipting system.

The following diagram shows the Client Server environment:



2.2. Definitions

2.2.1. Client

The client (i.e. the external driver) requests a service provided by a server.

This client can be a workstation-based application that acts as a front-end, for example 3r – CSC's telemarketing and teleservicing solution, and AWD - the Automated Workflow Distributor.

It can also be a browser-based thin-client solution like CSC's Midrange Gateway (MG) and Midrange Service Processor (MSP) software.

2.2.2. Server

The server is the provider of a service to a client. For our purposes, the server will be a SMART-based application that acts as the back-end processor recording and managing data captured by workstations. Examples of servers include:

- LIFE CSC's life insurance solution
- POLISY CSC's non-life insurance solution
- FSU The Financial Services Umbrella.

2.2.3. Business Object

A Business Object is a software component which models the objects that an insurer would use. For example, this may be:

CLI	Client Details
POL	Policy Details
CLM	Claim Details.

The Business Object has defined within it the data structure of this object. For example, a Claim would contain the claim number, the date of loss and the various claimants attached to the claim.

In addition, the Business Object has defined within it a number of functions or methods that correspond to the actions one would like to have performed on the object. For example a claim object would require methods to Register, Modify and Pay. We call these functions Business Methods (or Business Verbs).

Typical Business methods may be:

CLI.CRT	Create Client Details
CLTPFO.RTV	Retrieve Client Portfolio

Note the convention of naming a Business Method as **Object.Method**.

Business Objects should operate as pure black boxes and should have no knowledge of who is calling them.

2.2.4. Business Method

The business method defines what function needs to be performed on the object. For example, the classic methods for most business objects include:

CRT	Create
DLT	Delete
ADD	Add
RMV	Remove
RTV	Retrieve
CHG	Change

Methods may use other business objects/methods. For example a complex or Super business object/method such as **POL.CRT** - Create Policy will be typically composed of other sub business objects/methods such as; **POLHDR.CRT** - Create Policy Header, **CLT.CRT** - Create Client Details, **LIFASS.ADD** - Add Life Assured and so on.

2.2.5. Business Object Handler - BOH

The Business Object Handler or BOH exists to separate and therefore insulate the business object from the various communications connectivity solutions. A number of communications options must be configurable; therefore, the BOH acts as the interface between the business object and the connectivity part of the solution.

The job of the BOH is to act as a common interface to the business objects by handling the request and response data and calling the business objects. The BOH caters for commitment control issues and error trapping/handling.

Different front-end packages and different communication protocols have different requirements for handling a job with its associated data on the system. To allow these different packages/protocols to interface with business objects (via the BOH), each of them will need a specific handler. These different handlers could be named BOHVM (for use with AWD/ViewManager), BOHJAVA (for use in conjunction with EE), BOHMSP (for use in conjunction with MSP), BOHMQMON (for use in conjunction with WebSphere MQ) etc. They will contain the product specific communication aspects, like wrapping and unwrapping the request and response data in a format that is specific for that particular product/communication protocol. They will call business objects via the BOH.

A number of these handlers exist as example programs and are delivered as part of FSU. See Appendix 1 - BOHMQMON for more information about the WebSphere MQ handler.

Three special (object only) business object handlers exists for use within the online system (ONLBOH), the batch system (BCHBOH) and the DIARY system (DRYBOH).

2.2.6. Request message

All exchanges between client and server follow a Request-Response pair pattern. A request message is a message sent to the business object. It contains the data arguments necessary for the business object to complete its function. A response message is sent back to the caller from the business object.

The conversation between the client and server could be designed such that a 'MORE' indicator needs to be used. Such a situation may arise when we have a large amount of data to send and we want the client to decide if more is required. For example, a large cash batch may contain dissections for hundreds of policies. Therefore, request messages may span a number of physical communications messages.

For programmers of traditional SMART-based applications, the concept is similar to a CALL and RETURN. The difference is that you may be calling a routine with different parameters depending on what you want the routine to do, and that you may receive different responses based on what happened within the routine.

2.2.7. Response message

A response message is a message returned from the business object to the client. It contains a completion status as well as other data the client may have asked for. In the case of an error the response message will contain details concerning the error(s).

A conversation could be designed so that a 'MORE' indicator be sent between the client and server. Such a situation may be useful when there is a large amount of data to send and it is necessary for the client to decide if more is required. For example, all clients with a common name may resolve to thousands of names.

2.2.8. Message Identifier

Message formats can be defined by a Message Identifier. Since each of the methods invoked in an object requires different data to complete, the business objects will be sent different request messages dependent on the method. Note too that some methods can share the same message format. For example, the **DLT** and **RTV** methods for the **CLT** object both require only the Client Number. The same message format can then be shared.

In addition, where an application has been enhanced and the object now has the capacity to perform extra functionality within a given method, an older workstation based application should still be able to invoke the business object using the old message format.

At a low level the message format may be thought of as a COBOL copy member.

2.2.9. Application Software

Since the business object is designed as a black box, it is irrelevant how the processing is performed behind the scenes. However, as mentioned previously, the client-server enablement means that all functionality defined within FSU, LiFE and POLISY can be used by business objects.

This will allow those CSC Customers with customised versions of the system to move forward and have their own versions of business objects developed quickly.

A business object generator exists which will capture the definitions of the message format, and map that definition to existing SMART-based applications. The generator will then produce various COBOL copybook members and code which will orchestrate the calling of existing COBOL mainline programs developed with SMART.

In future, applications could also be developed from the ground up using the client-server methodology. Such business objects will not require existing application software and, therefore, need not be generated.

2.3. Structures

This section describes how business objects communicate with each other and refers in particular to how this is implemented using COBOL. This section will only be of relevance to developers and designers of business objects.

Both the request and response message must conform to a standard structure so that the business object handler and the business objects can process the data. The structure has the following components:

- Leader Header
- Message Header
- Message Data

Each message will always contain these 3 parts. It is possible to compose complex messages that contain multiple instances of Message Data or even Message Header/Data parts, but that makes the processing of these messages very complex.

The recommendation is to use only one Message Header/Data pair and ensure there is a well designed structure present in the Message Data part.

2.3.1. Leader Header Information

The Leader Header precedes all information sent in the data between the client and the server. This information contains control data, for example it contains the name of the business object to be called and the method to invoke against that business object. The Leader Header is implemented through the LDRHDR copybook within SMART.

The format is as follows:

Offset	Attribute	Description	Type	Length
1	MsgRefNo	Unique message number. Allows for a request and reply correlation and audits. This number can be used by the client communications systems to uniquely identify each message. The key is UserId, WksId, MsgRefNo. If the value passed to the BOH is not numeric, this field will be defaulted to zeroes.	N	8
9	UsrPrf	User Profile. The sender's user identifier. This is the system user profile name.	AN	10
19	WksID	Workstation ID. The workstation name as configured on the system.	AN	10
29	ObjID	Object ID – The name of the object to which this message is being sent.	AN	10
39	VrbID	Method ID – The method to which the object will respond.	AN	10
49	TotMsgLng	Message Length. This is the length of the entire message including this LdrHdr. If the value passed to the BOH is not numeric, this field will be	N	5

Offset	Attribute	Description	Type	Length
54	OpMode	<p>defaulted to zeroes.</p> <p>Operation Mode.</p> <p>0 or 1 – Perform validation and update if no error detected.</p> <p>2 – Perform validation and update</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to 0.</p>	AN	1
55	CmtControl	<p>Commitment Control.</p> <p>N – Commitment control is started (needed for SMART-based applications to function), but no commits take place.</p> <p>Y – Start commitment control and commit updates if no errors are detected. Rollback changes if fatal or validation errors are detected.</p> <p>E – Commitment Control is externally started, commits and rollbacks are performed externally.</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to N.</p>	AN	1
56	RspMode	<p>Response Mode.</p> <p>I – Perform this transaction immediately and wait to return response message.</p> <p>D – Deferred. Log this message for asynchronous processing within AT.</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to I.</p>	AN	1
57	MsgIntent	<p>Message Intent</p> <p>R – Request Message.</p> <p>A – Reply Message.</p> <p>If the value passed to the BOH is not equal to R, this field will be defaulted to R.</p>	AN	1
58	More-Ind	<p>Message continuation.</p> <p>N – There is no continuation.</p> <p>Y – Subsequent data will be sent.</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to N.</p>	AN	1
59	ErrLvl	<p>Error Level. This is used only in response messages.</p> <p>0 – No errors detected.</p> <p>1 – Validation Errors have been detected - look further into the packet for more details.</p> <p>2 – Fatal Errors detected - look further into the packet for more details.</p>	AN	1
60	Ignore-DRIVER-Held	<p>Ignore situation that DRIVER is Held via the HLDDRIVER command and proceed with BO execution.</p> <p>N – Respect HLDDRIVER status and abort.</p> <p>Y – Ignore HLDDRIVER status and continue execution.</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to N.</p> <p>This parameter should be set to Y if business objects are called during batch processing and on-line access to the system has been stopped by using the HLDDRIVER command.</p>	AN	1

Offset	Attribute	Description	Type	Length
61	Suppress-RCLRSC	<p>Suppress the execution of the RCLRSC (Reclaim Resources) command in the BOH.</p> <p>N – Execute the RCLRSC command.</p> <p>Y – Skip the RCLRSC command.</p> <p>If the value passed to the BOH is none of the above, this field will be defaulted to N.</p> <p>This parameter can be set to Y in cases where business objects are called repeatedly from within the same transaction. The BOH executes a RCLRSC command after each business object invocation, and one of the things this does it to close all Open Data Paths (ODPs) to database files. Unfortunately the RCLRSC command does not initialise the working-storage used in database IO modules, which can lead to MCH3402 exceptions as the IO module still believes the ODP is active on a subsequent call, whereas in reality it isn't. By setting this parameter to Y, the RCLSRC command will not be executed and therefore the IO modules should function as expected on subsequent calls.</p>	AN	1
62	Filler	Reserved for future use.	AN	39

2.3.2.Message Header

For each message segment within the whole transmission, at least one message header / message data pair follows. The Message Header is implemented through the MSGHDR copybook within SMART.

Offset	Attribute	Description	Type	Length
1	MsgID	Message ID - An identifier for the format of the data portion of the message.	AN	10
11	MsgLng	Message Length. This is the length of a single data portion of the message which follows this header. This excludes the MsgHdr.	N	5
16	MsgCnt	Message Count. This is the number of repeating groups which follow.	N	5
21	Filler	Reserved for future	AN	10

It is important to understand that the above message header copy member is used for ALL messages. Therefore, when coding, this copy member can appear more than once within the business object. To include it in a COBOL program, create a 01 level with any name and COPY this member immediately after it. Refer to any variables within the member with a qualified name.

2.3.3. Message Data

The data portion of the message is different for each message identifier. The Message ID describes the format of the message data. For example, for the method Client Create, the copybook could look like this:

```

01 CLTCRTI-REC.
  03 MESSAGE-HEADER.
    05 MSGID PIC X(010).
    05 MSGLNG PIC 9(005).
    05 MSGCNT PIC 9(005).
    05 FILLER PIC X(010).
  03 MESSAGE-DATA.
    05 BGEN-S2465.
      07 BGEN-S2465-ADDRTYPE PIC X(001).
      07 BGEN-S2465-BIRTHP PIC X(020).
      07 BGEN-S2465-CLNTNUM PIC X(008).
      07 BGEN-S2465-CLTADDR01 PIC X(030).
      07 BGEN-S2465-CLTADDR02 PIC X(030).
      07 BGEN-S2465-CLTADDR03 PIC X(030).
      07 BGEN-S2465-CLTADDR04 PIC X(030).
      07 BGEN-S2465-CLTADDR05 PIC X(030).
      07 BGEN-S2465-CLTDOBX PIC 9(008).
      07 BGEN-S2465-CLTDODX PIC 9(008).
      07 BGEN-S2465-CLTPCODE PIC X(010).
      07 BGEN-S2465-CLTPHONE01 PIC X(016).
      07 BGEN-S2465-CLTPHONE02 PIC X(016).
      07 BGEN-S2465-CLTSEX PIC X(001).
      07 BGEN-S2465-CTRYCODE PIC X(003).
      07 BGEN-S2465-DIRMAIL PIC X(001).
      07 BGEN-S2465-DOCNO PIC X(008).
      07 BGEN-S2465-ETHORIG PIC X(003).
      07 BGEN-S2465-GIVNAME PIC X(020).
      07 BGEN-S2465-LANGUAGE PIC X(001).
      07 BGEN-S2465-MAILING PIC X(001).
      07 BGEN-S2465-MARRYD PIC X(001).
      07 BGEN-S2465-MIDDL01 PIC X(020).
      07 BGEN-S2465-MIDDL02 PIC X(020).
      07 BGEN-S2465-NATLTY PIC X(003).
      07 BGEN-S2465-OCCPCODE PIC X(004).
      07 BGEN-S2465-RACRIND PIC X(001).
      07 BGEN-S2465-SALUTL PIC X(008).
      07 BGEN-S2465-SECURITYNO PIC X(015).
      07 BGEN-S2465-SERVBRH PIC X(002).
      07 BGEN-S2465-SOE PIC X(010).
      07 BGEN-S2465-SRDATE PIC 9(008).
      07 BGEN-S2465-STATCODE PIC X(002).
      07 BGEN-S2465-STATDSC PIC X(010).
      07 BGEN-S2465-SURNAME PIC X(030).
      07 BGEN-S2465-UKPENSIND PIC X(001).
      07 BGEN-S2465-VIP PIC X(001).

```

2.4. Business Object Code Structure

All the business objects developed have a standard structure - in much the same way as the SMART traditional on-line and batch programs have a standard structure.

An example and discussion of the relevant portions of a business object program can be found in this document in the section 3.2.5 Generated Business Object Example.

2.5. Error Handling

The architecture needs to be able to cater for a number of unexpected scenarios. These include data validation errors, communications faults, host application errors and so on. The mechanism for identifying the condition as well as the process for handling it forms part of this design and is described below.

2.5.1. Handling System Errors

A system error is an error that is encountered in the server application software. This may occur when the software encounters unexpected conditions where it may not be safe to proceed with processing. For example, a required table entry may not be found, or a claim may exist which is not linked to any particular policy.

These kinds of error are already monitored within POLISY and LiFE and those familiar with SMART-based applications will recognise calls to the SYSERR routine.

When such a system error is encountered the business object handler will be notified (by detecting a BOMB status in WSSP) and verb processing will stop. The action taken will be to:

- Rollback any updates to the database.
- Read back the ELOG record (which was written by SYSERR).
- Format and respond with the fatal error response SFERRREC using a status code of 'BOMB'.
- Set the ERRLVL indicator in the Leader Header to '2'.

The client application will be able to detect this condition and format a dialog box to advise the user of the situation.

2.5.2. Handling Communications Errors

Communications errors detected by the client need to be monitored for and handled by the client application software.

Communications errors detected by the server are handled differently and, depending on which connectivity option is chosen, errors can be reported differently. If such a condition occurs, an error message advising of whatever useful information is available will be logged onto the joblog for the server. Where possible, an ELOG record will be logged for statistical purposes. Action will be:

- Rollback any updates to the database.
- If possible, write an ELOG record.
- Format an error message and log it on the joblog.
- Format and return a fatal error response, consisting of the error message code 'BOMB'. The SFERRREC message structure is used for this purpose.
- Set the ERRLVL indicator in the Leader Header to '2'.

2.5.3.SFERRREC - Fatal Error

Should a fatal application error be detected, the business object will format an SFERRREC response message and send it back to the caller.

A single error may be sent to the caller. At the time of writing, the SFERRREC message has the following format:

Code in Business Object:

```
01 FATAL-ERROR.
   COPY SFERRREC.
```

Copybook layout:

```
03 MESSAGE-HEADER.
05 MSGID                PIC X(10).
05 MSGLNG               PIC 9(05).
05 MSGCNT               PIC 9(05).
05 FILLER               PIC X(10).

03 SFERR.
05 ERORSTATUZ          PIC X(04).
05 ERORDESC            PIC X(30).
05 BATCH               PIC X(05).
05 BRANCH              PIC X(02).
05 COMPANY             PIC X(01).
05 ERRLITS.
07 ERRLIT              PIC X(29) OCCURS 04.
05 FUNC                PIC X(05).
05 GENDATE             PIC X(10).
05 GENTIME             PIC X(08).
05 IOMOD               PIC X(10).
05 LASTPROG            PIC X(05).
05 MASTMENU            PIC X(05).
05 RFORMAT             PIC X(10).
05 SDATE               PIC X(10).
05 SECTIONNO           PIC X(04).
05 STIME               PIC X(08).
05 SUBMENU             PIC X(05).
05 SUBRNAME            PIC X(10).
05 SYSERRKEY           PIC X(100).
05 TERMINALID          PIC X(10).
05 TRANSCODE           PIC X(04).
05 USERID              PIC 9(06).
05 VN                  PIC X(01).
05 DATALOC             PIC X(01).
05 FILLER              PIC X(50).
```

The COBOL routine SFERR is responsible for formatting the SFERRREC message. To assist in writing the business object, a call to SFERR is automatically included from within the MAINX copybook.

2.5.4. Handling Editing Errors

Editing errors or validation errors are merely warnings that a program can detect during data validation. Such errors will not cause a program to abend. However, if the data in error was allowed to be saved to the database, it may cause other programs to abend at a later stage, hence such errors need to be corrected before proceeding.

These validations are currently monitored in the 2000 sections of applications developed using SMART. When an edit check fails, the program returns to the BOH a list of error codes against the field that has caused the error. The field in error, the error code and the associated text will be returned to the client in the trailer part of the response message. Actions will be:

- Rollback any updates to the database.
- Look up error code description and help information. The codes and description information will be sent back to the caller in an SVERR response message.

2.5.5. BOVERRREC - Validation Error

Should a validation error be detected, the business object will format a BOVERRREC response message and send it back to the Client.

Up to 25 validation errors may be sent to the Client. Any in excess of this are ignored. At the time of writing, the BOVERRREC message has the following format:

Code in Business Object:

```
01 VALIDATION-ERROR.
   COPY BOVERRREC.
```

Copybook layout:

```
03 MESSAGE-HEADER.
05 MSGID              PIC X(10).
05 MSGLNG             PIC 9(05).
05 MSGCNT             PIC 9(05).
05 FILLER             PIC X(10).
03 MESSAGE-DATA.
05 BOVERR-LANGUAGE    PIC X(01).
05 BOVERR-PROG        PIC X(10).
05 BOVERR-ENTID       PIC X(08).
05 BOVERR-EXTRA       PIC X(30).
05 BOVERR-ERRORS.
07 FILLER             OCCURS 25.
09 BOVERR-BOFOCCUR    PIC 9(05).
09 BOVERR-FIELD       PIC X(50).
09 BOVERR-EROR        PIC X(04).
09 BOVERR-DESC        PIC X(24).
```

The generated code within the business object is responsible for formatting the BOVERRREC message. To assist in writing the business object, a COBOL copy member BOVERRCPY should be used after each call to the 2000- section of a 'P' program. This will trap validation errors if requested. The ERRLVL indicator on the leader header is set to '1'.

2.5.6. Handling Escape Messages

An escape message is used by the operating system to indicate to a program that a serious problem has occurred. For example, if a program is not found or a file becomes damaged, an escape message is sent to the program attempting to use the object. These messages have a severity of 99 and are of the form MCH9999 or CPF9999.

Some escape messages such as COBOL errors may produce internal machine errors (e.g. MCH1202) and an answerable message will be sent to the program. Such messages offer the user the chance to continue processing with a default value.

The BOH establishes a COBOL error monitor program. This is called SMTERRHAND and is the same routine used to monitor errors from within the batch submission facility. It will receive control whenever an escape message is sent to the job and will return a copy of the message to the program invoking the BOH.

Action taken is:

- Rollback any updates to the database.
- Log a copy of the message to the joblog.

Note that by passing the Escape message to the program that invokes the BOH the responsibility of handling that Escape message is also transferred. This means the program invoking BOH will need to cater for such messages, e.g. in CL a MONMSG will be required to handle the Escape message and take appropriate action. Where CSC's Midrange Service Processor software (MSP) is used, no special action is required as the IBM Toolbox for Java APIs throw Java Exceptions which are caught by MSP and returned as errors in XML format to the invoking system. If MSP is not used, take care to handle Escape messages; If this is not done, jobs might end up with a MSGW status, meaning that no further processing will be done until the job has been ended and restarted.

2.5.7. Paging and Scrolling

Special consideration must be made when dealing with scrolling transactions. The Business Object must maintain a pointer to the current location between interactions with the client program. For example, the client program may ask for a page size of 30 records. The Business Object must always attempt to read ahead by 1 record to determine if there is more available. If so, the More Indicator is set to a 'Y' in the response leader header and the Message Count is set to 30.

If the client wants the next page (30) records, then the business object is called again. Note that the More Indicator must be set to 'Y', otherwise the Business Object will restart the selection from the beginning.

The Business Object then returns another page if possible and so the process continues until there are no more records available to return to the client program. In this case, the records found are returned in the last page, and the Message Count is set to however many were found. The More Indicator is set to 'N'.

2.6. SMART Business Objects

This section describes the SMART Business Objects that are available for use by developers.

2.6.1. Session Object: SESSION

2.6.1.1. Object Description

Summary

Module: SESSION
 Methods: START, RTRV, UPD, END, CHECK
 Linkage: LDRHDR, SESSIONI, SESSIONO
 Request MSGID: SESSIONI
 Response MSGID: SESSIONO

The SESSION object is SMART owned and comprises functions to commence a DRIVER session and pass and update WSSP parameters.

2.6.1.2. Method Descriptions

Session Start: SESSION.START

A VRBID (Method) of 'START' causes SESSION to execute code to initialise WSSP parameters (WSSPCOMN) and write them to the WSSP data-area for later access. It starts commitment control if requested, validates the company, branch and language, user-id and accounting period, and checks the user's authority for the company and branch.

Request Parameters for Message Format: SESSIONI			
Field Name	Description	Format	Size
Company	Company Number	Char	1
Branch	Branch Number	Char	2
Language	Sign-on Language	Char	1
Acctyr	Accounting Year	Num	4
Acctmn	Accounting Month	Num	2
Total Message Data Length: 10			

No response message is required if no error is encountered.

Retrieve Session WSSP: SESSION.RTRV

A VRBID of 'RTRV' causes SESSION to execute code to retrieve WSSP parameters (WSSPCOMN and WSSP user areas) from the WSSP data-area for use within the business objects.

There are no specific request parameters beyond the LDRHDR and VRBID.

Response Parameters for Message Format: SESSIONO			
Field Name	Description	Format	Size
WSSP	WSSP Common and User WSSP Data	Char	4066
Total Message Data Length: 4066			

Update Session WSSP: SESSION.UPD

A VRBID of 'UPD' causes SESSION to execute code to rewrite WSSP parameters (WSSPCOMN and WSSP user areas) to the WSSP data-area for subsequent use within the business objects. This would be used typically when WSSP values are updated dynamically by a submenu or subsequent program, e.g. for batch number allocation.

There are no specific request parameters beyond the LDRHDR and VRBID.

Response Parameters for Message Format: SESSIONO			
Field Name	Description	Format	Size
WSSP	WSSP Common and User WSSP Data	Char	4066
Total Message Data Length: 4066			

End Session: SESSION.END

A VRBID of 'END' causes SESSION to execute code to terminate the session.

There are no specific request or response parameters beyond the LDRHDR and VRBID.

Check Session: SESSION.CHECK

A VRBID of 'CHECK' causes SESSION to check the user-id's authority to company and branch and the WSSP values will be updated. This is an internal SMART function which will be used when the user-id on the LDRHDR changes value. This method should not be used by Client applications.

There are no specific request or response parameters beyond the LDRHDR and VRBID.

2.6.2. Fatal Error Object: SFERR

2.6.2.1. Object Description

Summary

Module: SFERR
 Methods:
 Linkage: LDRHDR, SFERRREC
 Request MSGID: SFERR
 Response MSGID: SFERR

The SFERR object is SMART owned and comprises functions to log and pass fatal errors from the application to the client system.

2.6.2.2. Method Descriptions

Fatal Error Processing: SFERR

There is no VRBID associated with this business object. It is called by all business objects when a fatal error has been encountered and logged via SYSERR. The error details are obtained from SYSERR and passed to the user, before performing a rollback.

Request Parameters for Message Format: SFERR			
Field Name	Description	Format	Size
Erorstatz	Error Status	Char	4
Erordesc	Error Description	Char	30
Batch	Financial Batch	Char	5
Branch	Branch code	Char	2
Company	Company Code	Char	1
Errlit	Error Literal (occurs 4 times)	Char	29
Func	Function	Char	5
Gendate	Generated Date	Char	10
Gentime	Generated Time	Char	8
Iomod	I/O Module	Char	10
Lastprog	Last Program Executed	Char	5
Mastmenu	Master Menu	Char	5
Rformat	Record Format	Char	10
Sdate	Start Date	Char	10
Sectionno	Section Number Where Error Occurred	Char	4
Stime	Start Time	Char	8
Submenu	Submenu	Char	5
Subrname	Subroutine Name	Char	10
Syserrkey	SYSERR key	Char	100
Terminalid	Terminal Identifier	Char	10
Transcode	Transaction Code	Char	4
Userid	User Identifier Number	Num	6
Vn	Version	Char	1
Dataloc	Data Location	Char	1
Total Message Data Length: 283			

2.6.3. Validation Error Object: SVERR

2.6.3.1. Object Description

Summary

Module: SVERR
 Methods:
 Linkage: LDRHDR, SVERRREC
 Request MSGID: SVERR
 Response MSGID: SVERR

The SVERR object is SMART owned and comprises functions to log and pass validation errors from the application to the client system.

Note:

This is an old-style validation error subroutine prior to SMART release 0003. From 0003 onwards validation errors are handled by each individual business objects through generated code that uses the BOVERRREC copybook. This subroutine should no longer be used.

2.6.3.2. Method Descriptions

Return Validation Errors: SVERR

There is no VRBID associated with this business object. It is called by business objects (created prior to SMART release March 2000) when a validation error has been encountered that is NON FATAL. Programs which use error codes as warnings in the code will cause this object to be invoked to pass the appropriate text message to the remote user. It reads the EROR file to obtain the appropriate text for the error code encountered and can pass up to 25 such codes.

Note: The Field "Field in error" will not be populated with a value by SVERR.

Request Parameters for Message Format: SVERR			
Field Name	Description	Format	Size
Errors	Occurs 25 times		
Field	Field in error	Char	25
Error	Error code for field	Char	4
Description	Error code description	Char	24
Language	Sign-on Language	Char	1
Prog	Program Number	Char	10
Total Message Data Length: 1336			

2.7. Notes on the Business Object Handler

2.7.1.Linkage

The linkage to the Business Object Handler (program BOH) consists of two fields, the Request Area and the Response Area. Both these fields are alphanumeric and are defined as PIC X(32000). The first 100 characters of both areas will always be the Leader Header, the rest can be used for the actual message Request and Response Data.

Note that the 32K data limit on linkage is sufficient for the majority of business objects. A special technique can be used to cater for the situation where more 32K of data needs to be passed to or received from a business object. Refer to section "4.12 – Processing Inbound Messages Over 32k Long" for more information.

2.7.2.Starting a Session

The Business Object Handler needs to emulate what the DRIVER command does for users in the traditional green screen environment. DRIVER checks authorities, company, branch, accounting period and sets up WSSP values. This processing will also need to take place for business objects, otherwise the Pnnnn programs will not be executed successfully.

To perform these actions, there is a special business object called SESSION: before executing a business object, the SESSION business object must be called with a method of START (and without errors). Calling this business object is exactly the same as a call to any other business object. The input to BOH is the Leader Header plus the information needed for the SESSION business object (for the layout, see the SESSIONI copybook).

The data held in the Request Area to BOH will look something like this:

1	2	3	4	5
1234567890123456789012345678901234567890				
12345678JOHNSMITH	QPADEV0001	SESSION	START	00
				1
6	7	8	9	0
1234567890123456789012345678901234567890				
1400YIRN0				
1	1	1	1	1
1	2	3	4	5
1234567890123456789012345678901234567890				
SESSIONI	0001000001		210E200003	

Positions 100 – 140 are the input for the Session business object. The first 30 characters are the Message Header, detailing the Message Id ('SESSIONI'), the length of the message data (excluding the length of the message header itself, so 00010) and the number of times this data is repeated (only once, so 00001).

The last 10 characters (positions 130 – 140) are defined by the SESSIONI copybook, i.e. company, branch, language and accounting period.

Positions 1 – 100 are the Leader Header, containing a message reference number ('12345678'), the user id ('JOHNSMITH'), a workstation id ('QPADEV0001'), the business object to be called ('SESSION') and the method ('START'), the total message length (00140), the operation mode('0'), the commitment control parameter ('Y'), the response mode ('I'), the message intent ('R'), the more indicator ('N') and the error level ('0').

How often the Session business object needs to be executed depends on the communication protocol that is being used. If the connection to the host system stays active once a business object has been executed, then it will be enough to call Session.Start once. During the day, all other business objects can then make use of the existing connection and the information that has been set up by Session.

If the communication works in a way that the connection is terminated each and every time a business object is executed and control is passed back to the front-end system, the Session business object will need to be executed each and every time. The consequence is that the session information will have to be passed along with the Request data. The specific BOHxxxxx program will then need to call BOH twice, first with Session.Start and then with the actual business object.

3. The Business Object Generator

To be able to produce business objects CSC has built the Business Object Generator. This generator allows developers to:

- Store definitions of business objects, methods and their interfaces
- Generate COBOL wrappers around transaction programs developed using SMART
- Store any manually added code to enable easy re-use and re-generation of business objects and methods.

There are three commands within the suite: DSNBOBJ, CRTBOBJ and TSTBOBJ.

DSNBOBJ provides the facility to define the business object details, its methods, its interface and the FSU, LiFE and/or POLISY transaction programs used to perform the business function. Further, it is possible to store and maintain manual code additions to enable easy regeneration of a business object.

CRTBOBJ allows the user to generate COBOL code for:

- The business object
- Copy members defining its interface
- Test rig software to allow the user to test the object before operating in a Client/Server mode.

TSTBOBJ provides the user with a generic test tool that can be used instead of, or in conjunction with, the generated test rigs.

The next sections will start with a description of the DSNBOBJ and CRTBOBJ commands. An example of a generated business object program is discussed, detailing the important sections. The TSTBOBJ command is described and the document concludes with the data model containing the files involved in this part of the SMART system.

3.1. DSNBOBJ - Design Business Object

3.1.1. Parameters

None.

3.1.2. S0200 – Work With Business Object Definitions

IGBFD - SYSTEM		Work With Business Object Definitions		S0200 02
Type Option. Press Enter.				
1-Create	2-Modify	3-Copy	4-Delete	
5-Enquire	6-Insert Src	7-CRTBOBJ	12-W/W Methds	
Position to:				
Sl	Business Object	Implmt. As	Type Description	Insert Source Enabled/Exists
	ACC	ACC	Account Maintenance	Y / Y
	AGT	AGT	Agent Maintenance	Y / Y
	ASG	ASG	Assignee Maintenance	Y / Y
	AUT	AUT	User authority for Fr...	N / N
	BEN	BEN	Beneficiaries	Y / Y
	BIL	BIL	Billing -Suppress+Cha...	Y / Y
	BIS	BIS	SUPER - Billing Busin...	N / N
	CBA	CBA	Client Bank Account	Y / Y
	CFI	CFI	Coverage Fund Interes...	Y / Y
	CFU	CFU	Coverage Fund Details	Y / Y
				More...
F1=Help F3=Exit F5=Refresh				

This screen displays basic business object details. From this screen, the user is able to:

- Create a new Business Object Definition, by entering a '1' into the Select field of line 1 and entering the name of a new object. Screen S0201 is displayed.
- Modify an existing Business Object Definition, by entering '2' against the business object. Screen S0201 is displayed.
- Copy an existing Business Object Definition, by entering '3' against the business object. Screen S0639 is displayed.
- Delete an existing Business Object Definition, by entering '4' against the business object. Screen S0205 is displayed.
- Display an existing Business Object Definition, by entering '5' against the business object. Screen S0201 is displayed.
- Insert Source code for an existing Business Object, by entering '6' against the business object. Screen S0206 is displayed.
- Run the CRTBOBJ command for an existing Business Object, by entering '7' against the business object.
- Work With the Methods of an existing Business Object, by entering '12' against the business object. Screen S0202 is displayed.

The screen offers a 'position to' facility and the page up/down keys can be used. The Refresh function key can be used to quickly position to the top of the screen.

The 'Insert Source Enabled' and 'Exists' show which Business Objects can use Insert Source processing and which already have source code inserted. The 'Enabled' field is stored on the Business Object Details file (PGBOPF) and can be changed manually by entering 2 against the Business Object.

The Type column indicates whether it is a normal business object or a Super business object.

Options 1, 2, 3, 4 & 7 can only be used when in a Development Level. Options 5, 6 and 12 can be used in any Level.

3.1.3.S0201 – B/O Definition (Maintenance)

```

IGBFD - SYSTEM      B/O Definition Maintenance  S0201 02

Object. . . : CLIENT

Description :

Member name :          Subsystem :

Alias . . . :

Type. . . . : S      Insert Source Allowed : Y

F1=Help  F3=Exit  F12=Cancel

```

This screen is used to capture information relating to the business object. This data will be stored in the business object repository for later use by the business object generator.

The field labelled 'Description' is text only and entry is mandatory.

The field labelled 'Member Name' indicates the name of the physical source file member that will later be generated into the QLBSRC file in the development library. It may be useful to generate a temporary source member that can later be manually merged into an existing business object. This name may be up to 10 characters long and must begin within an alphanumeric character and must not contain embedded blanks. The entry is mandatory.

The value entered here will be checked against the Software Dictionary. When in Create mode, if the "Member Name" is found on the Dictionary, the entry will not be allowed. When in Modify mode and the "Member Name" is changed, the new value will be validated as described for Create Mode.

The subsystem name must exist on Table T1685 and a window facility is provided. This entry is mandatory.

The field labelled 'Alias' is not currently used but is intended for future use by workstation software where method names may be longer than, or different to, those defined within the business object repository.

The field labelled 'Type' indicates the type of business object. Valid values are space and S – a space indicates a normal business object and an S indicates a Super business object.

The field labelled 'Insert Source Allowed' governs the manual addition of code to the business object definition. If set to Yes, the only way to add code to a business object is via DSNBOBJ and using option 6 (Insert Src) against the business object. It will not be possible to edit the business object using the ED command.

If the field is set to No, then the option 6 will not be allowed and the business object must be manually edited using the ED command.

When option 5 (Enquire) is used on screen S0200, screen S0201 is displayed with all fields protected.

3.1.4. S0205 – Delete B/O Definition

```

IGBFD - SYSTEM   Delete B/O (Method) Definition S0205 02

Object. . . :   AGT

Description :   Agent Maintenance

Alias . . . :

Delete Level:   IGBFD   Delete Generated Elements:  Y

Override Password :
                  Confirm . . : N

F1=Help  F3=Exit  F12=Cancel

```

This screen is presented when the user requests the deletion of a business object definition.

The user must specify the Level from which to delete the Business Object Definition in the “Delete Level” field. This will be up to and including the specified Level. If it exists, the Business Object definition will then be copied down from the Level above the specified Level to the current Development Level and to all other Levels below the specified Level. Should the Business Object exist in one of these lower Levels (Dictionary is checked for “Member Name”) the definition records will not be copied to this Level or any Levels below. If the Release Level is specified, then the definition records will be deleted from all Levels in the Virtual Machine except those where the Software Dictionary indicates the Business Object also exists.

The user can request that the Elements generated for the Business Object and associated Methods are also deleted from the specified Level by entering ‘Y’ in the “Delete Generated Elements” field. This submits the DLTOBJLVL and DLTOBJREL commands for all the possible generated Elements.

The user can enter the “Override Password” to ensure that all Elements are successfully deleted. If the specified Delete Level is not the current Level, this field is mandatory.

The user must confirm the deletion by entering a ‘Y’ in the confirmation field.

All the methods, program lists, field lists and inserted source related to this business object will be removed.

```

IGBFD - SYSTEM   Delete B/O (Method) Definition S0205 02

Object. . . :   ACC
Method. . . :   BALLST
Description :   Account Balance List

Alias . . . :

Delete Level:   IGBFD   Delete Generated Elements:   Y

Override Password :           Copy down Method :   Y
                        Confirm . . : N

F1=Help  F3=Exit  F12=Cancel

```

This screen is presented when the user requests the deletion of a Business Object Method definition. The fields are the same as those presented when deleting a Business Object, with the addition of the “Copy down Method” field.

By entering ‘Y’ in this field the user can request that the definition records for the selected Method are copied down from the Level above the specified Level (if they exist) and this includes Inserted Source records. If the user enters ‘N’ in the field, the definition records will be deleted but not copied from the above Level.

When the “Delete Generated Elements” field is set to ‘Y’ for a Method, only the generated Elements for the selected Method will be deleted.

3.1.5.S0202 – Work With B/O Method Definitions

IGBFD - SYSTEM		Work with B/O Method Definitions		S0202 02
Business Object: ACC				
Account Maintenance				
Type Option. Press Enter.				
1-Create	2-Modify	3-Copy	4-Delete	
5-Enquire	7-GENXSL			
Sel	B/O Method	Implmt. As	Description	
	BALENQ	ACCBALLENQ	Account Balance - Generic	
	BALLST	ACCBALLST	Account Balance List	
	MOVLST	ACCMOVLST	Account Movement List	
	TRNLST	ACCTRNLST	Transaction Listing	
Bottom				
F1=Help F3=Exit F5=Refresh F12=Cancel				

This screen displays basic business object method details. From this screen, the user is able to:

- Create a new Method Definition for the business object, by entering a '1' into the Select field of line 1 and entering the name of a new method. Screen S0203 is displayed.
- Modify an existing Method Definition for the business object, by entering '2' against the method. Screen S0203 is displayed.
- Copy an existing Method Definition for the business object, by entering '3' against the method. Screen S0640 is displayed.
- Delete an existing Method Definition for the business object, by entering '4' against the method. Screen S0205 is displayed.
- Display an existing Method Definition for the business object, by entering '5' against the method. Screen S0203 is displayed.
- Run the GENXSL command for an existing Business Object method , by entering '7' against the method.

Options 1, 2, 3, 4 and 7 can only be used when in a Development Level. Option 5 can be used in any Level.

3.1.6.S0203 – B/O Method Definitions Program List

IGBFD - SYSTEM		B/O Method Definition - Program List		S0203 02	
Business Object:		ACC	(Implemented as : ACC)
Method . . . :		BALLST			
Description . :		Account Balance List			
Implemented as :		ACCBALLST	Alias:		
Suppress Screen Group Tag in XSL . . :		N	Section		
Seq no.	Program	Type	1	P	2 3 4
10	P6235	P	X		Contract Enquiry - Sub-Account

Bottom

F1=Help F3=Exit F5=Refresh F10=Resequenece F12=Cancel

This screen displays the program list for a particular business object method. The user is prompted to enter the names of the transaction programs required for the method. For example, for an Agent Create method, the programs P5043 and P5035 need to be executed. The designer will enter these two program names in the list and nominate the sections of these programs to be executed.

A good knowledge of the function contained within each section of the transaction programs is a pre-requisite to effective design. In most cases it is safest to select ALL sections. However, this has an adverse effect on response time.

For example, it may not always be necessary to execute the 4000 section of a program since the Business Object controls program sequencing. Unfortunately, there are some programs that perform updates to WSSP fields or perform KEEPS functions within the 4000 section and these may be required in a subsequent program.

The field labelled 'Description' is text only and is mandatory.

The field labelled 'Implemented as' indicates the name of the physical source file member that will later be generated into the QLBSRC file in the development library. It may be useful to generate a temporary source member that can later be manually merged into an existing business object. This name may be up to 9 characters long and must begin with an alphanumeric character and must not contain embedded blanks. The entry is mandatory. The reason only 9 characters are allowed is that this name will be suffixed with an 'I' and an 'O' for the input and output copybooks for this method.

The value entered here will be checked against the Software Dictionary. If in Create mode and any Element that could possibly be generated from this Method (Command, Program, Copybook, XSL) is found on the Dictionary, the entry will not be allowed. If in Modify mode and the "Member Name" is changed, the new value will be validated as described for Create Mode.

The field labelled 'Alias' is not currently used but is intended for future use by workstation software where method names may be longer than or different to those defined within the business object repository.

The field labelled "Suppress Screen Group Tag in XSL" is used to indicate whether or not the Screen Group Tag <Snnnn> needs to be suppressed when the XSL stylesheet is generated for the method.

The field labelled 'Seq no.' is used to manipulate the order of the programs. In modify mode, the numbers can be overwritten. Upon pressing the <ENTER> key or after using the Resequencing Function key, the list will be redisplayed using the new order. Note that the numbers are automatically incremented by 10. The first line can be used to add a new program to the list. As the sequence number is defaulted to 999, this new program will be added to the end of the list. If the new program needs to be inserted into the list at a specific position, it is simply a matter of changing the sequence number from 999 into the required number.

The field labelled 'Program' indicates the name of the program that will be called from this method.

The field labelled 'Type' indicates what type of function is to be run. A 'P' or 'N' means that a Pnnnn program is required. A value of 'O' indicates that another business object will be called from within this method. A value of 'M' indicates that another business object *method* will be called from within this method.

Note: When using a type of 'P', it is a requirement that the Pnnnn program is converted to be Client-Server enabled, post SMART 9509. For Pnnnn programs without a screen a type of 'N' should be used.

A type of 'M' is only allowed for Super business objects and conversely you can only define type 'M' for Super business objects.

The fields labelled 'Section 1 P 2 3 4' can be used to indicate which of the 5 sections of this on-line program will be executed by the method. Note that for a program of type 'P', at least one section must be ticked. For type 'O', none of the section fields must be selected.

To delete a program from the list blank out the sequence number. After pressing the <ENTER> key or the Refresh or Resequencing function keys the list will be redisplayed without the deleted program.

Validation is present to prevent the developer from blanking out a section for which there is an Insertion Point with inserted source by displaying an error message. Similarly, if the developer has cleared or replaced a program for which there is an Insertion Point with inserted source, the cleared/replaced program will be restored to the screen and an error message displayed.

If the program detects no more changes to the list, the <ENTER> key will result in the display of the next screen S0204 – the field list.

3.1.7.S0204 – B/O Method Definitions Field List

IGBFD - SYSTEM B/O Method Definition - Field Mapping							S0204 01
Business Object: ACC							
Method : BALLST							
					Add		
ScreenPart	Occur	Field	Type	Size	Use	Inf	Description
S6235		CHDRNUM	A	8			CONTRACT NUMBER
		CHDRSTATUS	A	10			CONTRACT RISK STATUS
		CNTCURR	A	3			CURRENCY
		CNTTYPE	A	3			CONTRACT TYPE
		CTYPEDES	A	30			CONTRACT TYPE DESCRIPTION
		JLIFENAME	A	47			JOINT LIFE NAME
		JLIFENUM	A	8			JOINT LIFE NUMBER
		LIFENAME	A	47			LIFE CLIENT NAME
		LIFENUM	A	8			LIFE INSURED NUMBER
		PREMSTATUS	A	10			PREMIUM STATUS
		REGISTER	A	3			REGISTER
		COMPONENT	A	8	O		COMPONENT
		CURR	A	3	O		CURRENCY CODE
		RECTYPE	A	1	O		
S6235SFL	100						
							More...
Add fld I . . . + Add fld O . . .							
F1=Help F3=Exit F5=Refresh F12=Cancel							

For each program chosen in the Program List screen, the user must select the fields that are of importance and how they are to be used within the method.

The screen fields are retrieved from the display file definition and each field is displayed with the type, size and number of decimal places.

The developer can use any of the fields as I - Input, O - Output, B - Both Input and Output or D - Default in the interface to the method. Fields that are 'output' only on the display file cannot be selected as Input or Both. Due to a restriction in the IBM i DSPFFD command, hidden fields are displayed in this list and can be selected for both input and output. These fields are not input/output capable on the screen, therefore they should never be selected as either input or output to/from a method.

Fields with a type of C, M, V or Z cannot be selected. Fields with a type of D or T will be converted to a type of N in the input/output copybooks with the appropriate length.

Fields with a type of E will be converted to a type of S in the input/output copybooks with the appropriate length.

The Occurs field is used for subfile screens. For these screens it is possible to enter a value that will indicate how many subfile lines are used in the business object. The generated input/output copybooks will include an OCCURS clause for the subfile fields that will correspond to the value entered.

For fields defined with usage I or B, the "Additional Info" checkbox can be used to define the "Retain Screen Value" for a field. Selecting this checkbox and pressing <Enter> will display screen S0213.

```

IGBFD - SYSTEM      Additional Field Info  S0213 01

Field : CNTCURR

Retain Screen Value : N

Fl=Help  F3=Exit  F12=Cancel

```

The Retain Screen Value parameter can be used to indicate that when the Snnnn-field has a value other than spaces/zeroes, it will NOT be overwritten with the BGEN-field from the business object input copybook.

A value of 'Y' in the RSV field indicates that when the Snnnn-field has a value other than spaces/zeroes, it will NOT be overwritten with the BGEN-field from the business object input copybook. Instead, the value as set up in the 1000- section of the Pnnnn program will be retained. Note this screen is not selectable for subfile fields. The reason for this is that for subfile fields the generated business object code will clear the subfile (SCLR) and then add the required number of subfile lines (SADD). In this situation there are no screen field values to retain.

For Fields defined as D, the "Additional Info" will be selected automatically if no additional information exists already. Selecting this checkbox for Fields with Usage D will also display screen S0213, but with a slight different layout.

```

IGBFD - SYSTEM      Additional Field Info  S0213 01

Field : CNTCURR

Default Value . . :
Use Blank Default : N

Fl=Help  F3=Exit  F12=Cancel

```

For Fields defined with usage D this screen can be used to define the Default Value that the BO generator should use to initialise the field. If the field is defined as Numeric, the value entered should be a valid Numeric value. If the field is defined as a Date field, the value entered should be a valid date. The Default value cannot be blank, unless the Blank Default Value parameter is set to Yes.

Additional Input and Output fields can be added by using the selection boxes at the bottom of the screen. Selecting either field will display screen S0638 where additional field details can be created/modified.

3.1.8.S0638 – B/O Method Definitions Additional Copybook Fields

IGBFD - SYSTEM B/O Method Definition Additional Copybook Fields S0638 01						
Business Object: ACC				Additional Fields: INPUT		
Method : BALLST						
Seq no.	Level	Field	Group	Type	Size	Occurs
9999						
10	07	FIELDA		A	3	
20	07	FIELDDB		A	4	
30	07	FIELDDC		N	2	
						Bottom
F1=Help F3=Exit F5=Refresh F10=Resequence F12=Cancel						

This screen captures details of additional fields that are to be added to the generated Input or Output copybook. The 'Additional Fields:' title at the top of the screen will indicate if the fields will be added to the input or the output copybook.

To add a new field, enter the details on the 1st line of the subfile. To delete a field blank out the sequence number. The additional fields will be added to the input/output copybook as follows:

```

03 MESSAGE-HEADER.
...
03 MESSAGE-DATA.
05 BGEN-S1234.
....
===== 05 ADDITIONAL-FIELDS. <=====
07 BGEN-FIELDA PIC X(003).
07 BGEN-FIELDB PIC X(004).
07 BGEN-FIELDDC PIC 9(002).
...

```

The fields in the copybook will be sequenced by the number specified in the 'Seq no.' field. Changing this sequence number will re-sequence the fields.

Valid COBOL levels for the fields in the copybook are 07 and 09. The first field in the list must start with level 07.

The 'Field' field contains the name of the field to be added to the copybook. If the field is a group heading, the name is free format otherwise the field must exist on the field dictionary. Standard Window facilities exist for this field.

Enter an X in the 'Group' field to indicate that the field is a group-heading field. If the field is a group field it cannot have a picture clause. It may have an OCCURS clause.

This method for adding fields should be used in preference to the use of Inserted Source (see later) for copybooks, as the business object generator will take additional field details into account when generating the test rigs. The use of Inserted Source for copybooks should be limited to complex copybook structures only.

3.1.9.S0634 – B/O Method Definition - Field Mapping

IGBFD - SYSTEM B/O Method Definition - Field Mapping			S0634 01
Business Object: ACC		Method : BALLST	
1-Change			
I/O	Scrnpart	Field	XML Tag
O	S6235SFL	COMPONENT	ContractComponent
O		CURR	Currency
O		RECTYPE	
O		SACSCODE	
O		SACSCODED	
O		SACSCURBAL	
O		SACSTYP	
O		SACSTYPD	
O	Add. Fld	ACCTCURR	+ AccountingCurrency
O		ACCTYEAR	AccountingYear
O		ACCTMONTH	
			Bottom
F1=Help F3=Exit F12=Cancel			

If there are any XML Tags defined for any of the fields selected for input or output, then screen S0634 is displayed.

This screen is used to view/maintain the mapping of XML Tags against the fields selected as Input and/or Output for a Business Object, plus any additional fields selected for Input and/or Output.

Firstly all Input fields are displayed, followed by all Output fields. Fields that have been selected for both Input and Output will therefore be displayed twice in the list.

Each field will also show previously a previously selected XML Tags. If no tag has been selected, but tags exists on the FLDAPF file, then a "+" sign in front of the XML Tag field will indicate that one or more tags are available for selection. If a tag has already been selected and there is a "+" sign, then this means that one or more alternative tags to the one selected exists on the FLDAPF file.

To add, change or remove a tag, simply press F4 on the tag and (de-)select the relevant tag on the popup window screen S0637.

IGBFD - SYSTEM Field / XML Tag Selection		S0637 01
Field: ACCTCURR Input/Output: O		
Select an XML tag, or de-select all to remove current tag		
X	AccountingCurrency	
	Currency	
		Bottom
F1=Help F3=Exit F12=Cancel		

The available XML Tag values as stored on the FLDAPF file are maintained in DRIVER. Please refer to section 3.2.6 Development Utilities – Field / XML Tag Mapping for further information.

3.1.10. S0206 – B/O Definition – Insert Source

The Business Object Definition Insert Source screen is accessed by entering a 6 against a Business Object on the Work With Business Definitions screen. This screen displays a list of the available Insertion Points for the selected Business Object. Some of these insertion points are standard for all Business Objects, others are based upon the Methods defined for the Business Object and the programs, sections and fields selected for those methods. Sections are included in the list of Insertion Points for ease of navigation only and cannot be selected for source insertion.

IGBFD - SYSTEM		Business Object Definition - Insert Source		S0206 01
Business Object:		ACC	(Implemented as : ACC)	
Enter A to Insert After, B to Insert Before, D to Delete and E to Enquire				
Sel.	Type	Insertion Point		
		AMENDMENT HISTORY		
	A	WORKING-STORAGE SECTION		
	A	01 FORMATS		
	A	FILE SCHEMA COPYBOOKS		
		1000-INITIALISE SECTION		
		1010-START		
		1090-EXIT		
		2000-CALL-PROGRAMS SECTION		
		2010-CALL		
		2090-EXIT		
		MTH-ACCBALLST SECTION		
		MTH-ACCBALLST-BEGIN		
	B	MTH-ACCBALLST-P6235-1000		
			More...	
Insert Type: B=Before, A=After, X=Both				
F1=Help F3=Exit				

The Type field will be filled for any Insertion Points that already have inserted source based on where the source is inserted. Where inserted source exists before the Insertion Point, a 'B' is displayed in the Type field. Similarly, where inserted source exists after the Insertion Point, an 'A' is displayed in the Type field. Where inserted source exists both before and after the Insertion Point, an 'X' is displayed in the Type field.

To insert source before an Insertion Point, enter a 'B' against the Insertion Point. Similarly, to insert source after an Insertion Point, enter an 'A' against the Insertion Point. Validation is included to prevent insertion of source after an 'Exit' Insertion Point and to prevent insertion of source before one of the following standard insertion points:

- AMENDMENT HISTORY
- WORKING-STORAGE SECTION
- 01 FORMATS
- FILE SCHEMA COPYBOOKS

To enquire upon inserted source, enter an 'E' against the Insertion Point.

When creating or amending source at an insertion point using option 'A' or 'B', the Auto Tag Inserted Source screen S0211 is displayed, providing the Auto Tagging or Work Unit software

default is switched on. If both software defaults are switched off the screen is not displayed.

To delete all inserted source for an Insertion Point, enter a 'D' against the Insertion Point. This will display a confirmation screen requesting that confirmation of the delete.

If inserted source exists both before and after an insertion point, option 'D' will remove both the inserted source before and after the insertion point. If it is required to remove either the inserted source before or after the insertion point, use either option 'B' or 'A' and delete the source lines in the SEU screen.

Please note that the same program can be selected more than once for inclusion in a business object method. In the situation where the order of programs is P1111, P2222 followed by a second call to P1111, the insertion points generated will be:

MTH-ADWCRC-P1111-**001**-1000

...

MTH-ADWCRC-P1111-**001**-4000

MTH-ADWCRC-P2222-1000

...

MTH-ADWCRC-P2222-4000

MTH-ADWCRC-P1111-**002**-1000

...

MTH-ADWCRC-P1111-**002**-4000

Super Business Objects will only have 1 insertion point: IDENTIFICATION DIVISION. For this insertion point only after inserted source can be entered.

3.1.11. S0207 – B/O Definition – Insert Copybook Source

The Business Object Definition Insert Copybook Source screen is displayed by entering a 6 against a Business Object from within the Work With Business Definitions screen, and then pressing <enter>. The screen (S0207) Business Object Definition – Insert Copybook Source, is then displayed. A list of the Input and Output copybooks that will be generated for the selected Business Object, based upon the Methods defined for the Business Object is shown.

```

IGBFD - SYSTEM   Business Object Definition - Insert Copybook Source S0207 01

Business Object:  ACC                      ( Implemented as : ACC          )

Enter I to Insert, D to Delete and E to Enquire

      Insert Source
Sel.    Exists    Insertion Point (Copybook)
      Y          ACCBALLSTI
      Y          ACCBALLSTO
      Y          ACCBONENQI
      Y          ACCBONENQO
      Y          ACCMOVLSTI
      Y          ACCMOVLSTO
      Y          ACCTRNLSTI
      Y          ACCTRNLSTO

F1=Help  F3=Exit

Bottom

```

The Insert Source Exists field will be set to 'Y' for any copybook that already has inserted source.

To insert source to the end of a copybook, enter an 'I' against the copybook.

To enquire upon inserted source, enter an 'E' against the Insertion Point.

Processing will then take the developer to the Auto Tag Inserted Source screen, S0211, which may or may not be displayed depending on the values of the Auto Tagging and Work Unit software defaults, and whether the developer is creating or modifying Inserted source or enquiring on existing Inserted source.

To delete all inserted source for a copybook, enter a 'D' against the Insertion Point. This will display a confirmation screen requesting that the developer confirms the delete request.

To partially delete inserted source at an Insertion Point, enter an 'I' against the relevant copybook to modify the existing Inserted source. In the SEU screen delete the appropriate source lines. Deleting all of the source lines can be used instead of the delete processing to delete all inserted source before or after an Insertion Point.

As indicated above, the recommendation is to use the 'Additional Fields' feature instead of Inserted Source for copybooks. Only if complex structures are required, e.g. an OCCURS clause within an OCCURS clause, should the Inserted Source be used.

3.1.12. S0211 – Auto Tag Inserted Source

This window is displayed when selecting to insert source against an insertion point from S0206 (or S0207 in the case of a copybook).

The window will only be displayed if a work unit or tag needs to be captured. This is dependent on the values of the Work Unit and Auto Tagging utility defaults. If both are disabled there is no need to capture a work unit or tag and the window will not be displayed. Similarly, if the developer is enquiring on Inserted Source the window is not displayed. If either is enabled the window will be displayed as shown below.

IGBFD - SYSTEM	Auto Tag Inserted Source	S0211 01
Business Object : ACC		
Insertion Point : ACCBALLSTO		
Before or After : C		
Work Unit or Tag: GBF007		
F1=Help F3=Exit F12=Cancel		

If Work Unit processing is enabled the value entered in the Work Unit or Tag field must be a valid work unit to which the developer is authorised to edit under. Otherwise there is no validation on the value entered except that it must be entered.

Processing will then take the developer to a Source Edit Utility screen allowing them to modify any existing inserted source or create new inserted source.

The very first line will be a comment line, in blue, indicating the insertion point. If no inserted code exists a second blue comment line will be displayed. This line is there to help the developer to format the code.

The blue coloured lines will not be stored as inserted code, however if the first character of such a line is overwritten with a space character, the line becomes active and WILL be stored. This can be useful when for example wanting to add a record format. Overwriting the first character will activate the line and the developer then only needs to input the correct file name.

Columns . . . :	1 71	Edit	QTEMP/QLBLSRC
SEU==>			AFTER
FMT C** 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7		
	***** Beginning of data *****		
0001.00	*1010-START		
0002.00	*A---B+++++		
	***** End of data *****		
F3=Exit F4=Prompt F5=Refresh F9=Retrieve F10=Cursor F11=Toggle			
F16=Repeat find F17=Repeat change F24=More keys			

If source already exists for the selected Insertion Point and Type, the screen will look similar to.

```

Columns . . . :   1  71          Edit          QTEMP/QLBLSRC
SEU==>          AFTER
FMT C* .....*. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
***** Beginning of data *****
0001.00      *ADDITIONAL SECTIONS
0002.00      *A---B+++++
0003.00      /
0004.00      S5039-SREAD-SUBFILE SECTION.
0005.00      *****
0006.00      S5039-START.
0007.00
0008.00      MOVE O-K          TO SCRN-STATUZ.
0009.00      MOVE SREAD      TO SCRN-FUNCTION.
0010.00
0011.00      CALL 'S5039IO'      USING SCRN-SCREEN-PARAMS
0012.00                                S5039-DATA-AREA
0013.00                                S5039-SUBFILE-AREA.
0014.00
0015.00      IF SCRN-STATUZ      NOT = O-K
0016.00      AND SCRN-STATUZ      NOT = MRNF

F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Cursor   F11=Toggle
F16=Repeat find   F17=Repeat change   F24=More keys

```

On exit, if Auto Tagging is switched on, each new or changed line of source will be tagged with the tag or work unit entered. When tagging a line that has already been tagged and is being commented out, the tag is inserted inside the existing tag. If, for any reason, a tag cannot be inserted where it should be, the screen is redisplayed with a message to indicate this and the user is forced to re-edit the inserted source. Each line that caused the tagging error has exclamation marks displayed in the first six positions. No tagging is done where Work Units are enabled and the work unit specified is a retrofit work unit.

If Work Units are enabled, the program or copybook for which insert source has been added is registered against the Work Unit previously specified.

When an enquiry is requested the developer will be taken directly to the SEU screen, allowing them to browse all of the inserted source for the Business Object and Insertion Point. 'Before' Inserted Source will be displayed first if the Business Object Insertion Point has inserted source both before and after the Insertion Point.

In order for a developer to distinguish between manually inserted source and inserted source generated by the CRTBOBJ command, all generated inserted source will be flagged with a letter 'G' in the first position of the source line. Note that a subsequent manual amendment of such an inserted source line will lead to automatic removal of this character, so there is no need to remove this by hand.

3.1.13. S0212 – Delete Inserted Source

To delete inserted source, enter a 'D' against an Insertion Point from the Business Object Definition Insert Source screen. The screen displays the business object and insertion point for which the inserted source is to be deleted. The developer must enter a 'Y' to confirm that they wish to proceed with the deletion. Leaving the field as N or pressing an exit key (including F12) will result in the inserted source not being deleted.

```
IGBFD - SYSTEM      Delete Inserted Source      S0212 01

Business Object : AGT
Insertion Point : FILE SCHEMA COPYBOOKS

Are you sure you want to delete the inserted source?

                N (Enter Y or N)

F1=Help  F3=Exit  F12=Cancel
```

3.2. CRTBOBJ - Create Business Object

3.2.1. Parameters

"Implemented As" member name - IMPLMTAS

The implemented as name of the business object. This must be a valid name previously created using DSNBOBJ. Either the implemented as name or the full business object name must be entered.

Business Object - BOBJ

The full business object name. This must be a valid business object previously created using DSNBOBJ.

Replace source - REPLACE

This is an option to copy any generated source from temporary files in QTEMP back to the development source library. If set to *NO, this will not replace any existing code within the development environment.

Developers may find it useful to generate code in QTEMP and manually merge the generated code into existing source members.

This parameter defaults to *YES.

Compile generated objects - COMPILE

This is an option to compile all the generated objects. If set to *NO, the generated objects will not be automatically compiled.

Developers may find it useful not to compile the generated objects as manual changes need to be made first.

This parameter defaults to *YES.

Compile BO with *SRCDBG - SRCDBG

This is an option to compile the generated business object with option *SRCDBG.

This parameter defaults to *NO.

Work Unit - WORKU

If work unit processing is enabled, then a valid work unit must be entered. This will default to the user's default work unit. All code generated and transferred to development libraries will be registered in this work unit and assigned to the current user.

Generate business object - PGM

An option to generate and compile the Business Object.

Defaults to *YES.

Generate message copy members - MSG

An option to generate the Business Method interface copy members.

Defaults to *YES.

Generate test harness - TST

An option to generate and compile the Business Method test rig source members.

Defaults to *YES.

Generate XSL IO stylesheets - XSL

An option to generate the Business Method XSL style sheets.

Defaults to the value of the CRTBOBJXSL Software Default (which is either *Yes or *NO). If this Software Default is not present, the default will be *NO.

Job Queue - JOBQ

The name of the job queue to which the processing will be submitted. This allows the user to generate the source in the interactive job by using the *NONE option and having the compilations performed in batch.

The default is BACKGROUND. Other options are QBATCH, *NONE or NIGHT.

3.2.2. Source Members Generated

3.2.2.1. Business Object

A source member is created with a name of the 'Member Name' as defined in the Business Object in PGBOPF. If this member already exists when transferring from QTEMP to the development source library, then a copy of the old member is made to QLBSRC file in the development GEN library. This library may be called PDEVGEN for example.

The CRTBOBJ command will search for inserted source code before and/or after each Insertion Point and include all inserted source code for the Insertion Point in the correct place within the generated Business Object Cobol program.

If Insert Source is not switched on, all source code generated to move subfile fields between the Business Object copybook and the screen (and vice-versa) is written to the generated Business Object Cobol program. However, if Insert Source is switched on, provided there are no Insert Source records for the Subfile Input (or Output) insertion point, the generated source code is written as Insert Source records and inserted into the generated Business Object Cobol program.

3.2.2.2. Message copy members

Two source members are created with a name of 'Member Name' for each Business Method. This name is concatenated with an 'I' or an 'O'. An 'I' suffix indicates the request message layout. An 'O' suffix indicates a response message layout.

Fields defined with a usage type of 'I' or 'B' will be included in the request message member.

Fields defined with a usage type of 'O' or 'B' will be included in the response message member.

For example, if the Client business object has three methods with member names of CLTCRT, CLTDLT and CLTRTV, then six copy members named CLTCRTI, CLTCRTO, CLTDLTI, CLTDLTO, CLTRTVI and CLTRTVO are generated.

If this member already exists when transferring from QTEMP to the development source library, then a copy of the old member is made to QLBSRC file in the development GEN library. This library may be called PDEVGEN for example.

The generator will check for inserted source code for each generated copybook and add this to the end of the copybook.

3.2.2.3. Test Harness

To assist developers, the generator produces a COBOL program and a matching Command object. This command will prompt for input parameters to the business object and will call the business object via the Business Object Handler.

It will also print to a spool file a copy of the request message before the call and the response messages after the call.

The names of these source members are the same as the Member Name for each method. In the above example, three command names CLTCRT, CLTCHG and CLTDLT would be generated. Also, three COBOL Command Processing Programs with the same names (suffixed by an X) would be created.

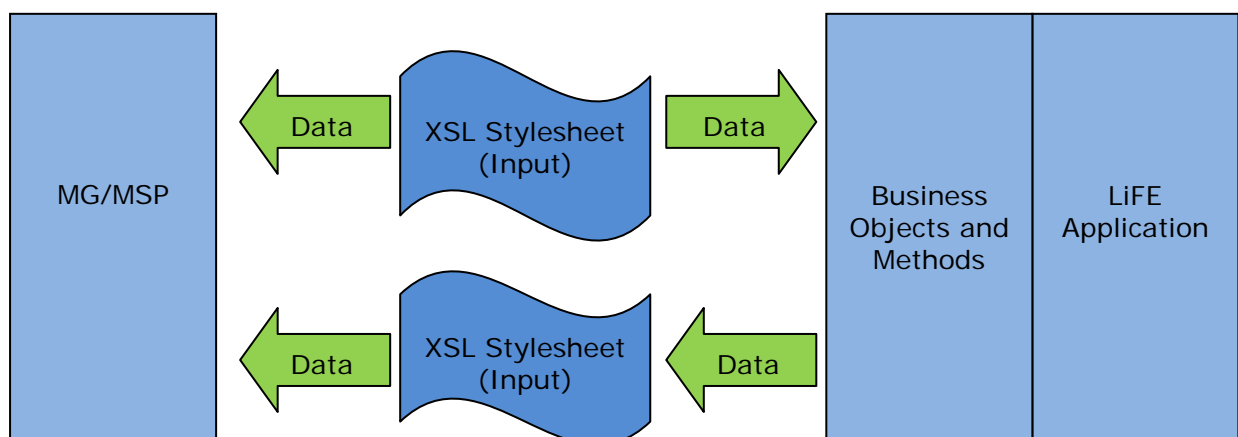
If the member already exists when transferring from QTEMP to the development source library, then a copy of the old member is made to QLBSRC or QCMDSRC file in the development GEN library. So for example in level PDEV, this library will be called PDEVGEN.

If the QCMDSRC file does not exist in the development GEN library, it is automatically created.

Note: Although the test harness functionality is a viable method of testing, it is being superseded with the online 'Business Object Test Tool (BOTT)' as it allows for much larger and more complex business objects. See section [Business Object Test Tool](#) below

3.2.2.4. XSL IO stylesheets

Back-end XSL stylesheets are required for Graphical Business Function developments which use the Enterprise Enabler (EE) and Midrange Gateway (MG) infrastructure. These XSL stylesheets enable the passing of data from a server-based application using MG/EE to and from a LiFE application (the host). They contain the data descriptions that MG/EE needs in order for the data to be successfully input and/or output. For each field required in the front-end application (such as GBF), the equivalent field will exist on the host application (LiFE) and therefore must exist in the back-end XSL stylesheet for the successful transformation of data. Essentially, the back-end XSL stylesheets are a bridge for data to be passed between the MG/EE infrastructure and LiFE.



The XSL Generator automatically generates these back-end XSL stylesheets, thereby removing the need for a developer to manually create them. These stylesheets are stored in the Integrated File System (IFS).

As MG/EE can make calls to individual Business Object/Methods, it is at this level that the XSL stylesheets are generated. In other words, there should be an Input and Output XSL stylesheet for each Business Object/Method that needs to be invoked by MG/EE. A field will be defined in the XSL stylesheet for every field specified in a Business Object/Method. As each Business Object/Method will have an Input and Output copybook created, the content and definition of the Business Object/Method copybooks determines the content and definition of the XSL stylesheet.

Please note that the CRTBOBJ command will generate XSL style sheets for ALL methods defined for a business object. If stylesheets need to be (re-)generated for only one method, then the GENXSL command can be used.

3.2.3. Software Management

These generated members are standard program objects, commands and copybooks and they will be registered to the SMART Software Dictionary. Therefore, the normal commands such as ED, CB, PR, DLTOBJLVL and their graphical equivalents are available.

The Business Object definition is linked to the Element specified in the "Member Name" field, so when this Element is promoted the Business Object definition records and related Field / XML Tag Mapping records are transferred to the target Level and propagated to all Levels below the Target Level. Should the Business Object exist in one of these lower Levels (Dictionary is checked for "Member Name") the definition records will not be copied and a Duplicate Memo report for the Business Object will be generated.

3.2.4. Summary

If a business object named CLT had three methods named CRT, CHG and DLT, then the following source members and types would be generated.

	CLT	*CBL
Business Object		
Testrig CPP	CLTCRTX	*CBL
	CLTCHGX	*CBL
	CLTDLTX	*CBL
Input/Output Copybook	CLTCRTI	*CPY
	CLTCRTO	*CPY
	CLTCHGI	*CPY
	CLTCHGO	*CPY

	CLTDLTI	*CPY
	CLTDLTO	*CPY
	CLTCRT	*CMD
Testrig command		
	CLTCHG	*CMD
	CLTDLT	*CMD
XSL Stylesheets (*)	SMART400_CLT_CRT_IN	*XSL
	SMART400_CLT_CRT_OUT	*XSL
	SMART400_CLT_CHG_IN	*XSL
	SMART400_CLT_CHG_OUT	*XSL
	SMART400_CLT_DLT_IN	*XSL
	SMART400_CLT_DLT_OUT	*XSL
XML Fieldmapping file (*)	SMART400_bo_mth_fieldmapping (**)	*XML

(*) The CRTBOBJPFX Software Default governs the "SMART400_" prefix.

(**) This file is only created for a method if XML tags have been defined for one or more field(s) used by the business object method.

3.2.5. Generated Business Object Example

All the business objects developed have a standard structure - in much the same way as the traditional on-line and batch programs developed using SMART have a standard structure. This structure is shown in the example below for the business object named CLI which handles Clients. The relevant portions of the program are highlighted with an arrow and are explained after the code example.

```

IDENTIFICATION DIVISION.
PROGRAM-ID.                                CLI
.
ENVIRONMENT DIVISION.
WORKING-STORAGE SECTION.
.

⇒ 1
*      Input and Output copybooks for each method included.
COPY CLIALTCRTI.
COPY CLIALTCRTO.
COPY CLICRCI.
COPY CLICRCO.
COPY CLICRPI.
COPY CLICRPO.
COPY CLIENQI.
COPY CLIENQO.
COPY CLIUPCI.
COPY CLIUPCO.
COPY CLIUPPI.
COPY CLIUPPO.
.

⇒ 2
01 VALIDATION-ERROR.
COPY BOVERRREC.
01 FATAL-ERROR.
COPY SFERRREC.
.

⇒ 3
* Screen copybook for each program to call.
COPY S2465SCR.
COPY S2466SCR.
COPY S2472SCR.
COPY S2480SCR.
COPY S2500SCR.
.

⇒ 4
COPY MSGHDR
REPLACING MESSAGE-HEADER BY WSAA-MSGHDR.
* The following copybooks are used in the MAINX copybook and
* should NOT be removed.
COPY LDRHDR
REPLACING LEADER-HEADER BY WSAA-LDRHDR.
COPY MSGDTA
REPLACING MESSAGE-DATA BY WSAA-REQUEST.
COPY MSGDTA
REPLACING MESSAGE-DATA BY WSAA-RESPONSE.
COPY SESSIONO.
.

⇒ 5
LINKAGE SECTION.
COPY LDRHDR.
COPY MSGDTA
REPLACING MESSAGE-DATA BY REQUEST-DATA.
COPY MSGDTA
REPLACING MESSAGE-DATA BY RESPONSE-DATA.
.

⇒ 6
PROCEDURE DIVISION
USING LEADER-HEADER

```

REQUEST-DATA
RESPONSE-DATA.

⇒ 7

```
COPY MAINX.
.
1000-INITIALISE SECTION.
*****
1010-START.

      MOVE O-K                      TO SCRNL-STATUS.
      MOVE '0'                     TO ERLVL OF LEADER-HEADER.

1090-EXIT.
      EXIT.
.
```

⇒ 8

```
2000-CALL-PROGRAMS SECTION.
*****
2010-CALL.

      EVALUATE VRBID OF LEADER-HEADER

          WHEN 'ALTCRT' '
              PERFORM MTH-CLIALTCRT
          WHEN 'CRC' '
              PERFORM MTH-CLICRC
          WHEN 'CRP' '
              PERFORM MTH-CLICRP
          WHEN 'ENQ' '
              PERFORM MTH-CLLENQ
          WHEN 'UPC' '
              PERFORM MTH-CLIUPC
          WHEN 'UPP' '
              PERFORM MTH-CLIUPP
          WHEN OTHER
              PERFORM INVALID-METHOD-PASSED
      END-EVALUATE.

2090-EXIT.
      EXIT.
.
```

⇒ 9

MTH-CLICRP SECTION.

⇒ 9-A

```
MTH-CLICRP-BEGIN.
      MOVE REQUEST-DATA          TO CLICRPI-REC.
```

⇒ 9-B

```
MTH-CLICRP-P2480-1000.
      COPY SCALLORD
          REPLACING SECT          BY '1000'
                      PROG        BY 'P2480'
                      SCREEN-REC   BY S2480-DATA-AREA.
```

⇒ 9-C

```
MTH-CLICRP-P2480-INPUT.
      MOVE BGEN-S2480-ACTION
          OF CLICRPI-REC         TO S2480-ACTION.
```

⇒ 9-D

```
MTH-CLICRP-P2480-PRE.
      COPY SCALLORD
          REPLACING SECT          BY 'PRE'
                      PROG        BY 'P2480'
                      SCREEN-REC   BY S2480-DATA-AREA.
```

⇒ 9-E

```
MTH-CLICRP-P2480-2000.
      MOVE S2480-ACTION          TO SCRNL-ACTION.

      PERFORM SCREEN-CHECK-S2480.

      COPY SCALLORD
          REPLACING SECT          BY '2000'
```

```

PROG          BY 'P2480'
SCREEN-REC    BY  S2480-DATA-AREA.

```

```

PERFORM SET-UP-ERRORS-P2480.
COPY BOVERRCPY
  REPLACING EXIT-PARA    BY MTH-CLICRP-EXIT.

```

⇒ 9-F

```

MTH-CLICRP-P2480-3000.
COPY SCALLORD
  REPLACING SECT          BY '3000'
                PROG       BY 'P2480'
                SCREEN-REC  BY  S2480-DATA-AREA.

PERFORM SET-UP-ERRORS-P2480.
COPY BOVERRCPY
  REPLACING EXIT-PARA    BY MTH-CLICRP-EXIT.

```

⇒ 9-G

```

MTH-CLICRP-P2480-4000.
COPY SCALLORD
  REPLACING SECT          BY '4000'
                PROG       BY 'P2480'
                SCREEN-REC  BY  S2480-DATA-AREA.

```

⇒ 9-H

```

MTH-CLICRP-P2465-1000.
COPY SCALLORD
  REPLACING SECT          BY '1000'
                PROG       BY 'P2465'
                SCREEN-REC  BY  S2465-DATA-AREA.

```

⇒ 9-I

```

MTH-CLICRP-P2465-INPUT.
MOVE BGEN-S2465-ADDRTYPE
  OF CLICRPI-REC          TO S2465-ADDRTYPE.
MOVE BGEN-S2465-CLTSEX
  OF CLICRPI-REC          TO S2465-CLTSEX.
MOVE BGEN-S2465-CTRYCODE
  OF CLICRPI-REC          TO S2465-CTRYCODE.
MOVE BGEN-S2465-GIVNAME
  OF CLICRPI-REC          TO S2465-GIVNAME.
MOVE BGEN-S2465-MARRYD
  OF CLICRPI-REC          TO S2465-MARRYD.
MOVE BGEN-S2465-SALUTL
  OF CLICRPI-REC          TO S2465-SALUTL.
MOVE BGEN-S2465-SURNAME
  OF CLICRPI-REC          TO S2465-SURNAME.

```

⇒ 9-J

```

MTH-CLICRP-P2465-PRE.
COPY SCALLORD
  REPLACING SECT          BY 'PRE'
                PROG       BY 'P2465'
                SCREEN-REC  BY  S2465-DATA-AREA.

```

⇒ 9-K

```

MTH-CLICRP-P2465-2000.

PERFORM SCREEN-ERRORS-S2465.

COPY SCALLORD
  REPLACING SECT          BY '2000'
                PROG       BY 'P2465'
                SCREEN-REC  BY  S2465-DATA-AREA.

PERFORM SET-UP-ERRORS-P2465.
COPY BOVERRCPY
  REPLACING EXIT-PARA    BY MTH-CLICRP-EXIT.

```

⇒ 9-L

```

MTH-CLICRP-P2465-3000.
  COPY SCALLORD
    REPLACING SECT          BY '3000'
              PROG          BY 'P2465'
              SCREEN-REC    BY S2465-DATA-AREA.

  PERFORM SET-UP-ERRORS-P2465.
  COPY BOVERRCPY
    REPLACING EXIT-PARA    BY MTH-CLICRP-EXIT.

```

⇒ 9-M

```

MTH-CLICRP-P2465-OUTPUT.
  MOVE S2465-CLNTNUM          TO BGEN-S2465-CLNTNUM
                                OF CLICRPO-REC.

```

⇒ 9-N

```

MTH-CLICRP-P2500-1000.
  COPY SCALLSFL
    REPLACING SECT          BY '1000'
              PROG          BY 'P2500'
              SCREEN-REC    BY S2500-DATA-AREA
              SUBFILE-REC   BY S2500-SUBFILE-AREA.

```

⇒ 9-O

```

MTH-CLICRP-P2500-PRE.
  COPY SCALLSFL
    REPLACING SECT          BY 'PRE'
              PROG          BY 'P2500'
              SCREEN-REC    BY S2500-DATA-AREA
              SUBFILE-REC   BY S2500-SUBFILE-AREA.

```

⇒ 9-P

```

MTH-CLICRP-P2500-SFLINP.
  MOVE SCLR                      TO SCRNL-FUNCTION.
  CALL 'S2500IO'                 USING SCRNL-SCREEN-PARAMS
                                S2500-DATA-AREA
                                S2500-SUBFILE-AREA.

  IF SCRNL-STATUZ                NOT = O-K
  AND SCRNL-STATUZ                NOT = ENDP
    MOVE SCRNL-STATUZ            TO SYSR-STATUZ
    PERFORM 600-FATAL-ERROR
  END-IF.

  MOVE 00010                     TO WSAA-SFL-SIZE.
  MOVE 1                         TO SCRNL-SUBFILE-RRN.

  PERFORM VARYING WSAA-SFL-CNT
    FROM 1 BY 1
    UNTIL WSAA-SFL-CNT > WSAA-SFL-SIZE

  MOVE BGEN-S2500-ACTION
  OF CLICRPI-REC (WSAA-SFL-CNT)
  TO S2500-ACTION
  MOVE SADD                      TO SCRNL-FUNCTION
  CALL 'S2500IO'                 USING SCRNL-SCREEN-PARAMS
                                S2500-DATA-AREA
                                S2500-SUBFILE-AREA

  IF SCRNL-STATUZ                NOT = O-K
  AND SCRNL-STATUZ                NOT = ENDP
    MOVE SCRNL-STATUZ            TO SYSR-STATUZ
    PERFORM 600-FATAL-ERROR
  END-IF
  END-PERFORM.

```

⇒ 9-Q

```

MTH-CLICRP-P2500-2000.

  PERFORM SCREEN-CHECK-S2500.

```

```

COPY SCALLSFL
  REPLACING SECT          BY '2000'
              PROG        BY 'P2500'
              SCREEN-REC   BY S2500-DATA-AREA
              SUBFILE-REC  BY S2500-SUBFILE-AREA.

PERFORM SET-UP-ERRORS-P2500.
COPY BOVERRCPY
  REPLACING EXIT-PARA     BY MTH-CLICRP-EXIT.

```

⇒ **9-R**

```

MTH-CLTCLT-P2500-3000.
COPY SCALLSFL
  REPLACING SECT          BY '3000'
              PROG        BY 'P2500'
              SCREEN-REC   BY S2500-DATA-AREA
              SUBFILE-REC  BY S2500-SUBFILE-AREA.

PERFORM SET-UP-ERRORS-P2500.
COPY BOVERRCPY
  REPLACING EXIT-PARA     BY MTH-CLICRP-EXIT.

```

⇒ **9-S**

```

MTH-CLTCLT-P2500-OUTPUT.
MOVE S2500-CLNTNUM          TO BGEN-S2500-CLNTNUM
                             OF CLICRPO-REC.

MOVE S2500-CLTDTL          TO BGEN-S2500-CLTDTL
                             OF CLICRPO-REC.

MOVE S2500-OCCPCODE        TO BGEN-S2500-OCCPCODE
                             OF CLICRPO-REC.

MOVE S2500-OCCPDESC        TO BGEN-S2500-OCCPDESC
                             OF CLICRPO-REC.

```

⇒ **9-T**

```

MTH-CLICRP-P2500-SFLOUT.
MOVE SSTR1                  TO SCR1-FUNCTION.
CALL 'S2500IO'              USING SCR1-SCREEN-PARAMS
                              S2500-DATA-AREA
                              S2500-SUBFILE-AREA.

IF SCR1-STATUZ              NOT = O-K
AND SCR1-STATUZ              NOT = ENDP
  MOVE SCR1-STATUZ          TO SYSR-STATUZ
  PERFORM 600-FATAL-ERROR
END-IF.

MOVE 00010                  TO WSAA-SFL-SIZE.
PERFORM VARYING WSAA-SFL-CNT
  FROM 1 BY 1
  UNTIL WSAA-SFL-CNT > WSAA-SFL-SIZE
  INITIALIZE BGEN-S2500-SFL OF CLICRPO-REC (WSAA-SFL-CNT)
END-PERFORM.

PERFORM VARYING WSAA-SFL-CNT
  FROM 1 BY 1
  UNTIL WSAA-SFL-CNT > WSAA-SFL-SIZE
  OR SCR1-STATUZ = ENDP

MOVE S2500-ACTION          TO BGEN-S2500-ACTION
                             OF CLICRPO-REC (WSAA-SFL-CNT)

MOVE S2500-DECL-GR-SALARY  TO BGEN-S2500-DECGRSAL
                             OF CLICRPO-REC (WSAA-SFL-CNT)

MOVE S2500-INCOME-DESC     TO BGEN-S2500-INCDISC
                             OF CLICRPO-REC (WSAA-SFL-CNT)
MOVE S2500-INCOME-SEQ-NO   TO BGEN-S2500-INCSQNO
                             OF CLICRPO-REC (WSAA-SFL-CNT)

MOVE S2500-TAX-YEAR        TO BGEN-S2500-TAXYR
                             OF CLICRPO-REC (WSAA-SFL-CNT)

MOVE SRDN                  TO SCR1-FUNCTION
CALL 'S2500IO'              USING SCR1-SCREEN-PARAMS
                              S2500-DATA-AREA

```

S2500-SUBFILE-AREA

```

IF  SCR-STATUS      NOT = O-K
AND SCR-STATUS      NOT = ENDP
  MOVE SCR-STATUS    TO SYSR-STATUS
  PERFORM 600-FATAL-ERROR
END-IF
END-PERFORM.

```

⇒ 9-U

```

MTH-CLICRP-END.
  MOVE LENGTH OF MESSAGE-DATA OF CLICRPO-REC
                                TO MSGLN OF CLICRPO-REC.
  MOVE 1                        TO MSGCNT OF CLICRPO-REC.
  MOVE 'CLICRPO '              TO MSGID  OF CLICRPO-REC
  IF NO-ERROR OF LEADER-HEADER
    MOVE CLICRPO-REC           TO RESPONSE-DATA
    MOVE LENGTH OF CLICRPO-REC TO TOTMSGLN OF LEADER-HEADER
  END-IF.

MTH-CLICRP-EXIT.
  EXIT.
.

```

⇒ 10

```

MTH-CLICRC SECTION.
MTH-CLICRC-BEGIN.
.
MTH-CLOCRC-EXIT.
  EXIT.
.
MTH-CLIEHQ SECTION.
MTH-CLIEHQ-BEGIN.
.
MTH-CLIEHQ-EXIT.
  EXIT.
.

```

⇒ 11

```

INVALID-METHOD-PASSED SECTION.
*****
INVALID-METHOD-PASSED-START.

  MOVE 'IVMT'              TO SYSR-STATUS.
  MOVE WSAA-PROG           TO SYSR-SUBRNAME.
  MOVE SPACES              TO SYSR-DBPARAMS.
  STRING 'Invalid method passed: ' DELIMITED BY SIZE
                                VRBID OF LEADER-HEADER DELIMITED BY SIZE
                                INTO SYSR-DBPARAMS

  END-STRING.
  MOVE '2'                 TO SYSR-SYSERR-TYPE.
  PERFORM 600-FATAL-ERROR.

INVALID-METHOD-PASSED-EXIT.
  EXIT.
.

```

⇒ 12A

```

SCREEN-CHECK-S2465 SECTION.
*****
SCREEN-CHECK-S2465-START.
*   This section holds code to perform the validation
*   that is normally performed within the screenIO
*   program for date, time and window fields.

    IF S2465-ADDRTYPE-ERR                = SPACES
    AND S2465-ADDRTYPE                  NOT = SPACES
        MOVE SPACES                      TO RITM-READITM-REC
        MOVE 'READ'                      TO RITM-FUNCTION
        MOVE SPACES                      TO WSAA-ITEMKEY
        MOVE 'IT'                        TO WSKY-ITEM-ITEMPFX
        MOVE WSSP-FSUCO                  TO WSKY-ITEM-ITEMCOY
        MOVE 'T3690'                     TO WSKY-ITEM-ITEMTABL
        MOVE S2465-ADDRTYPE              TO WSKY-ITEM-ITEMITEM
        MOVE WSAA-ITEMKEY                TO RITM-ITEMKEY
        CALL 'READITM' USING RITM-READITM-REC
        IF RITM-STATUZ                    NOT = O-K
            MOVE 'E031'                   TO S2465-ADDRTYPE-ERR
        END-IF
    END-IF

    IF S2465-CLTDOBX-ERR                 = SPACES
    AND S2465-CLTDOBX                    NOT = 99999999
        MOVE 'CONV'                       TO DTC1-FUNCTION
        MOVE S2465-CLTDOBX                TO DTC1-INT-DATE
        CALL 'DATCON1'                     USING DTC1-DATCON1-REC
        IF DTC1-STATUZ NOT = O-K
            MOVE 'E032'                   TO S2465-CLTDOBX-ERR
        END-IF
    END-IF

SCREEN-CHECK-S2465-EXIT.
EXIT.

```

⇒ 12B

```

SCREEN-CHECK-S2465 SECTION.
*****
SCREEN-CHECK-S2465-START.
*   This section holds code to perform the validation
*   that is normally performed within the screenIO
*   program for date, time and window fields.

    MOVE 'VAL'                           TO VAL-FUNCTION.
    CALL 'S2465VAL'                       USING SCRN-SCREEN-PARAMS
                                           S2465-DATA-AREA
                                           VAL-FUNCTION
                                           VAL-ERRORS.

SCREEN-CHECK-S2465-EXIT.
EXIT.

```


⇒ 13A

```

SET-UP-ERRORS-P2465 SECTION.
*****
SET-UP-ERRORS-P2465-START.

    MOVE 'P2465'                TO BOVERR-PROG.
    MOVE 1                      TO WSAA-CNT.

    IF SCRN-ERROR-CODE          NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE SCRN-ERROR-CODE    TO BOVERR-EROR    (WSAA-CNT)
        MOVE 'S2465-SCRN-ERROR-CODE' TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

    IF S2465-ADDRDESC01-ERR      NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE S2465-ADDRDESC01-ERR TO BOVERR-EROR    (WSAA-CNT)
        MOVE 'S2465-ADDRDESC01'  TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

.
SET-UP-ERRORS-P2465-EXIT.
EXIT.
.

```

⇒ 13B

```

SET-UP-ERRORS-P2465 SECTION.
*****
SET-UP-ERRORS-P2465-START.

    MOVE 'P2465'                TO BOVERR-PROG.
    MOVE 1                      TO WSAA-CNT.

    IF SCRN-ERROR-CODE          NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE SCRN-ERROR-CODE    TO BOVERR-EROR    (WSAA-CNT)
        MOVE 'S2465-SCRN-ERROR-CODE' TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

    MOVE 'ERR'                  TO VAL-FUNCTION.
    CALL 'S2465VAL'              USING SCRN-SCREEN-PARAMS
                                   S2465-DATA-AREA
                                   VAL-FUNCTION
                                   VAL-ERRORS.

    PERFORM VARYING VAL-X
        FROM 1 BY 1
        UNTIL VAL-X              > VAL-ERROR-CNT
        OR WSAA-CNT              > 25

        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE VAL-ERROR(VAL-X)   TO BOVERR-EROR    (WSAA-CNT)
        MOVE VAL-FIELD(VAL-X)   TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-PERFORM.

.
SET-UP-ERRORS-P2465-EXIT.
EXIT.
.

```

⇒ 14

```

SCREEN-CHECK-S2480 SECTION.
*****
SCREEN-CHECK-S2480-START.
*   This section holds code to perform the validation
*   that is normally performed within the screenIO
*   program for date, time and window fields.
    CONTINUE.

SCREEN-CHECK-S2480-EXIT.
    EXIT.

/
SET-UP-ERRORS-P2480 SECTION.
*****
SET-UP-ERRORS-P2480-START.

    MOVE 'P2480'                TO BOVERR-PROG.
    MOVE 1                      TO WSAA-CNT.

    IF SCRN-ERROR-CODE          NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE SCRN-ERROR-CODE   TO BOVERR-EROR   (WSAA-CNT)
        MOVE 'S2480-SCRN-ERROR-CODE' TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

    IF S2480-ACTION-ERR          NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE S2480-ACTION-ERR   TO BOVERR-EROR   (WSAA-CNT)
        MOVE 'S2480-ACTION'     TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

    IF S2480-CLTTWO-ERR          NOT = SPACES
        MOVE ZEROES            TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE S2480-CLTTWO-ERR   TO BOVERR-EROR   (WSAA-CNT)
        MOVE 'S2480-CLTTWO'     TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

SET-UP-ERRORS-P2480-EXIT.
    EXIT.
.

```

⇒ 15

```

SCREEN-CHECK-S2500 SECTION.
*****
SCREEN-CHECK-S2500-START.
*   This section holds the code to call the screen
*   validation program that will perform the
*   validation for date, time, window fields and DBCS.
    CONTINUE.

SCREEN-CHECK-S2500-EXIT.
    EXIT.

```

⇒ 16

```

SET-UP-ERRORS-P2500 SECTION.
*****
SET-UP-ERRORS-P2500-START.

    MOVE 'P2500'                TO BOVERR-PROG.
    MOVE 1                      TO WSAA-CNT.

    IF SCR-ERROR-CODE            NOT = SPACES
        MOVE ZEROES             TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE SCR-ERROR-CODE      TO BOVERR-ERROR  (WSAA-CNT)
        MOVE 'S2500-SCR-ERROR-CODE' TO BOVERR-FIELD (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-IF.

    MOVE 'ERR'                  TO VAL-FUNCTION.
    CALL 'S2500VAL'             USING SCR-SCREEN-PARAMS
                                   S2500-DATA-AREA
                                   S2500-SUBFILE-AREA
                                   VAL-FUNCTION
                                   VAL-ERRORS.

    PERFORM VARYING VAL-X
        FROM 1 BY 1
        UNTIL VAL-X > VAL-ERROR-CNT
              OR WSAA-CNT > 25

        MOVE ZEROES             TO BOVERR-BOFOCCUR(WSAA-CNT)
        MOVE VAL-ERROR(VAL-X)    TO BOVERR-ERROR  (WSAA-CNT)
        MOVE VAL-FIELD(VAL-X)    TO BOVERR-FIELD  (WSAA-CNT)
        PERFORM 8100-GET-ERROR-DESC
    END-PERFORM.

    IF WSSP-SECTIONNO            NOT = '2000'
        GO TO SET-UP-ERRORS-P2500-EXIT
    END-IF.

    MOVE 00010                  TO WSAA-SFL-SIZE.
    PERFORM VARYING SCR-SUBFILE-RRN
        FROM 1 BY 1
        UNTIL SCR-SUBFILE-RRN > WSAA-SFL-SIZE
              OR SCR-STATUZ      = MRNF

        MOVE SREAD              TO SCR-FUNCTION
        CALL 'S2500IO'          USING SCR-SCREEN-PARAMS
                                   S2500-DATA-AREA
                                   S2500-SUBFILE-AREA

        IF SCR-STATUZ            NOT = O-K
        AND SCR-STATUZ            NOT = MRNF
            MOVE SCR-STATUZ      TO SYSR-STATUZ
            PERFORM 600-FATAL-ERROR
        END-IF

        IF SCR-STATUZ            NOT = MRNF
            PERFORM SET-UP-ERRORS-P2500-SFL
        END-IF
    END-PERFORM.

SET-UP-ERRORS-P2500-EXIT.
EXIT.

```

⇒ 17

```

SET-UP-ERRORS-P2500-SFL SECTION.
*****
SET-UP-ERRORS-P2500-SFL-START.

      MOVE 'SERR'                TO VAL-FUNCTION.
      CALL 'S2500VAL'            USING SCRN-SCREEN-PARAMS
                                   S2500-DATA-AREA
                                   S2500-SUBFILE-AREA
                                   VAL-FUNCTION
                                   VAL-ERRORS.

      PERFORM VARYING VAL-X
            FROM 1 BY 1
            UNTIL VAL-X          > VAL-ERROR-CNT
            OR WSAA-CNT          > 25

            MOVE VAL-BOFOCCUR(VAL-X) TO BOVERR-BOFOCCUR(WSAA-CNT)
            MOVE VAL-ERROR(VAL-X)    TO BOVERR-ERROR  (WSAA-CNT)
            MOVE VAL-FIELD(VAL-X)    TO BOVERR-FIELD  (WSAA-CNT)
            PERFORM 8100-GET-ERROR-DESC
      END-PERFORM.

SET-UP-ERRORS-P2500-SFL-EXIT.
EXIT.

```

⇒ 18

```

8100-GET-ERROR-DESC SECTION.
*****
8110-START.
      IF WSAA-CNT                > 25
        GO TO 8190-EXIT
      END-IF.

      MOVE SPACES                TO WSAA-DESC.

      MOVE WSSP-LANGUAGE          TO WSAA-LANG
      CALL 'GETERROR'            USING SVERR-ERROR(WSAA-CNT)
                                   WSAA-LANG
                                   WSAA-DESC
                                   WSAA-STAT.

      IF WSAA-DESC                = SPACES
        MOVE SPACES              TO WSAA-LANG
        CALL 'GETERROR'          USING SVERR-ERROR(WSAA-CNT)
                                   WSAA-LANG
                                   WSAA-DESC
                                   WSAA-STAT
      END-IF.

      IF WSAA-STAT                = O-K
        MOVE WSAA-DESC            TO SVERR-DESC(WSAA-CNT)
      END-IF.

      ADD 1                      TO WSAA-CNT.

8190-EXIT.
EXIT.

```

⇒ 19

```

9000-UPDATE-WSSP SECTION.
*****
9010-PARA.
      CONTINUE.

9090-EXIT.
EXIT.

```

It will be helpful to now present a guided tour through the program to explain how it came to be shaped this way. Key features of the program, highlighted with an arrow, are explained below:

⇒ 1

The various request and response messages which detail the input and output data to and from the business object for a specific method. These copybooks will be used throughout the business object.

⇒ 2

These are the standard SMART Validation and Fatal Error routine copybooks. The Validation Error record will be used after calling the 2000- section of a 'P' program. The Fatal Error record will be used for any other error.

⇒ 3

All the screen copybooks associated with the P-programs called in this business object have to be included.

⇒ 4

The Message Header, Leader Header and Message Data copybooks can be used more than once in a business object. Here the 01 levels are replaced by WSAA- variables that will be used to temporarily store the values.

⇒ 5

This is the linkage area of the business object. This linkage always has the same layout and consists of three parts. The first part is the Leader Header, the second and third parts are the Request data and Response data respectively. The Request and Response data fields are 32000 bytes each in size. The format of the request and response data depends on the method. As can be seen at 9-A, the request data will be mapped to the method input copybook. At 9-U, the response data is based on the method output copybook.

⇒ 6

The driving copybook in a business object is MAINX. The structure of this copybook is:

- Retrieve and save WSSP values
- Execute section called 1000-INITIALISE
- Execute section called 2000-CALL-PROGRAMS
- Execute section called 9000-UPDATE-WSSP
- Update WSSP values with saved values.

⇒ 7

The purpose of this section speaks for itself – it can be used for initialisation purposes and to perform initial database reads if needed.

⇒ 8

This section decides which method needs to be executed, based on the Vrbid parameter on the Leader Header. If an unknown method is passed, an error will be produced and the business object will terminate.

⇒ 9

Per method, a section will be generated to handle all the processing required for this method. The structure of the section will be described in detail below. After generation of the business object, this section can be modified to meet the specific needs for the particular method. This can be done by manually editing the business object COBOL program, but it is recommended to use the Insert Source feature in the DSNBOBJ command to allow easy regeneration of the business object. Any existing Inserted Source will be automatically added to the generated business object program at the appropriate place in the code.

⇒ 9-A

Every method section starts with the move of the request data to the method input copybook. Once this has happened, the rest of the method can use the fields as defined within the input copybook, without having to worry about the positions and lengths of these fields within the Request data.

⇒ 9-B

For each of the sections selected during DSNBOBJ, a call to the Pnnnn programs needs to be executed. There are two copybooks used within a business object that contain the call to the Pnnnn program. This is SCALLORD for ordinary screens and SCALLSFL for subfile screens. The generator will determine what type of screen is associated with the Pnnnn program and generate the copy of the appropriate copybook.

For all selected programs for a method, the code at ⇒ 9-B through to ⇒ 9-G can be generated, depending on which sections are selected during DSNBOBJ.

If the 1000 section is selected during the DSNBOBJ, this code is generated.

⇒ 9-C

This code will ensure that the values in the method input copybook (in other words, the Request data) are used to populate the relevant Snnnn screen fields. For all screen fields selected as Input or Both during DSNBOBJ for this particular program, a MOVE statement will be generated.

⇒ 9-D

This code will execute the call to the Pnnnn program for the PRE section. The code is only generated if the PRE section is selected during the DSNBOBJ.

⇒ 9-E

This code will execute the call to the Pnnnn program for the 2000 section. The code is only generated if the 2000 section is selected during the DSNBOBJ.

Prior to the call to the Pnnnn program -

If the screen contains the field ACTION code is generated to populate the field SCRIN-ACTION.

The SCREEN-CHECK-Snnnn SECTION is performed. This section will perform the validation that is normally performed in the screen IO module i.e. validate date and time fields and validate fields that window to tables.

After calling the 2000 section of the Pnnnn program, the section SET-UP-ERRORS-Pnnnn is executed. This section will check for screen errors and, in case of errors, the BOVERRCPY copybook will ensure that the method ends and the fields in error, with the error codes and descriptions, are returned.

⇒ 9-F

This code will execute the call to the Pnnnn program for the 3000 section. The code is only generated if the 3000 section is selected during the DSNBOBJ.

Just like after the call to the 2000 section, the SET-UP-ERRORS-Pnnnn section will be executed after the 3000 section. This is because there are programs in the system (mainly submenu programs) that can return errors in the 3000 section.

⇒ 9-G

This code will execute the call to the Pnnnn program for the 4000 section. The code is only generated if the 4000 section is selected during the DSNBOBJ.

⇒ 9-H

The first program has now been executed for all the requested sections. Now the next program as specified during DSNBOBJ will be called. This piece of code is constructed in the same way as at ⇒ 9-B.

⇒ 9-I through 9-M

Similar to ⇒ 9-C through ⇒ 9-G.

⇒ 9-N

The second program has now been executed for all requested sections. Now the next program as specified during DSNBOBJ will be called. In this example the next program is a subfile program.

This section of code is constructed the same way as ⇒ 9-CB

⇒ 9-O

Similar to ⇒ 9-D

⇒ 9-P

This section will update the input capable fields on the subfile with the values from the input method copybook.

⇒ 9-Q

Similar to ⇒ 9-E

⇒ 9-R

Similar to ⇒ 9-F

⇒ 9-S

This section will ensure that the non subfile fields in the output method copybook are populated with the values from the screen.

⇒ 9-T

This section will ensure that the subfile fields in the output method copybook are populated with the values from the screen.

⇒ 9-U

At the end of the method, the message length, count and ID are updated in the Message Header. Then, if no errors have occurred, the Request data is set up based on the method output copybook and the total message length is updated.

⇒ 10

For all other methods, similar sections will be generated.

⇒ 11

If an invalid method is passed, a status of IVMT will be returned with an appropriate description.

⇒ 12A

This version of SCREEN-CHECK-Snnnn SECTION will be generated if a screen validation module does not exist for the screen. The screen validation module will not exist if the screen has last been compiled on a SMART release prior to release 0404.

This section is executed after the call to the 2000 section of a Pnnnn program. All input capable fields that are date or time fields or fields that allow windowing will be checked for errors. This processing normally takes place within the screen IO module but, as the business object does not call IO modules, the checking has to be performed here.

If fields are added to or removed from the screen, this section should be amended accordingly. This is simply a matter of regenerating the business object using the CRTBOBJ command. Please note that in case the Insert Source feature is not used and manual amendments have been made to the business object, that these amendments will be lost upon regeneration. In such a case it is advised to make a copy of the original business object code so that the manual code can be added in after the regeneration, or better still, to add the manual code as Inserted Source.

⇒ 12B

This version of SCREEN-CHECK-Snnnn SECTION will be generated if a screen validation module exists for the screen. The screen validation module will be created if the screen has been compiled under SMART release 0404 or later.

The SnnnnVAL module is called with a function of VAL to validate the screen fields. The screen validation module is recreated each time the screen is recompiled. This means that any changes to the online screen validation will be performed by the Business Object without having to change the Business Object code.

⇒ 13A

This version of SET-UP-ERRORS-Pnnnn SECTION will be generated if a screen validation module does not exist for the screen. The screen validation module will not exist if the screen has last been compiled on a SMART release prior to release 0404.

This section ensures that all screen errors are trapped and returned to the front-end system. The SET-UP-ERRORS-Pnnnn section will return up to 25 fields in error. For each field in error, the field name, the error code and the error description will be returned.

As with the SCREEN-CHECK-Snnnn section, this section might need to be amended when the screen is changed.

⇒ 13B

This version of SCREEN-CHECK-Snnnn SECTION will be generated if a screen validation module exists for the screen. The screen validation module will be created if the screen has been compiled under SMART release 0404 or later.

The screen validation module SnnnnVAL is called with a function of ERR to return all screen errors.

Because the screen validation module is regenerated each time the screen is recompiled this section does not need to be changed if the screen validation is changed.

⇒ 14

As can be seen, SCREEN-CHECK-Snnnn and SET-UP-ERRORS-Pnnnn sections will be included for all screens that play a role in this business object.

⇒ 15

This section will perform the validation for the non subfile fields that normally takes place in the screen IO module. If a screen validation module exists for the screen it will be called with a function of VAL. If a screen validation module does not exist the code to perform the validation will be generated in this section.

In this example there are no non subfile input capable fields therefore no validation code is generated for this section.

⇒ 16

This section will set up the errors for the non subfile fields in the screen. If a screen validation module exists it will be called with a function of ERR to set up the error codes for the non subfile fields. If a screen validation module does not exist code to return the error codes will be generated in this section.

This section will also call the screen IO module with a function of SREAD in order to perform the screen IO module validation for each line in the subfile. It will perform the SET-UP-ERRORS-Pnnnn-SFL section for each line in the subfile to return the error codes.

⇒ 17

This section will return the error codes for the subfile fields in the screen. If a screen validation module exists it will be called with a function of SERR. If there is no screen validation module code will be generated to return the error codes.

⇒ 18

This section is called from the SET-UP-ERRORS-Snnnn section and should not be removed if the section is used.

⇒ 19

This section can be used to update WSSP. This can be particularly useful if a business object calls other business objects in turn, in which case WSSP sometimes needs to be updated to ensure that the processing functions correctly.

3.2.6. Development Utilities – Field / XML Tag Mapping

Company 0, Development Utilities contains a submenu Field / XML Tag Mapping.

This submenu is used to create, modify and delete the mapping of Fields against XML Tags.

The tags are used during generation of the Input and Output eXtensible Style Sheet (XSL) elements by the CRTBOBJ (Create Business Object) command and the GENXSL command.

It is possible to enter more than one tag per field, to allow the use of a different tag for the same field on the Input and Output XSL. The developer will have an option to select the appropriate tag when using the DSNBOBJ command.

Currently the entry of tags is free-format, the idea being that they conform to a certain standard, for example the ACORD standard.

3.2.6.1. S0632 – Field / XML Tag Mapping

IGBFD - SYSTEM		Field / XML Tag Mapping		S0632 01	
Type Option. Press Enter.					
1-Create		2-Modify		4-Delete	
Position to:					
Contains :					
Sel	Field	XML Tag			
	ACCTCURR	AccountingCurrency			
	ACCTCURR	Currency			
	ACCTYEAR	AccountingYear			
	COMPONENT	ContractComponent			
	CURR	Currency			
					Bottom
F1=Help F3=Exit F4=Prompt F5=Refresh					

Use the "Position to" fields to start the display at the specified point in the list of Fields.

Use the "Contains" fields to place a filter over the list of Fields and XML Tags. Only Field names and tags matching the selection criteria will be displayed.

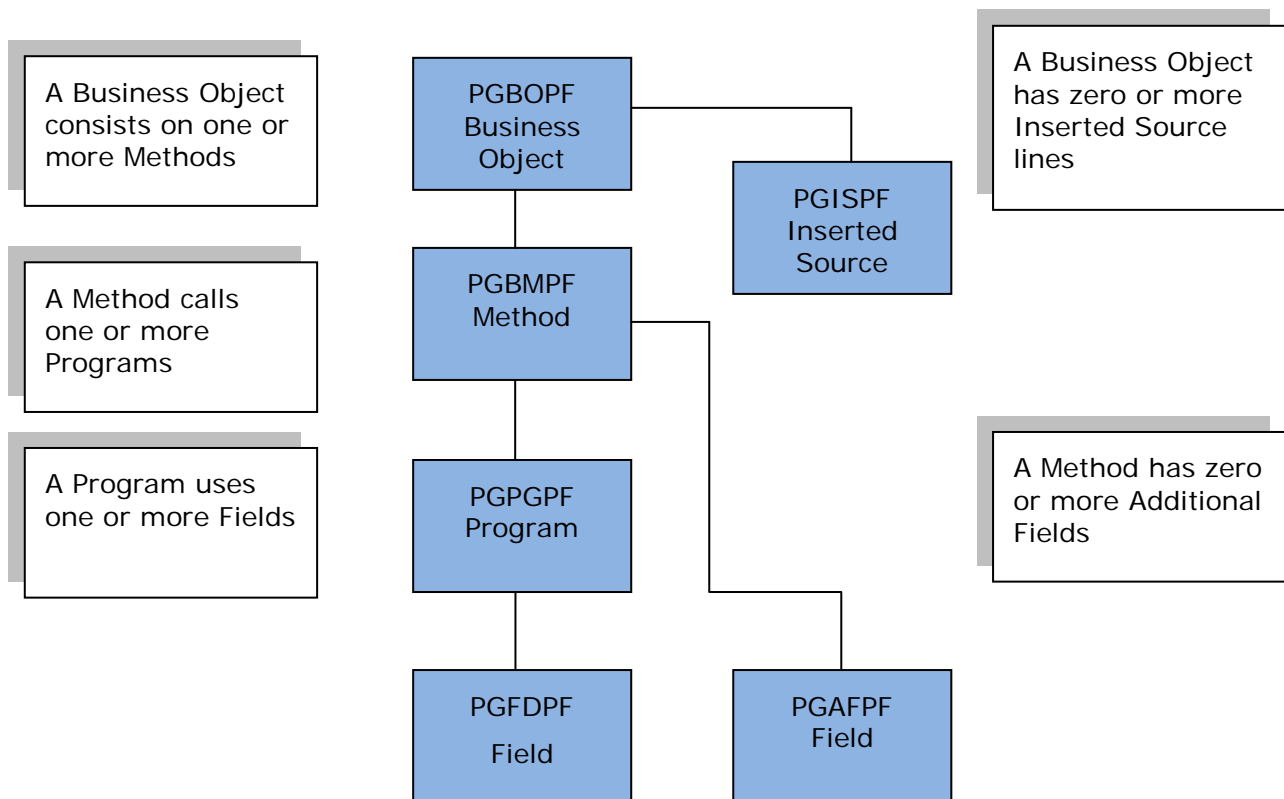
Please note that the XML Tag fields are case-sensitive.

The options 1-Create, 2-Modify and 4-Delete will show dialog S0633:

IGBFD - SYSTEM	Field / XML Tag Mapping	S0633 01
Field	: ACCTCURR	
XML Tag	: AccountingCurrency	
Are you sure? N		
F1=Help F3=Exit F12=Cancel		

The 'Are you sure' line will only be displayed in Delete-mode.

3.2.7. Data Model



3.3. TSTBOBJ

This command allows the user to test the business object for correctness. This has been superseded with an online subsystem that allows easier control of the testing function by allowing clone, delete, rerun and printing of test cases. TSTBOBJ is still valid for use, but is more limited in its functionality.

Please refer to **Business Object Test Tool** Section for more information.

3.3.1. Parameters

Input Library - INPLIB

The library name containing the input file. This field is mandatory and the library must exist.

Input File - INPFILE

The file in the input library containing the input file member. This field is mandatory and the file must exist.

Input Member - INPMBR

The file member containing the input to the business object. This field is mandatory and the file member must exist.

Output Library - OUTLIB

The library name containing the output file. This library must exist. The parameter defaults to the input library (*INPLIB)

Output File - OUTFILE

The file in the output library containing the output file member. This file must exist.

Output Member - OUTMBR

The file member containing the output to the business object. If this file member does not exist, it will be created by the command. The output member must either have a different location to the input member, or must have a different name.

3.3.2. Input and Output files used by TSTBOBJ

3.3.2.1. Creation of files

The input and output files used by TSTBOBJ must be created manually by using the CRTSRCPF command. On this command, a file length of 112 must be specified.

This length is chosen for ease of use when determining the positions of values within the input/output request data. Source files consist of three fields: a sequence number, a date and the actual source line. Choosing a record length of 112 for the file ensures that each source line has room for exactly 100 characters.

The choice has been made to use a Source Physical File instead of a Database Physical File, as Source file members can be easily modified using PDM. The lines are numbered and the COLS function is available to easily display the position numbers of a particular value within the source member.

3.3.2.2. Layout of input file member

The input file members must adhere to the standard defined below.

```
Columns . . . : 1 71          Browse          BOTSTDTA/INP
SEU==>          CLTRTV
***** Beginning of data *****
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+ ...7
0001.00 SESSIONI 0001000001      210E200003
0002.00 00000001PETERX      Wksid      CLT      RTV      001390YIRN0
0003.00 CLTRTVI 0000900001      D50000005
***** End of data *****
```

The first line always contains the session details. For the layout and field positions, see the SESSIONI copybook.

The second line contains the leader header information. For the layout and field positions, see the LDRHDR copybook.

The third and all following lines contain the actual request data that needs to be passed to the business object. The format of the data is dependent on the method. The first part will always be the message header (for the layout, see the MSGHDR copybook), followed by the message data.

3.3.2.3. Output

The command will produce a report as well as storing the output in the file member specified on the command. The output file member will have the same layout as the input file member.

The printout will contain the names of the input/output file members, the session input and the business object request and response data. Any errors encountered will also be printed.

- This page is intentionally left blank –

4. Business Object Design/Build Considerations

4.1. Conversion

The new Client-Server structure with screen/program inversion is not imposed on existing customers.

A data area, CLTSRV, exists which governs whether conversion takes place or not. If programs are to remain in the old format, this data area should have the value *NO. If the value is *YES, a program and screen are converted when the screen is recompiled. If the program is not in the environment, it is copied down automatically.

A converted screen DDS and COBOL program can be identified as follows:

1. A comment line is inserted at the top of the source. Note that this line must not be removed, as it is this line that indicates to SMART on compilation that this program has already been converted. If the line is taken out, the conversion will be performed again, which will lead to incorrect code resulting in compilation errors.
2. MAINF is replaced with MAING in the Procedure Division of the online program.

The old structure produces the following IBM i objects:

- Pnnnn *PGM
- Snnnn *FILE
- SnnnnIO *PGM

The new structure is as follows for ordinary screens (no subfile):

- Pnnnn *PGM
- Snnnn *FILE
- Snnnn *PGM
- SnnnnVAL *PGM

And for subfiles:

- Pnnnn *PGM
- Snnnn *FILE
- Snnnn *PGM
- SnnnnIO *PGM
- SnnnnVAL *PGM

Note that, for subfiles, the SnnnnIO is not a screen IO module in the traditional sense. Subfiles are a feature of the IBM i screen files and are not available to us when we remove the screen portion of the relationship. To retain the subfile functionality, therefore, a data queue is used. The SnnnnIO program manipulates this data queue.

Existing COBOL code is scanned and replaced to enforce the new structure. Unfortunately, this conversion is not forgiving and can produce a non-working program, if the program was not structured in a standard way. Non-standard programs identified in the base systems have been manually converted and will have been delivered to sites with the A9604 release.

An additional section, called PRE-, is introduced between the 1000- and 2000- sections. This section will usually be a dummy, but will have code in it if the conversion finds logic before the SnnnnIO call at the beginning of the 2000- section. In such cases, it is important that any business object calling this program should include the PRE- section at the appropriate point

4.2. DRIVER Run-Time support

SMART supports client-server and non client-server style screens & programs. This means that you do not have to convert all screens and programs at the same time. Instead, an incremental approach can be used whereby only those screens & programs used by business objects are converted to client-server format.

The way this works is that SUBDRIVER checks for the presence of a Snnnn program. If found, this Snnnn program is called, otherwise the Pnnnn program will be executed. In case a Snnnn program is present, it is this program that will call the Pnnnn program. This means that both pre- and post-conversion structures are supported.

4.3. Standards

- Names of business objects must begin with an alphabetic character and be up to 10 characters long.
- Business objects must be written in ANSI85 COBOL and must be portable. No platform specific extensions are allowed.
- No SQL updates are allowed as these are not under commitment control.
- All parameter fields are display format strings, with no string terminator.
- Use of computation numerics is prohibited. In other words, no COMP fields.
- Numerics must be right aligned, padded with leading zeros. Signs must be separate and trailing. For example, positive premium of one hundred and twenty dollars and seventy five cents stored in a 7 digit field with 2 decimal places will be represented as '0012375 ' or '0012375+'. The negative equivalent is '0012375-'.
- Dates must be passed as display format in ISO format. An example is '19950417'.
- Alphanumeric fields are normally left justified and padded right with spaces. For example 'Mr J Botting '.
- Where systems require DBCS (Japanese Kanji, Korean Hanggul, Chinese Modern and Chinese Traditional) double byte strings will contain a Shift-In / Shift-Out code pair bracketing the string.
- Field names within the business object parameter list will consist of the field names as registered in the field dictionary - not the 'Name Used in Program'.
- Field names must not be COBOL reserved words.

4.4. Using the Business Object Generator

The business object generator has distinct productivity advantages and enforces standard structures.

It is recommended that the generator should be used as follows:

1. Select only the sections necessary to the Method for each program selected. Note that for Inquiry, only the 1000- section is required; for update, you may only need the 2000- and 3000- sections; for create you will need 1000-, 2000- and 3000- sections; in some cases the PRE- section will also be required. The 4000- section is not usually required, since the business object is itself controlling the switching. For submenus, however, and programs using GENSSW or OPTSWCH there may be some WSSP or KEEPS functionality in the 4000- section that is required.
2. When selecting fields for the copybooks, only select fields that are relevant to the front-end system or essential to the application. Description fields are not likely to be relevant as input fields for example. The following rules may be useful guidelines:
 - Display only fields should be Output fields
 - Input fields should be either Input or Both fields
 - Any field which is validated by a 2000- section should be included as Input.
3. Use the CRTBOBJ command to submit the jobs to create the following:
 - The business object COBOL module
 - The input copybook for the method
 - The output copybook for the method
 - The test rig
 - The XSL input and output stylesheets and if required, the XML fieldmapping file. (If you just want to (re-)generate the XSL stylesheets for a particular method, the GENXSL can also be used).
4. Use the Insert Source feature during DSNBOBJ to manually add code. The Insert Source feature will store these code fragments in database files, which allows the developer to change and regenerate the business object without losing any of the manually coded functionality.

4.5. The Front-end System

The obvious question to be asked is why. What is the business trying to achieve? What are the expected benefits? Has the front-end system been chosen? If so, what is its architecture, database requirements, development cycle, critical functions, interfaces, internal logic, etc?

Increasingly, the customer has a need to introduce performance improvements, telephony centres and Internet access. Client-server enablement is the appropriate response to these business needs. Any development, however, must be spearheaded from the front-end system, not the applications on the host system.

A well-designed front-end system will encompass process-engineering principles. For example, the screen used by a telephony operator will guide the operator and the caller through a carefully designed script to provide both parties with sufficient information to conclude business satisfactorily in the shortest possible time, avoiding redundancy and unnecessary double-takes. The system will usually include some kind of workflow system that drives the back office.

The LiFE and POLISY systems were designed to capture data relating to a specific database entity, one screen usually per major database file. This is not the way the front-end system will operate. If it does, it is unlikely to provide the performance improvements expected.

Consequently, it is pointless creating business objects until the process design of the front-end system is complete. The IBM i consultants are important in this design phase since the data captured must still satisfy the needs of the LiFE or POLISY systems. In what order these data are captured, however, is the responsibility of the front-end system designer. For this reason, it is recommended that the two teams work closely together, otherwise the front-end system could require major re-engineering of the IBM i application.

4.6. Business Objects

The business object can be seen as a COBOL program that does the following:

- Unpacks the request data
- Interprets the method (function)
- Calls the appropriate sections of the required online programs to complete the method
- Packs the response data.

Validation and most other logic is controlled by the existing, converted online programs.

There are two broad categories of business objects to be considered.

4.6.1. Standard Business Object

A standard business object is one that calls 'P' programs and/or subroutines not derived from screen programs.

This category can further be split into two based upon the number of programs, or multiple occurrences of the same program, in the program stack.

Simple

A simple standard business object is one that calls one or two (usually no more) 'P' programs and/or subroutines. The most common example would be calling the transaction program (therefore one program in the stack) or calling the submenu followed by the transaction program (therefore two programs in the stack).

Complex

When emulating GENSSW/OPTSWCH program switching one will normally need to have the same program appearing more than once in the program stack – each entry using different sections of the 'P' program. Code will also normally need to be inserted into the business object to switch check boxes on or off between the program calls.

4.6.2. Super Business Objects

These business objects call other business objects and/or subroutines. They are required when the same business object may need to be called more than once (e.g. we need to call the business object that creates coverages for every coverage in the request message) or where the calling of a business object is conditional (e.g. we only want to call the direct debit create business object if direct debit data is present in the request message).

4.7. What to Avoid

- Converting programs which are badly structured - you may have to completely rewrite the program and then build the business object
- Including Work With style subfile submenus and emulating end-user behaviour on such screens. Typically a user selects one or more lines from such a submenu and processing is invoked for each selected line. It does not make sense to emulate this selection process in the business object. Instead, the request data should be used to determine which entities to act against and to invoke subsequent processing accordingly.
- Depending heavily on the subfile processing as present in the sections of the Pnnnn programs. SMART provides a facility for handling the Pnnnn subfile processing in business object mode using data queues. This has limitations however
 - Only the SADD, SCLR, SSTRT and SREAD functions are available.
 - Get Changed (SRNCH) cannot be implemented.
 - Large subfiles are likely to exceed the 32K linkage limit and will then require MORE processing to page the data back and forth, resulting in multiple conversations for one transaction (see next section for subfile processing hints).
 - The 1000- section only loads a single display page. This might require several conversations to complete the transaction (this can be handled by coding additional ROLU calls in the business object itself).
- Using default error messages such as 'Field Must Be Entered'; this will make no sense when it arrives on the front-end system, since the field cannot be highlighted as it would be on the host system. Meaningful error messages must be added to all key on-line programs.
- Code validation on the front-end system. Keeping tables in sync between two platforms can be a nightmare.

4.8. Notes on Subfile Processing

4.8.1. Retrieving Data from a Subfile Screen

Most subfile screens, when called up for the first time, only write enough subfile records to fill the first screen. Whilst this is fine for use on-line, it creates problems when the screen is called by a business object.

The business object normally performs section 1000 of the program. However, if there is more than one screen of data, the program called will only return the records on the first screen. This problem can be overcome by creating a new section within the business object that sets the SCRN-STATUZ to ROLU and performs section 2000 of the program. This new section should be performed until SCRN-SUBFILE-MORE = SPACES (assuming that such indicator is used correctly in the original program).

4.8.2. Updating a Subfile Screen

A requirement to update subfile records from a business object is that the front-end system must pass the details of all subfile records involved, even if the intention is to change just one of them. This simplifies processing, enabling us to simply overwrite the old records with the new ones. To do this, the front-end system has to perform a retrieve and temporarily store the subfile records in its memory to update them before passing them back.

The first thing we must check when updating is whether the LiFE program called builds the whole subfile when section 1000 is invoked or only builds one screen of data. In the latter case, proceed as explained in "Retrieving Data from a Subfile Screen".

The next stage is to actually write the changed records to the subfile. However, in many cases (especially where accounting calculations are involved) there are hidden subfile fields on the screen. These fields are generally used for comparisons and for balancing purposes. If we simply write the new data to the subfile, we will blank out these fields. In many cases this will lead to a balancing failure.

A safe way to do the update is to read each subfile record before moving the new data in the respective fields and performing a SUPD on it. A guideline is that you should be able to change no more and no less than what you can change on the screen when working on-line.

Another point to note is that sometimes we may want to reduce the number of subfile records attached to a transaction (e.g. retrieve 7 records but only 4 are written back). In this case, you must make sure that the excess records are blanked out.

4.9. Notes on Error Trapping/Handling

4.9.1. Retrieving Subfile Errors from a Business Object

In some programs, you may find that checks following a validation may overwrite the area of the subfile where errors are stored, whilst displaying the errors on the screen.

This is obviously not a problem when working on-line, however it can cause seemingly inexplicable crashes when working from a business object. To overcome this, it may be possible to store the SUBFILE-ERRORS area in working storage and restore it when all further checks are completed. **Please note that this may not always be applicable.**

4.9.2. Use of ERROR-INDICATORS OF VALIDATION-ERROR

If you are using these indicators outside the normal calls to sections of LiFE programs, you may need to initialise them to spaces every time you start the program.

4.9.3. Use of SVERR

Prior to the March 2000 release of SMART, the subroutine SVERR is used to report on validation errors occurring within the business object as well as on errors occurring in LiFE programs. The SVERRCPY copybook is included after the call to the 2000 section of a Pnnnn program and includes a call to this routine. The routine returns up to 100 error codes plus the descriptions.

Business objects generated using the March 2000 release of SMART no longer use this copybook. The error trapping will be performed by the business object itself. The error code and description, along with the screen and field in error will be returned. This not only makes debugging a business object easier, but also means that the front-end system can interpret the fieldnames and present more accurate error information to the user.

4.10. Notes on the use of business objects in a DIARY setting

Customers using CSC's DIARY product may wish to invoke existing functionality through the use of business objects. Please refer to the DIARY documentation for further information.

4.11. Data Considerations

During the design phase, the most important decision to be taken is the location of strategic data. A solution that involves data redundancy or duplication is probably not the most favourable one. The question of how to keep the data synchronised between platforms should be avoided by design that excludes the need for it.

The most common approach so far has been to split data as follows:

4.11.1. Front-end Database

- Client Prospect Details
- Marketing Details
- Telephony Statistics
- Work Flow
- Management Information systems
- Quotation Details
- Basic Portfolio Details.

4.11.2. Server Database

- Actual Client Details (contract owners, assured, interested parties, beneficiaries, etc.)
- All administrative details relating to contracts and clients
- Financial Transactions
- Portfolio Details.

In this scenario, a Client is only registered on the host system when a contract is issued and monies received. This avoids holding large numbers of dormant Clients on the server database, while retaining the information on the front-end database for marketing and MIS use.

A related issue - and usually complex one - is printing. If quotations are provided from the host system application, then all printing will take place via the LiFE or POLISY system. If quotations are done from the front-end system, the burden of printing is likely to be split between the two, with a resulting complication in retaining a correspondence history in one database. If the configuration includes a good work flow component, such as AWD, synchronisation of the

correspondence history can be generated via triggers (or maybe an ODBC update from the front-end to write a LETC on the host system directly).

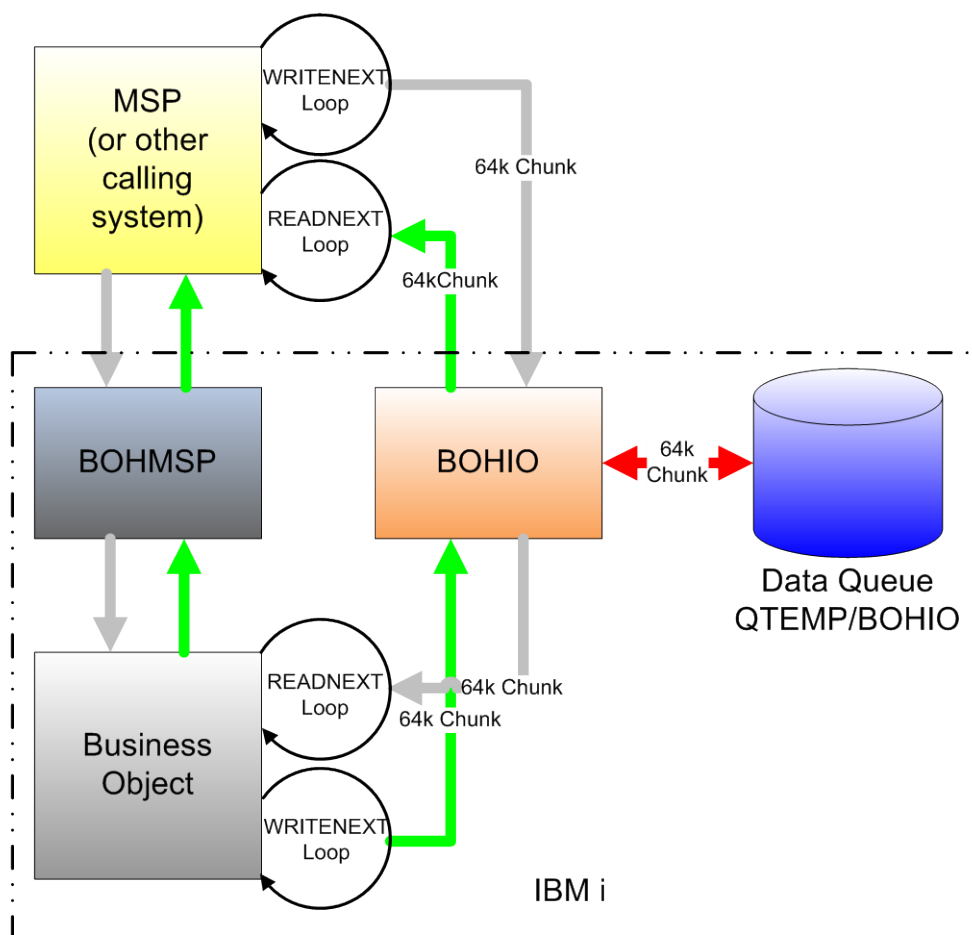
4.12. Processing Inbound Messages Over 32k Long

4.12.1. Overview

The Linkage fields for business object handlers currently have a maximum size of 32 000 (32K) characters each. This means that the data passed to or returned from a business object is limited to this amount in a single call.

Whilst this limit is large enough for the vast majority of business objects, there have been instances where this size restriction has hampered developments. A change has been made to allow for messages greater than 32k to be passed to business objects via Data Queues.

4.12.2. Implementation



In order to make use of this functionality, the invoking software must call BOHIO prior to the Business Object Handler call in order to store the request data that is required by the business object. The data is stored in 64k chunks on a data queue in QTEMP. For each 64k chunk of data a call to BOHIO must be made.

The business object is then called via the Business Object Handler program (i.e. BOHMSP).

The business object must then call BOHIO at the start of processing to retrieve the 64k request data chunks from the data queue. It must loop through all of the entries on the data queue to get the complete message.

Once the business object has completed its processing, it will call BOHIO again to store the response data.

Once control is returned to the invoker, the invoker must then call BOHIO again to retrieve the response data from the data queue that was put on there by the business object.

It is fundamental that the same session is used to call BOHIO as calls the BOH program to maintain the QTEMP connection.

Note that the Midrange Service Processor has been amended to make use of this facility. If either the request data or the response data for a business object call is greater in size than 32K, then BOHIO will be used to write the request data and read the response data..

4.12.3. BOHIO

4.12.3.1. Linkage

The linkage to the BOHIO module looks like this:

FUNCTION	X(10)
STATUZ	X(04)
DATA	X(64000)

The BOHIO subroutine provides the following functions:

WRITEFIRST

This function will check to see if the temporary data queue exists. If not, it will be created. If the queue already exists, it will be deleted first to clear out any remnants of previous calls. Once the queue is created, an entry is added to the data queue.

WRITENEXT

This function will add further entries to the data queue. The invoker should continue to use the WRITENEXT function until all request data has been passed to BOHIO.

READNEXT

This function should be used once control is returned to the invoker. The function will retrieve the response data. The first available entry is read from the queue and removed. Once the last entry has been read, an ENDP status is returned.

4.12.3.2. Example of BOHIO usage with a Business Object.

```

01  WSAA-FUNCTION                                PIC X(10) .
01  WSAA-STATUZ                                  PIC X(04) .
01  WSAA-QUEUE-ENTRY.
      03  WSAA-QUEUE-ENTRY-P1                    PIC X(32000) .
      03  WSAA-QUEUE-ENTRY-P2                    PIC X(32000) .
-----

BOHIO-PROCESSING SECTION.
BOHIO-START.

***** Read the Request Data put on the queue by the Invoker.

MOVE O-K                                TO WSAA-STATUZ.
MOVE 'READNEXT'                          TO WSAA-FUNCTION.
MOVE SPACES                              TO WSAA-QUEUE-DATA.

PERFORM UNTIL WSAA-STATUZ NOT = O-K

      CALL 'BOHIO'                        USING WSAA-FUNCTION
                                              WSAA-STATUZ
                                              WSAA-QUEUE-ENTRY

**** Process the WSAA-QUEUE-ENTRY or store in Working Storage
**** until all data has been retrieved
      END-PERFORM.

**** Perform relevant processing

**** Write the response data back to the Data Queue for the Invoker to pick up

MOVE O-K                                TO WSAA-STATUZ.
MOVE 'WRITEFIRST'                        TO WSAA-FUNCTION.

PERFORM UNTIL condition
      CALL 'BOHIO'                        USING WSAA-FUNCTION
                                              WSAA-STATUZ
                                              WSAA-QUEUE-ENTRY

      MOVE 'WRITENEXT'                    TO WSAA-FUNCTION
END-PERFORM.

BOHIO-EXIT.
EXIT.

```

4.13. Miscellaneous

4.13.1. WSSP-COMMON-AREA

MAINX initialises the WSSP-COMMON-AREA at the start of each program and restores it when execution is terminated. As the WSSP information is also stored in a separate area (as a backup), this can always be restored upon completion of a business object, even when a session update has taken place within the business object itself. The reason for this was that, previously, WSSP information was updated by LiFE programs, but never returned to its original form. Thus, when another unrelated business object was called, it would have used corrupted information causing unpredictable results or abending altogether. This problem was of particular relevance in cases where a business object calls another business object which, in turn, calls another business object.

5. Business Object Test Tool

5.1. BOTT Introduction

This section documents the functions relating to the use of the SMART Business Object Testing Tool ('BOTT') application. It is intended for the use of technical personnel involved in the manipulation and execution of Business Objects('BO's).

The following topics are covered within this guide:

- An overview of the BOTT solution.
- The on-line functionality of the BOTT solution.
- Integration with the Business Object Handler.
- Details about the Software Defaults and File Configuration required in order to execute the BOTT.

The majority of BOs are generated from existing screen definitions. Each BO can carry out one or more functions that are triggered by sending the BO a message.

These messages are constructed as a flat string of text, whose structure is defined by a COBOL copybook specific to each message. These messages could originate from a Graphical Front end.

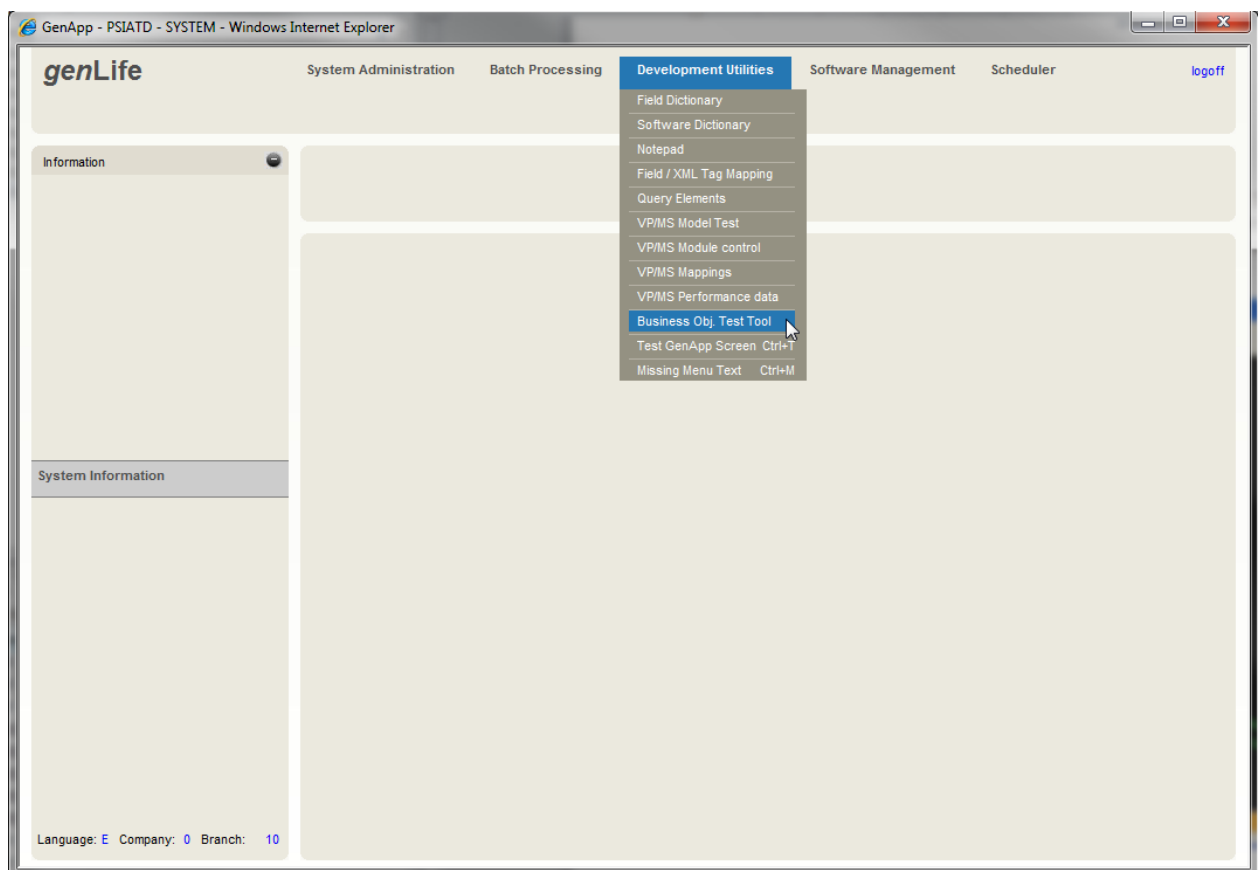
The BOTT provides the technical user with the ability to define, manipulate and analyse messages in a user friendly way. It will also provide the technical user the ability to execute the required BO and analyse the results. At each stage the results can be shown on the screen and/or emailed to the user.

The major advantage of this facility is that the Server part of the application can be tested easily and fully without any consideration for the client Front-End.

5.2. Overview

5.2.1. Online Subsystem

The BOTT solution consists of an online configuration subsystem. The online subsystem resides within System company under the Development Utilities master menu, however if testing non-SMART business objects the subsystem should be copied to each application company:



The purpose of BOTT is to store example test data against a particular BO and Method to verify that it works as expected, like a 'Benchmark'. Additional tests can be created to verify other test cases.

When the tests are run, the data is stored for reference. Both the input test data and results can be printed.

5.2.2. Business Object Test Data Admin Submenu

Business Object Test Data Admin - S1722

Library: PSIATDBOTT

File:

Input: INPDTA

Output: OUTDTA

Copybook: CPYDEF

Action: Work With Test Data Members ▼

F1=Help

OK Cancel Help

The submenu is used to capture the files and library that is to be used by the BOTT.

- The Input file is where the test data members are stored. The test data is stored in a flat file source structure as it is consistent with the previous BO test structure method.
- The Output file is where the BO result data is to be stored.
- The Copybook file contains the copybook definitions for the BOs and allows users to associate the data with the correct structure used at the time of execution. If the copybook subsequently changes we will be able to highlight the mismatch in structure.
- The Library is where all test files reside.

All Input fields are validated based on size and type. The program uses the default values (see software defaults in the SMART User Guide) for the environment to specify the correct initial entries although the user can select different libraries and files if required. A good example of such a need might be to check a test from another environment works or perhaps a developer requires a specific set of tests that they do not want deployed for general use.

5.2.3. Work with Business Object Test Data

Work-With Business Object Test Data - S1723

Input Filename: INPDTA Output Filename: OUTDTA

Library: PSIATDBOTT Copybook Filename: CPYDEF

>>> <<< Filter

Sel	Member	Object	Verb	Description
<input type="text"/>	ACCBALENQ	ACC	BALENQ	ACC BALENQ test
<input type="text"/>	ACCBALLST	ACC	BALLST	ACC BALLST test
<input type="text"/>	ACCBALLST2	ACC	BALLST	ACC BALLST test pauls test
<input type="text"/>	ACCMOVLST	ACC	MOVLST	ACC MOVLST test
<input type="text"/>	ACCTRNLST	ACC	TRNLST	ACC TRNLST
<input type="text"/>	AGSCRT	AGS	CRT	Agent Create
<input type="text"/>	AGSENG	AGS	ENQ	AGENT ENQUIRY
<input type="text"/>	AGSUPD	AGS	UPD	AGENT UPDATE
<input type="text"/>	AGTCRT	AGT	CRT	Agent create
<input type="text"/>	AGTENQ	AGT	ENQ	Agent Enquiry
<input type="text"/>	AGTEXISTS	AGT	EXISTS	Agent Exists
<input type="text"/>	AGTHRD	AGT	HRD	Agent Hierarchy Down

F1=Help F3=Exit F5=Refresh F6=Create F11=More Options F12=Cancel More...

OK Cancel Help

The “Work With Business Object Test Data” screen allows the User to:

- Create – Create a new BO Test Member.
- Modify – Amend an existing BO Test Member.
- Copy – Copy a BO Test Member to a new name or overwrite an existing test.
- Delete – Delete a BO Test Member.
- Enquire – View the BO Test Member. definition.
- Run – Run a BO Test Member test.
- View – The input and output information as well as providing an option to email the details.

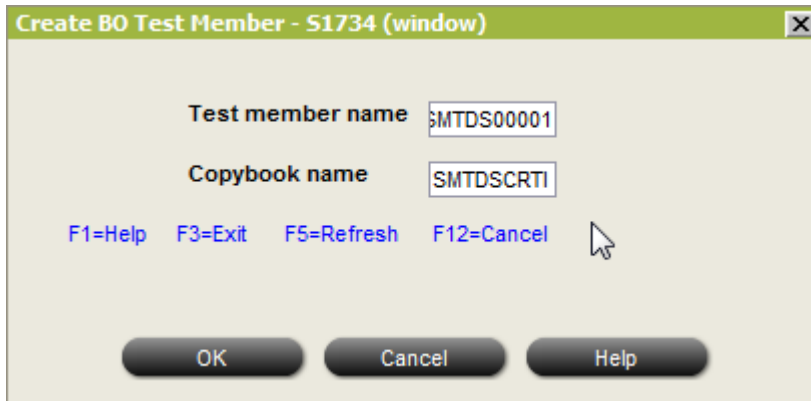
Filter functionality exists on the subfile, which allows the user to search by ‘Member’, ‘Object’ or ‘Verb’.

The selection field is validated to ensure that only the options shown for the user profile are visible.

The member field is highlighted if the copybook for the BO Test Member is out of date. Also, an asterisk is displayed on the subfile if the copybook for the BO Test Member does not exist.

5.2.4. Create a Business Object Test Member

To create a BO Test Member press <<F6>> on “Work With Business Object Test Data” screen:



The BO Test Member name can be anything, but it is suggested that an initial base version, which has the same name as the BO + Method, is used. For any subsequent tests it is recommended that a sequential number is added to the end of the BO Member name and incremented. The copybook field is validated to ensure that a copybook of that name actually exists.

The “Business Object Fields” screen is then displayed to allow the user to input the business test case:

Session B - [24 x 80]

File Edit View Communication Actions Window Help

NAQRD - SYSTEM Business Object Fields S1724 01

File. . : INPDTA Company: 0 Acc Year : 2011 Language: E
 Library : NAQRDBOTT Branch : 10 Acc Month: 6 User. . : IGAMMIE

Member. : SMTDS00001 Create a data set test 1
 Copybook: SMTDSCRTI Object: SMTDS Verb . . . : CRT

Field Name	Occ	Tp	Len	DP	Value
MSGID		A	10		
MSGLNG		Z	5		00000
MSGCNT		Z	5		00000
FILLER		A	10		
BGEN-S0218-FILN		A	6		BOX1PF
BGEN-S0616-WORKU		A	6		IGTEST
BGEN-S0219-AUTOSTAMP		A	1		Y
BGEN-S0219-COMDTA		A	1		N
BGEN-S0219-DSLGL		A	1		N
BGEN-S0219-FILD		A	40		BOTT test file create
BGEN-S0219-JRNL		A	1		N

More...

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

10/050

3902 - Session successfully started

Business Object Fields - S1724

File INPDTA Company 0 Acc Year 2013 Language E

Library PSIATDBOTT Branch 10 Acc Month 6 User IGAMME

Member SMTDS00001 Create a data set test 1

Copybook SMTDSCRTI Object SMTDS Verb CRT

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		
MSGLNG	N	5		00000
MSGCNT	N	5		00000
FILLER	A	10		
FILN	A	6		BOX1PF
WORKU	A	6		IGTEST
AUTOSTAMP	A	1		Y
COMDTA	A	1		N
DSLGL	A	1		N
FILD	A	40		BOTT test file create
JRNL	A	1		N

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

This screen is a subfile

Business Object Fields - S1724

File INPDTA Company 0 Acc Year 2013 Language E

Library PSIATDBOTT Branch 10 Acc Month 6 User IGAMMIE

Member SMTDS00001 Create a data set test 1

Copybook SMTDSCRTI Object SMTDS Verb CRT

Field Name [index]	Tp	Len	DP	Value
QRYF	A	1		N
SUBSYS	A	10		SDVBOTT
FDID[001]	A	10		COMPANY
FOCC[001]	N	2		00
FDID[002]	A	10		BRANCH
FOCC[002]	N	2		00
FDID[003]	A	10		LONGDESC
FOCC[003]	N	2		00
FDID[004]	A	10		
FOCC[004]	N	2		00
FDID[005]	A	10		

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

This screen allows the user to define the input parameters for a BO Test Member. The list of fields is generated from the copybook as well as the type and length. Arrays of a field are also shown with any nested levels limited at present to 5.

The data entered in the value fields is validated for size but not type as they are generic input fields.

Note: When first creating the test, all numerics are automatically initialised and dates are set to an initial value of 999999999, this is to reduce the risk of a fatal error with decimals.

5.2.5. Modify a Business Object Test Member

To find the test member you require you can put a partial description in any of the filter fields and the screen will display the elements that meet the filter criteria, for example;

Work-With Business Object Test Data - S1723

Input Filename: INPDTA Output Filename: OUTDTA

Library: PSIATDBOTT Copybook Filename: CPYDEF

>>> SMTDS <<< Filter

Sel	Member	Object	Verb	Description
<input type="text"/>	SMTDSCHG	SMTDS	CHG	SMART Data set change
<input type="text"/>	SMTDSCRT	SMTDS	CRT	Smart Data set create
<input type="text"/>	SMTDSCRT01	SMTDS	CRT	Smart Data set create
<input type="text"/>	SMTDS00001	SMTDS	CRT	Create a data set test 1

Modify
Copy
Delete
Enquire
Print Mbr
Run Test
View Outp
Prt Output
View CBook
Prnt CBook

F1=Help F3=Exit F5=Refresh F6=Create F11=More Options F12=Cancel

Filter applied - no matches

OK Cancel Help

To modify a BO Test Member select the modify option from the drop down select next to the BO Test Member in the S1723 Subfile.

Business Object Fields - S1724

File INPDTA Company 0 Acc Year 2013 Language E

Library PS1ATDBOTT Branch 10 Acc Month 6 User IGAMMIE

Member SMTDS00001 Create a data set test 1

Copybook SMTDSCRTI Object SMTDS Verb CRT

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		SMTDSCRTI
MSGLNG	N	5		12185
MSGCNT	N	5		00001
FILLER	A	10		
FILN	A	6		BOX1PF
WORKU	A	6		IGTEST
AUTOSTAMP	A	1		Y
COMDTA	A	1		N
DSLGL	A	1		N
FILD	A	40		BOTT test file create
JRNL	A	1		N

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

The “Business Object Fields” screen allows the user to change the initial values that were entered for the BO Test Member case. Any field value can be changed. Numeric fields must contain a numeric. Date fields must contain a date value in the form CCYYMMDD.

5.2.6. Copy a Business Object Test Member

To copy a BO Test Member select the copy option from the drop down list next to the BO Test Member in the S1723 Subfile.

Copy Test Data Member - S1725

Filename INPDTA

Library PSIATDBOTT

From: _____

Member SMTDS00001 Create a data set test 1

Copybook SMTDSCRTI Object SMTDS Verb CRT

To: _____

Member SMTDS00002 Create a data set test 2

Copybook SMTDSCRTI Object SMTDS Verb CRT

F1=Help F3=Exit F5=Refresh F12=Cancel

OK Cancel Help

The 'Copy Test Data Member' screen allows the user to copy the test details to a new BO Test Member. Validation exists to ensure that the '**To**' BO Test Member name is different to the '**From**' BO Test Member name. However, if the 'To' member already exists, the 'Copy Test Data Member' screen is redisplayed with a confirmation message as shown below:

Session A - [24 x 80]

File Edit View Communication Actions Window Help

NAQRD - SYSTEM Copy Test Data Member S1725 01

Filename.: INPDTA
Library.: NAQRDBOTT

From: _____

Member: SMTDS00001 Create a data set test 1
Copybook: SMTDSCRTI Object: SMTDS Verb: CRT

To: _____

Member: SMTDS00002 Create a data set test 2
Copybook: SMTDSCRTI Object: SMTDS Verb: CRT

Confirm Overwrite: N

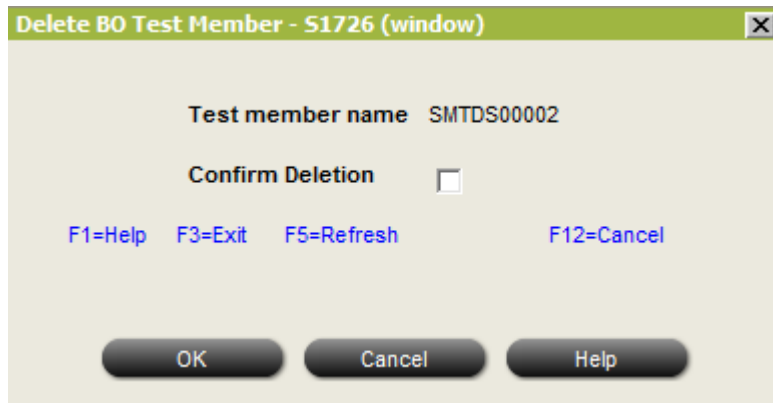
F1=Help F3=Exit F5=Refresh F12=Cancel
Overwrite existing member?

15/050
2902 - Session successfully started

The user must set the 'Confirm Overwrite' field to 'Y' to create a new copy of the test case. If the user does not wish to continue they must select an exit function key such as F3 or F12 to return to the submenu or list of test cases.

5.2.7.Delete a Business Object Test Member

To delete a BO Test Member select the Delete option from the drop down list next to the BO Test Member in the S1723 Subfile.



The 'Delete BO Test Member' screen allows the user to delete selected BO Test Member details. Deleting a BO Test Member removes its definition and test cases from the Request and Response files.

5.2.8. Enquire on a Business Object Test Member

To enquire on a BO test Member select the Enquire option from the drop down list next to the BO Test Member in the S1723 Subfile.

Business Object Fields - S1724

File INPDTA Company System Co Acc Year 2013 Language English

Library PSATDBOTT Branch Southern Acc Month 6 User IGAMMIE

Member SMTDS00001

Copybook SMTDSCRTI Object SMTDS Verb CRT

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		<input type="text" value="SMTDSCRTI"/>
MSGLNG	N	5		<input type="text" value="12185"/>
MSGCNT	N	5		<input type="text" value="00001"/>
FILLER	A	10		<input type="text"/>
FILN	A	6		<input type="text" value="BOX1PF"/>
WORKU	A	6		<input type="text" value="IGTEST"/>
AUTOSTAMP	A	1		<input type="text" value="Y"/>
COMDTA	A	1		<input type="text" value="N"/>
DSLGL	A	1		<input type="text" value="N"/>
FILD	A	40		<input type="text" value="BOTT test file create"/>
JRNL	A	1		<input type="text" value="N"/>

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

The "Business Object Fields" screen displays the BO Member test definition and the latest field values. No field values can be amended while in enquiry mode as the screen is protected.

5.2.9. Print a Business Object Test Member

To print a BO Test Member select 'Print Mbr' option from the drop down list next to the BO Test Member in the S1723 Subfile.

The 'Spool File Options' screen allows a user to view the member online and/or send the details, as a PDF, to an email address. If an email address has been entered then the 'Send as PDF file' indicator must be set.

Note: In order to send a PDF an SMTP server must be defined on table T1721 with a valid email address must be entered. The PRTSPLF command must be available in the installed version of SMART, if it is not please contact the CSC support team.

If the 'Show On Screen' flag is set to 'Y', the following screen is shown:

The following is an example of the generated PDF file sent via email:

psispoolfilespdfR1727.PDF - Adobe Reader

File Edit View Window Help

1 / 1 95.5%

Comment

Page Thumbnails: Go to specific pages using thumbnail images

BUSINESS OBJECT TEST DATA PRINT

Branch: 10 Member: SMTEERRCRT Error Code Create File: PSIAQDBOTT/INPDIA Run Date: 31/08/2012 Time: 9:58:16

Message Definition: Copybook: SMTEERRCRTI Object: SMTERR Verb: CRT

Fieldname	Occ	Type	Len	Dec	Value	Window	Element description
01 SMTEERRCRTI-REC							
02 MESSAGE-HEADER							
03 MSGID	A		10		SMTEERRCRTI		
03 MSGLNG	Z		5		00175		
03 MSGCNT	Z		5		00001		
03 FILLER	A		10				
02 MESSAGE-DATA							
03 BGEN-S0052							
04 BGEN-S0052-EROREROR	A		4		X999		ERROR NUMBER
04 BGEN-S0052-ERORPROG	A		10				PROGRAM
04 BGEN-S0052-LANG01	A		1		E	0 T1680	LANGUAGE
03 BGEN-S0616							
04 BGEN-S0616-WORKU	A		6		IGTEST		Work Unit
03 BGEN-S0053							
04 BGEN-S0053-ERORDESC	A		24		This is a test error		ERROR DESCRIPTION
***** E N D O F R E P O R T *****							

5.2.10. Run a Business Object Member Test

To run a BO Test Member test select the 'Run Test' option from the drop down list next to the BO Test Member in the S1723 Subfile. This will take the test data, build the call to the BO and then call the underlying programs to call the B/O.

A successful call returns a message similar to the following response message:

Business Object Fields - S1724

File	OUTDTA	Company	System Co	Acc Year	2013	Language	English
Library	PSIATDBOTT	Branch	Southern	Acc Month	6	User	IGAMMIE
Member	SMTDS00001 <input type="text" value="Create a data set test 1"/>						
Copybook	SMTDSCRT0	Object	SMTDS		Verb	CRT	

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		<input type="text" value="SMTDSCRT0"/>
MSGLNG	N	5		<input type="text" value="00001"/>
MSGCNT	N	5		<input type="text" value="00001"/>
FILLER	A	10		<input type="text"/>
FILLER	A	1		<input type="text"/>

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

In this instance the test was successful as the data returned had no errors.

Note: In this example the BO has no data to return as it is creating a dataset.

If there was an error in the data such as an incorrect subsystem name, the returning data would look like this:

Business Object Fields - S1724

File OUTDTA Company System Co Acc Year 2013 Language English

Library PSIATDBOTT Branch Southern Acc Month 6 User IGAMMIE

Member SMTDS00001

Copybook BOVERRREC Object SMTDS Verb CRT

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		BOVERRREC
MSGLNG	N	5		02124
MSGCNT	N	5		00001
FILLER	A	10		
BOVERR_LANGUAGE	A	1		E
BOVERR_PROG	A	10		P0219
BOVERR_ENTID	A	8		
BOVERR_EXTRA	A	30		
BOVERR_BOFOCCUR[001]	N	5		00000
BOVERR_FIELD[001]	A	50		S0219-SUBSYS
BOVERR_EROR[001]	A	4		E294

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

The error is returned in the BO Error copybook (BOVERRCPY) format. It shows that the error code is 'E294' and it was issued against the S0219-SUBSYSTEM in the program P0219. E294 is an invalid subsystem name.

5.2.11. View Business Object Test Output

To view the BO Test Member output, select the 'View Outp' option from the drop down list next to the BO Test Member in the S1723 Subfile.

Business Object Fields - S1724

File	OUTDTA	Company	System Co	Acc Year	2013	Language	English
Library	PSIATDBOTT	Branch	Southern	Acc Month	6	User	IGAMMIE
Member	SMTDS00001 <input type="text" value="Create a data set test 1"/>						
Copybook	SMTDSCRTO	Object	<input type="text" value="SMTDS"/>		Verb	<input type="text" value="CRT"/>	

Field Name [index]	Tp	Len	DP	Value
MSGID	A	10		<input type="text" value="SMTDSCRTO"/>
MSGLNG	N	5		<input type="text" value="00001"/>
MSGCNT	N	5		<input type="text" value="00001"/>
FILLER	A	10		<input type="text"/>
FILLER	A	1		<input type="text"/>

F1=Help F3=Exit F4=Prompt F5=Refresh F12=Cancel

OK Cancel Help

The screen displays the BO Test Member response data produced from running the business object test case, the data returned is in the format of the output copybook for the business object, for example SMTDSCRTO.

If errors occur they get displayed in the SFERRREC copybook format.

In this case SMTDSCRTO has been successfully returned by the BO.

5.2.12. Print Business Object Test Output

To print the output from a BO Test Member select 'Prt Output' from the drop down list next to the BO Test Member in the S1723 Subfile.

The 'Spool File Options' screen allows a user to View the output Online or to send the details as a PDF using a defined email address. Also, if an email address has been entered then the send indicators must be set.

The 'Spool File Options' screen allows a user to view the member online and/or send the details, as a PDF, to an email address. If an email address has been entered then the 'Send as PDF file' indicator must be set.

Note: In order to send a PDF an SMTP server must be defined on table T1721 and a valid email address must be entered. The PRTSPLF command must be available in the installed version of SMART, if it is not please contact the CSC support team.

If the 'Show on Screen' flag is set then the following screen is shown:

If the option to 'Send as PDF' is selected and a valid email applied, then the following is an example of the PDF file sent:

~5636353.PDF - Adobe Reader

File Edit View Window Help

1 / 1 95.5%

Comment

R1727
Company: 0
Branch: 10
Member: SMTDSCRT
Smart Data set create
File: PSIAQDBOTT/OUTDTA
Run Date: 31/08/2012
Time: 12:03:45

Message Definition: Copybook: SMTDSCRT Object: SMTDS Verb: CRT

Fieldname	Occ	Type	Len	Dec	Value	Window	Element description
01 SMTDSCRT-REC							
02 MESSAGE-HEADER							
03 MSGID	A		10		SMTDSCRT		
03 MSGLNG	Z		5		00001		
03 MSGCNT	Z		5		00001		
03 FILLER	A		10				
02 MESSAGE-DATA							
03 FILLER	A		1				

***** END OF REPORT *****

Note that in this example, when creating a data set there is no return data. In the following example we have run a software enquiry, and the return data in the PDF is far greater.

~7479512.PDF - Adobe Reader

File Edit View Window Help

1 / 10 95.5%

Comment

R1727
Company: 0
Branch: 10
Member: SMTSEEU
Software Enquiry - Elements Us
File: PSIAQDBOTT/OUTDTA
Run Date: 31/08/2012
Time: 12:08:42

Message Definition: Copybook: SMTSEEU Object: SMTSE Verb: EU

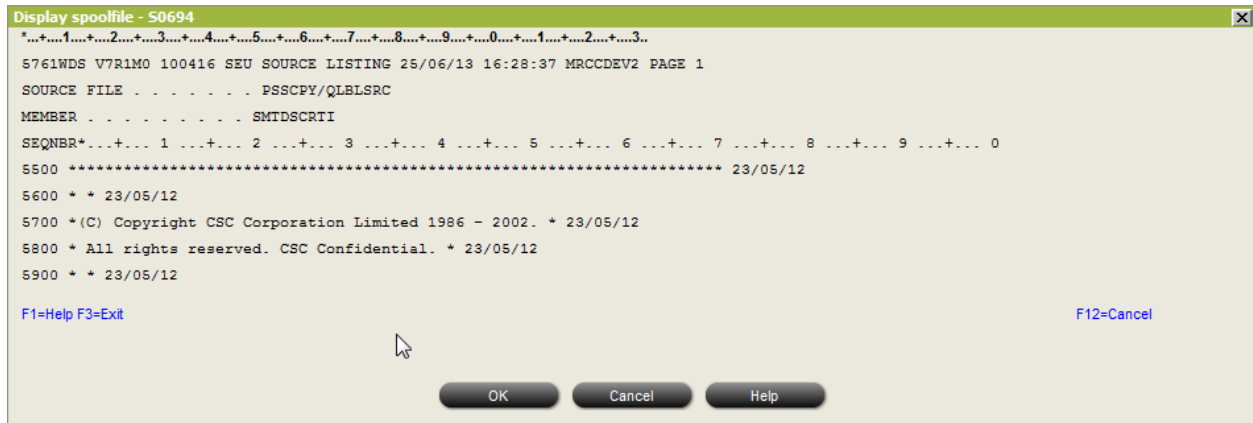
Fieldname	Occ	Type	Len	Dec	Value	Window	Element description
01 SMTSEEU-REC							
02 MESSAGE-HEADER							
03 MSGID	A		10		SMTSEEU		
03 MSGLNG	Z		5		15400		
03 MSGCNT	Z		5		00001		
03 FILLER	A		10				
02 MESSAGE-DATA							
03 BGEN-S0502							
04 BGEN-S0502-SFL	1	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Batch control file		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		BATC		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		LGL		
04 BGEN-S0502-SFL	2	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Create a Business Obj test dat		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		CRTBOFFCL		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		CLP		
04 BGEN-S0502-SFL	3	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Linkage to CRTBOFFCL subroutin		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		CRTBOFFREC		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		CPY		
04 BGEN-S0502-SFL	4	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Converts dates - external to i		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		DATCON1		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		CBL		
04 BGEN-S0502-SFL	5	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Convert dates - external to in		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		DATCON1REC		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		CPY		
04 BGEN-S0502-SFL	6	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Invalid Action		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		E005		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		ERR		
04 BGEN-S0502-SFL	7	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		An invalid action number has		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		E005		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		EHLE		
04 BGEN-S0502-SFL	8	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		Invalid date		Uses Element Name
05 BGEN-S0502-USEELEMENT		A	50		E032		USES OBJECT TYPE
05 BGEN-S0502-USETYPE		A	4		ERR		
04 BGEN-S0502-SFL	9	A	50				Container
05 BGEN-S0502-HCONT		A	50				Description
05 BGEN-S0502-SFCDSCR		A	50		An invalid date has been enter		

CSC/ESG MRCCDEV2

5.2.13. View the Business Object Member Copybook

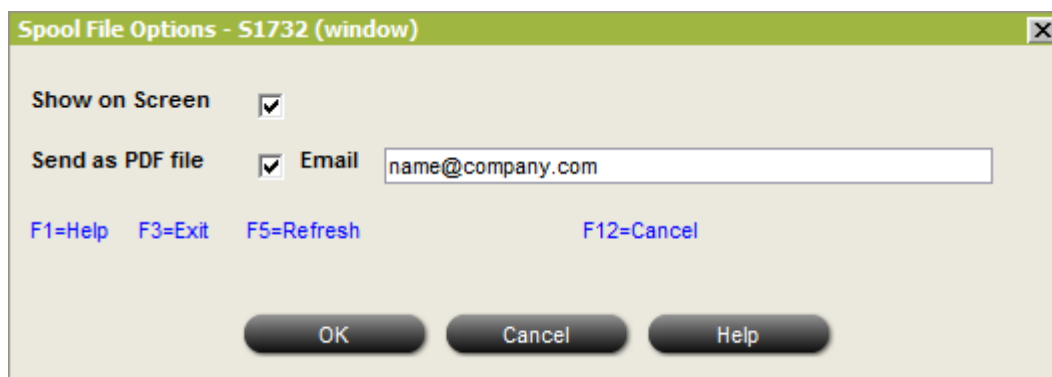
To view an input copybook for a BO Member:

The input copybook is generated as part of the BO creation process and defines the message structure for the request call to the B/O.



5.2.14. Print Business Object Member Copybook

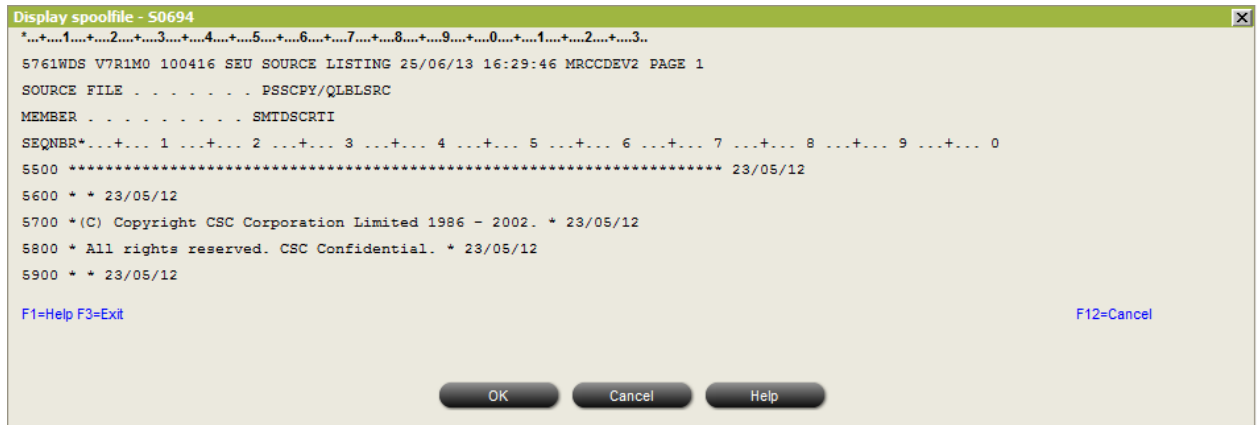
To print a BO Member copybook select 'Prnt CBook' option from drop down list next to the BO Test Member in the S1723 Subfile.



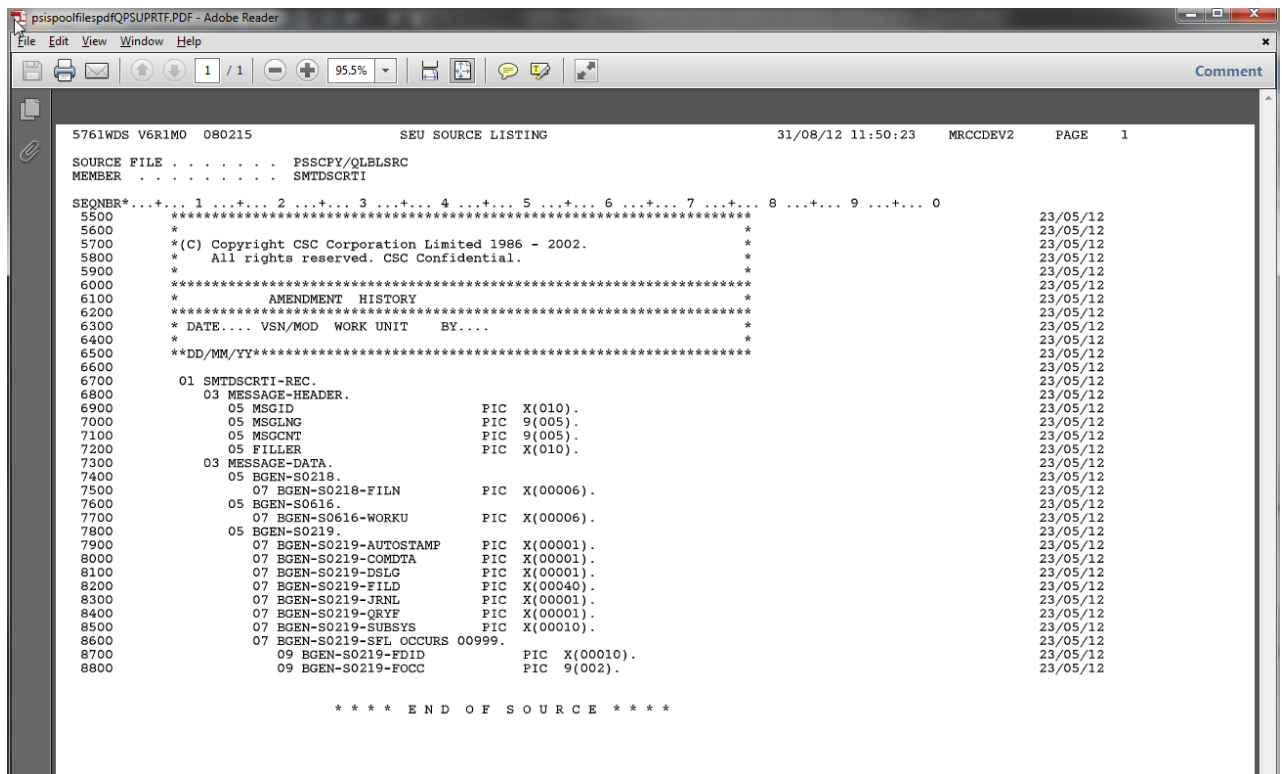
The 'Spool File Options' screen allows a user to view a BO Member online and/or send the details, as a PDF, to an email address. If an email address has been entered then the 'Send as PDF file' indicator must be set.

Note: In order to send a PDF an SMTP server must be defined on table T1721 and a valid email address must be entered. The PRTSPLF command must be available in the installed version of SMART, if it is not please contact the CSC support team.

If the 'Show on Screen' flag is set then the following screen is shown :



The following is an example of the PDF file sent to the requested email address if applicable:



5.2.15. Print the Business Object Member Test Data

To view this option, press <<F11>> to display the additional submenu options.

To print the BO Member Test data select 'Prt Input' option from drop down list.

Work-With Business Object Test Data - S1723

Input Filename: INPDTA Output Filename: OUTDTA
 Library: PSIATDBOTT Copybook Filename: CPYDEF

>>> SMTDS <<< Filter

Sel	Member	Object	Verb	Description
	SMTDSCHG	SMTDS	CHG	SMART Data set change
	SMTDSCRT	SMTDS	CRT	Smart Data set create
	SMTDSCRT01	SMTDS	CRT	Smart Data set create
	SMTDS00001	SMTDS	CRT	Create a data set test 1
Prt Input				

F1=Help F3=Exit F5=Refresh F6=Create F11=More Options F12=Cancel

OK Cancel Help

Spool File Options - S1732 (window)

Show on Screen ☒

Send as PDF file ☒ Email

F1=Help F3=Exit F5=Refresh F12=Cancel

OK Cancel Help

The 'Spool File Options' screen allows a user to view a BO Member online and/or send the details, as a PDF, to an email address. If an email address has been entered then the 'Send as PDF file' indicator must be set.

Note: In order to send a PDF an SMTP server must be defined on table T1721 and a valid email address must be entered. The PRTSPLF command must be available in the installed version of SMART, if it is not please contact the CSC support team.

If the 'Show on Screen' flag is set then the following screen is shown:

This option shows the copybook definition as well as each field's type, length and input data entered for it.

The screenshot shows a window titled "Display spoolfile - 50694". The content is as follows:

```
*...1...2...3...4...5...6...7...8...9...0...1...2...3...
R1727 BUSINESS OBJECT TEST DATA PRINT
Company: 0 Run Date: 25/06/2013
Branch: 10 Member: SMTDS00001 Create a data set test 1 File: PSIAATBOTT/INPDAT Time: 16:36:38
Message Definition - Copybook: SMTDSCRTI Object: SMIDS Verb: CRTI
Fieldname [Index] Type Len Dec Value Window Element description
05 MSGID A 10 SMTDSCRTI
05 MSGLNG N 5 12185
05 MSGCNI N 5 00001
05 FILLER A 10
```

At the bottom left, it says "F1=Help F3=Exit". At the bottom right, it says "F12=Cancel". At the very bottom, there are three buttons: "OK", "Cancel", and "Help".

5.3. Configuration

5.3.1. Software Defaults

The BOTT utilises software defaults to define The following Software Defaults relate to this subsystem.

Default	Description	Normal Value
BOTT_CPYF	Defines the file in the BOTT_LIB where copybooks are stored when BO Test Members are created and changed. This allows the detection of changes in Message structure.	CPYDEF
BOTT_INPF	Defines the file in the BOTT_LIB where the Input BO Test Members are stored.	INPDTA
BOTT_LIB	Defines the suffix for the Business Object Test Data Library. This is appended to the Level name.	BOTT
BOTT_OUTF	Defines the file in the BOTT_LIB where the Output BO Test Members are stored.	OUTDTA

5.3.2. File Definitions

The files and library named at the S1722 Submenu must already exist in order to use the tool. The way to create these files is defined below and also in the Screen Help.

File Type	Description	Command
Copybooks	Requires a source file with the default record length of 92.	CRTSRCPF FILE(library name/filename)
Input Data	Requires a source file with a record length of 112.	CRTSRCPF FILE(library name/filename) RCDLEN(112)
Output Data	Requires a source file with the default record length of 112.	CRTSRCPF FILE(library name/filename) RCDLEN(112)

6. Appendix 1 - BOHMQMON

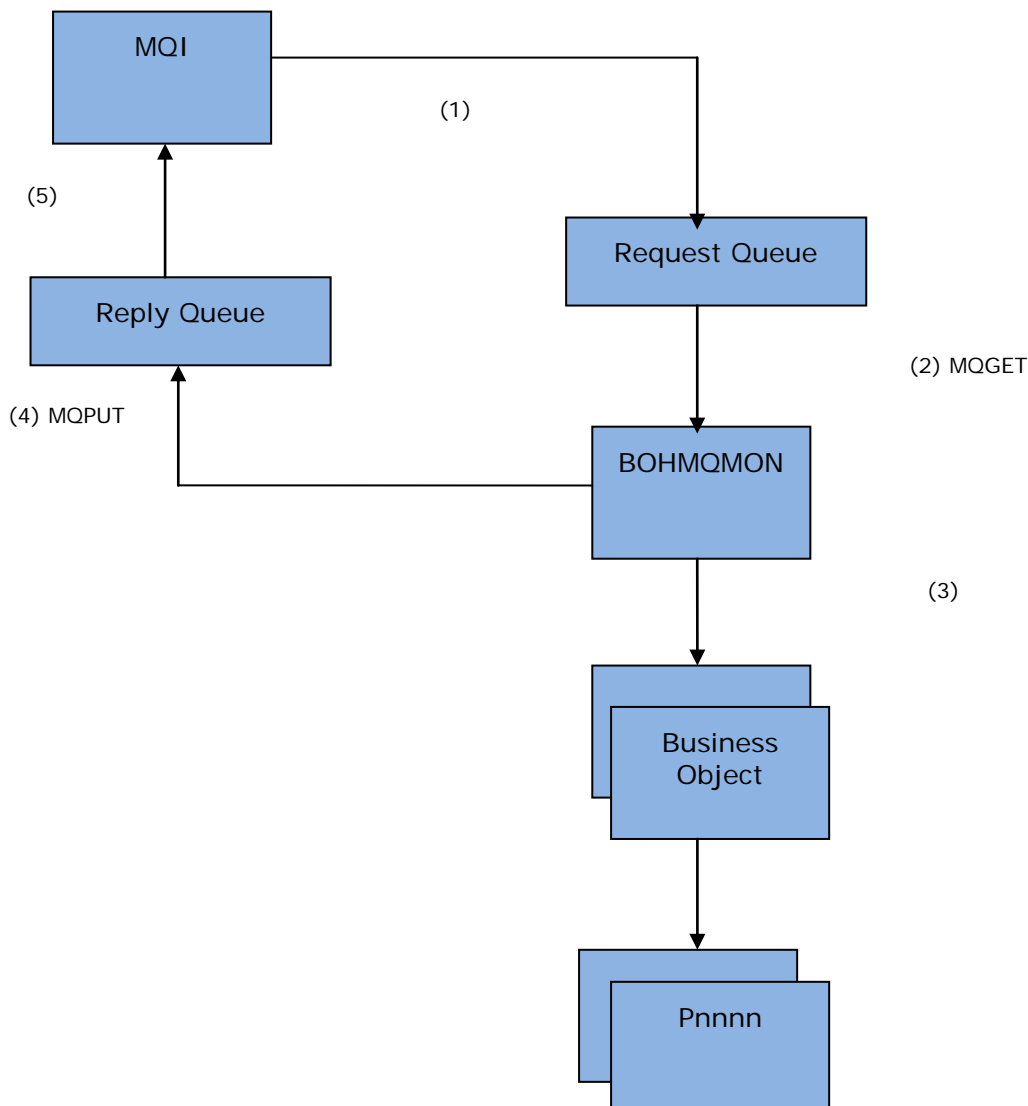
The BOHMQMON is an example FSU program that can be used as a starting point when building a solution that uses Business Objects in combination with IBM WebSphere MQ software for IBM i.

The appendix will explain how BOHMQMON handles the exchange of messages between the WebSphereMQ software and SMART business objects. It will also highlight the WebSphere MQ facilities that are used in the example program.

6.1. Exchange of Messages

This section provides an overview of the way WebSphere MQ and Business Objects communicate with each other by means of exchanging messages. First, a diagram is presented detailing the messages life cycle. This diagram is then explained in detail, followed by some technical information on how the example Business Object Handler for WebSphere MQ functions.

6.1.1. Messages Life Cycle



6.1.2.Explanation

MQI = Message Queue Interface

(1)

The outside world passes a message via the MQI to a request queue on the IBM i machine.

(2)

The program BOHMQMON is an example queue monitor program written by CSC.

To activate it, a job must be submitted to call the program and the only parameter passed to this program is the name of the queue to monitor.

As soon as a message arrives on the queue, the monitor will process it. The BOHMQMON program uses the MQGET function to get the business object request data from the request queue. If there are no more messages left on the queue to process, the monitor program will remain active by waiting for a de-queue operation.

(3)

Once a message is received via the MQGET function, the BOHMQMON program calls the generic SMART Business Object Handler to execute the requested business object. The business object in turn calls the existing on-line Pnnnn programs. Upon completion of the business object, control is returned to BOHMQMON.

(4)

Now the BOHMQMON program performs an MQPUT to write a message to the reply queue to return the business object response data.

(5)

The MQI ensures that the reply message is delivered back to the outside world.

6.1.3. BOHMQMON in Detail

6.1.3.1. Asynchronous Processing

WebSphere MQ works asynchronously – therefore, at the time of sending a message, there is no guarantee that the receiving system is active and/or is able to respond immediately. As WebSphere MQ works on the basis of assured delivery, the message will always be delivered, but there might be a time lag before processing takes place.

This aspect of asynchronous communication has an effect on the way the business object handler deals with information passed to a business object. An outline below of the way a SMART “session” works helps to understand this effect.

To use business objects successfully, the system needs to emulate what the DRIVER command does in the on-line LiFE or POLISY system. In the business objects solution, there is a special SMART business object - called SESSION - that will perform the equivalent actions of the DRIVER command. This business object will perform certain checks like environment locks, user validity, company and branch authorities etc. It will also set up the WSSP values that are used throughout the Pnnnn programs.

To start a business object conversation, this SESSION business object needs to be executed prior to any other processing. In a synchronous, real-time connection, this needs to be done only once and all other business object calls in the same conversation can re-use the information set up by the SESSION business object.

The WebSphere MQ solution is a case of asynchronous communication, so the information dealt with in one message is not available for use in any subsequent messages.

This means that the SESSION information cannot be re-used and will have to be passed with each message. For each message, it then also means that the SESSION business object needs to be executed as well.

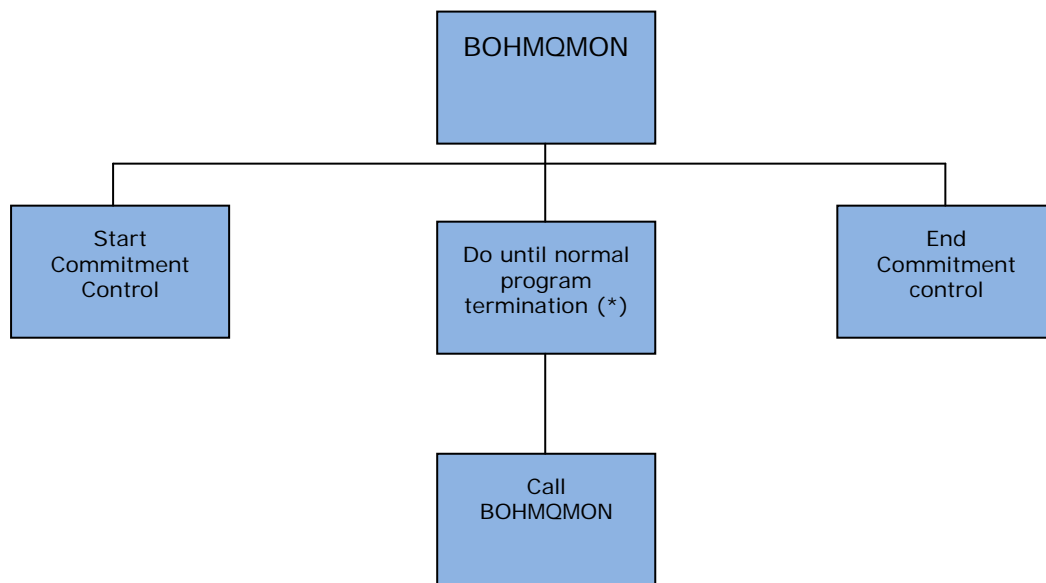
To explain it in very simple terms: - For synchronous communication methods, the “door stays open” throughout the conversation. For the WebSphere MQ implementation, however, the “door is opened and closed” per message.

6.1.3.2. BOHMQMON Program Suite

The business object handler for WebSphereMQ consists of two programs, BOHMQMONCL and BOHMQMON. There are two programs, as it is easier to execute commitment control actions and system error trapping from within a separate CL program.

BOHMQMONCL

The program does the following:

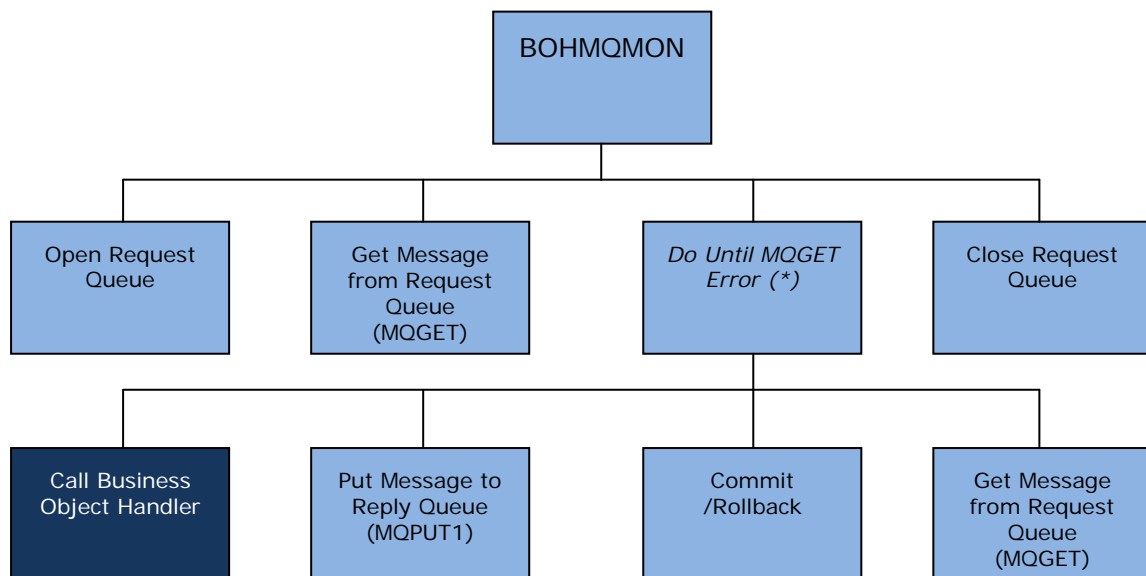


(*)

As BOHMQMON is a COBOL program, this program will end when a IBM i system error occurs (e.g. a CPFnnnn message is issued by the system). BOHMQMONCL traps all CPFnnnn, MCHnnnn, LBEnnnn and CBEnnnn messages and re-issues a call to BOHMQMON to ensure that the monitor job keeps active. Only when BOHMQMON terminates normally, BOHMQMONCL ends.

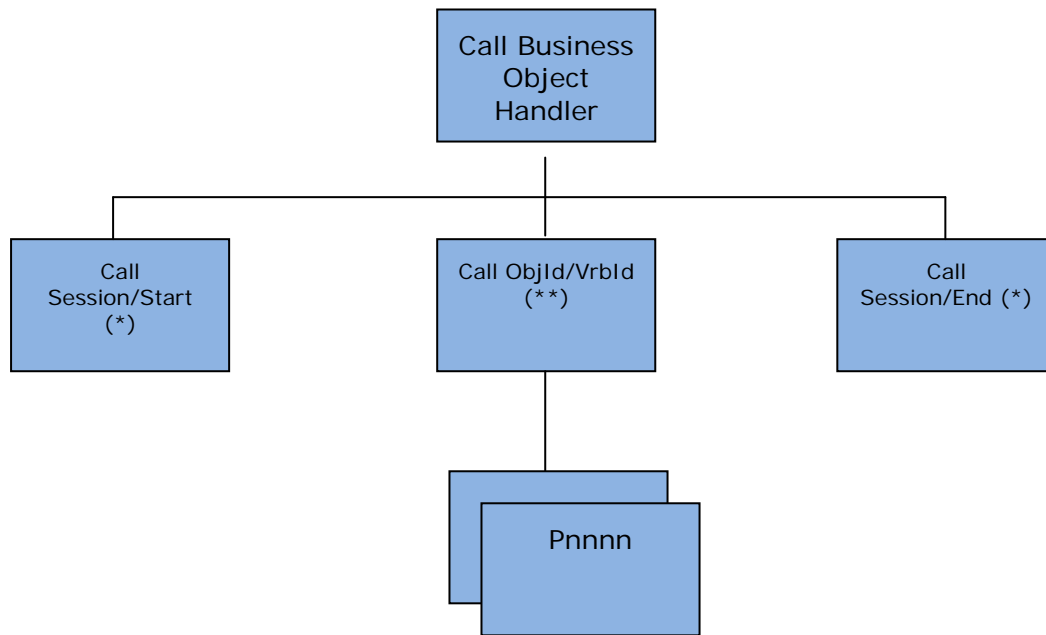
BOHMQMON

The program does the following:



(*)

MQGET errors such as when WebSphereMQ is quiescent or the queue is GET-disabled to stop the monitor job.



(*)

Before calling Session.Start, BOHMQMON checks the Session parameters. If these are the same as the previous call, the SESSION call will not be executed.

(**)

In case the business object traps an error, a rollback will be executed upon completion.

6.2. MQseries Facilities Used

The BOHMQMON program uses several WebSphere MQ facilities. Amongst these are the following:

- Reply-to-queue mechanism
- MQPUT1 versus MQPUT
- Request/Reply message correlation.

6.2.1. Reply-to-queue Mechanism

The MQ queue monitor program does not contain any hardcoded queue names to provide as much flexibility as possible. The queue that will be monitored for messages is passed in as a parameter and, for the output queue, the reply-to-queue mechanism is used.

This mechanism works on the basis that the reply queue name (and the reply queue manager name) is set up in the request message. These reply-to-queue parameters, as stored in the request message (and retrieved by the MQGET call), are used when writing the reply message using a MQPUT1 call (see next section for more information on this API).

This mechanism allows the most flexible approach with regards to queue names, as BOHMQMON does not need to know any queue name.

6.2.2. MQPUT1 versus MQPUT

To allow the reply-to-queue mechanism to work, the BOHMQMON program uses the MQPUT1 API. This API executes the elementary APIs MQOPEN, MQPUT and MQCLOSE all in one.

The only "problem" with this approach might be performance, as opening and closing the reply queue is less efficient than opening a reply queue once at the start of the monitor job and closing it when the monitor job ends.

If the number of queues used is not too large and the queue names are known, it is an option to change the BOHMQMON program to open all the queues at start-up and re-use the open connections as long as the monitor job is active. In this case, the MQPUT API can be used instead of MQPUT1.

6.2.3. Request/Reply Message Correlation

To enable the front-end application to pick up the correct reply message, WebsphereMQ provides an option to correlate request and reply messages.

Each message on a queue has a unique message id. As part of the reply message, the message id of the request message is returned. This so-called correlation id can then be used by the front-end application to read the reply message correlating to the request message.

6.3. Other Issues

6.3.1. Commitment Control

Commitment control is an issue that needs to be looked at while designing the transaction from front-end system to LiFE and back again.

There are several issues to consider, like what to do if the message is processed and the business object fails, or what if the business object ends successfully but the writing of the reply message fails?

The BOHMQMON program only looks at the IBM i side of the MQ transaction and uses two very simple principles:

- Once a message has been received via the MQGET statement, this message is removed from the request queue.

This means that when the business object fails with a system error or the MQPUT of the reply message fails, the MQGET is not rolled back.

The idea behind this is that the front-end will detect that no reply is returned (after a pre-defined time interval) and will then re-send the data if needed or take another predefined action. By removing the MQGET, we prevent situations where the MQGET keeps processing the same message and fails continuously.

- If the MQPUT fails, the business object updates are rolled back.

This again is to prevent processing from being performed twice. If the MQPUT fails, the front-end will time out and re-send the information. By rolling back the business object updates, we will prevent a situation where a second request message is sent which will then fail as all data is already there.

- This page is intentionally left blank -

7. Appendix 2 – ONLBOH and BCHBOH

The following business object handlers are generic subroutines, complementing the actions taken by the SMART online and batch 'driver' functionality.

Different versions are required as the SMART online and batch driver functionality performs different actions when processing a transaction.

7.1. Business object handler for online

ONLBOH – Online Business Object Handler (Used within Online programs)

- Ensures Commitment Control is handled by the standard DRIVER functionality
- Ensures Roll back processing is handled by the standard DRIVER functionality
- ATDIRECT Handling – If an AT program is called from a business object it forces a direct call instead of submitting the request
- Calls the required Business Object
- Error Handling – dealing with Fatal and validation errors and writing ELOG records where appropriate.

Linkage to the program consists of

- STATUZ field (alphanumeric 4)
- LEADER-HEADER
- REQUEST-DATA
- RESPONSE-DATA

7.2. Business object handler for batch

BCHBOH – Batch Business Object Handler (Used within Batch programs)

- Ensures Commitment Control is handled by the standard SMART batch driver functionality
- Ensures Roll back processing is handled by the standard SMART batch driver functionality
- Creates generic 'business object' linkage (i.e. WSSP record)
- Checks the Accounting period
- ATDIRECT Handling – If an AT program is called from a business object it forces a direct call instead of submitting the request
- Calls Business Object
- Error Handling – dealing with Fatal and validation errors and writing ELOG records where appropriate
- Allows the WSSP-COMPANY and WSSP-BRANCH values used in business object to be passed in linkage.

Linkage to the program consists of

- STATUZ field (alphanumeric 4)
- BSSCREC (as in batch program linkage LSAA-BSSCREC)
- BSPRREC (as in batch program linkage LSAA-BSPRREC)
- LEADER-HEADER
- REQUEST-DATA
- RESPONSE-DATA
- COMPANY
- BRANCH.

7.3. Typical usage of ONLBOH and BCHBOH

Fill the input fields for the business object call

MOVE values TO fields OF xxxyyyyl-REC.

Set default information (where xxx is business object and yyy is method)

```
MOVE LENGTH OF MESSAGE-DATA OF xxxyyyyl-REC
                                TO MSGLNG OF xxxyyyyl-REC.
MOVE 1                          TO MSGCNT OF xxxyyyyl-REC.
MOVE xxxyyyyl-REC              TO REQUEST-DATA.
MOVE LENGTH OF MESSAGE-DATA OF xxxyyyyO-REC
                                TO MSGLNG OF xxxyyyyO-REC.
MOVE 1                          TO MSGCNT OF xxxyyyyO-REC.

MOVE 'xxx'                     TO OBJID.
MOVE 'yyy'                     TO VRBID.
```

For online the following call can be made

```
CALL 'ONLBOH'                  USING WSAA-STATUZ
                                LEADER-HEADER
                                REQUEST-DATA
                                RESPONSE-DATA.
```

For batch the following call can be made

```
CALL 'BCHBOH'      USING WSAA-STATUZ
                     LSAA-BSSCREC
                     LSAA-BSPRREC
                     LEADER-HEADER
                     REQUEST-DATA
                     RESPONSE-DATA
                     LSAA-COMPANY
                     LSAA-BRANCH.
```

```
IF NOT NO-ERROR OF LEADER-HEADER
  PERFORM BO-ERROR
ELSE
  MOVE RESPONSE-DATA TO xxxyyyO-REC
END-IF
```