RESOURCE AND PATIENT MANAGEMENT SYSTEM

Electronic Health Record

(EHR)

Technical Manual Volume 4

Version 1.1
April 2020

Office of Information Technology
Division of Information Technology

Table of Contents

78.0 Super-Bills ... 1
78.1 Introduction ... 1
78.2 Architecture and Business Process Overview ... 2
78.3 Implementation and Maintenance ... 3
78.4 Routine Descriptions ... 3
78.5 File List ... 3
78.5.1 BGO CPT Preferences (# 90362.31) ... 3
78.6 Cross References ... 5
78.7 Exported Options ... 6
78.8 Exported Security Keys ... 6
78.9 Exported Protocols ... 6
78.10 Exported Parameters ... 6
78.11 Exported Mail Groups ... 6
78.12 Callable Routines ... 6
78.12.1 RPC: BGOCPTP3 STORE ... 6
78.12.2 RPC: BGOCPTPR CLONE ... 6
78.12.3 RPC: BGOCPTPR CLONEOTH ... 7
78.12.4 RPC: BGOCPTPR DELASSOC ... 7
78.12.5 RPC: BGOCPTPR GETASSOC ... 7
78.12.6 RPC: BGOCPTPR GETCATS ... 8
78.12.7 RPC: BGOCPTPR GETITEMS ... 8
78.12.8 RPC: BGOCPTPR GETLNAME ... 8
78.12.9 RPC: BGOCPTPR GETMGRS ... 8
78.12.10 RPC: BGOCPTPR OTHCATS ... 9
78.12.11 RPC: BGOCPTPR QUERY ... 9
78.12.12 RPC: BGOCPTPR SETACHK ... 9
78.12.13 RPC: BGOCPTPR SETASSOC ... 10
78.12.14 RPC: BGOCPTPR SETCAT ... 10
78.12.15 RPC: BGOCPTPR SETFREQ ... 10
78.12.16 RPC: BGOCPTPR SETITEM ... 10
78.12.17 RPC: BGOCPTPR SETMGR ... 11
78.12.18 RPC: BGOCPTPR SETNAME ... 11
78.12.19 RPC: BGOCPTPR VSTASSOC ... 11
78.13 External Relations ... 12
78.14 Internal Relations ... 12
78.15 Archiving and Purging ... 12
78.16 Components ... 12
78.17 Properties ... 12
79.0 Family History ... 14
79.1 Introduction ... 14
79.2 Architecture and Business Process Overview ... 15

79.3 Implementation and Maintenance 15
79.4 Routine Descriptions 16
79.5 File List 16
79.6 Cross References 16
79.7 Exported Options 16
79.8 Exported Security Keys 16
79.9 Exported Protocols 17
79.10 Exported Parameters 17
79.11 Exported Mail Groups 17
79.12 Callable Routines 17
79.12.1 RPC: BGOFHX DEL 17
79.12.2 RPC: BGOFHX GET 17
79.12.3 RPC: BGOFHX ICDLKUP 18
79.12.4 RPC: BGOFHX SET 18
79.13 External Relations 18
79.14 Internal Relations 18
79.15 Archiving and Purging 19
79.16 Components 19
79.17 Properties 19

80.0 Eye Exam 20

80.1 Introduction 20
80.2 Architecture and Business Process Overview 21
80.3 Implementation and Maintenance 21
80.4 Routine Descriptions 22
80.5 File List 22
80.6 Cross References 22
80.7 Exported Options 22
80.8 Exported Security Keys 22
80.9 Exported Protocols 22
80.10 Exported Parameters 22
80.11 Exported Mail Groups 23
80.12 Callable Routines 23
80.12.1 RPC: BGOVEYE DEL 23
80.12.2 RPC: BGOVEYE GET 23
80.12.3 RPC: BGOVEYE GETFLD 23
80.12.4 RPC: BGOVEYE SET 24
80.12.5 RPC: BGOVEYE1 VAL 24
80.13 External Relations 24
80.14 Internal Relations 24
80.15 Archiving and Purging 24
80.16 Components 25
80.17 Properties 25

81.0 Anticoagulation Goal 26

81.1 Introduction 26

81.2 Architecture and Business Process Overview 26
81.3 Implementation and Maintenance 27
81.4 Routine Descriptions 27
81.5 File List 27
81.6 Cross References 27
81.7 Exported Options 27
81.8 Exported Security Keys 28
81.9 Exported Protocols 28
81.10 Exported Parameters 28
81.11 Exported Mail Groups 28
81.12 Callable Routines 28
81.12.1 RPC: BGOVCOAG DEL 28
81.12.2 BGOVCOAG GET 28
81.12.3 RPC: BGOVCOAG SET 29
81.13 External Relations 29
81.14 Internal Relations 29
81.15 Archiving and Purging 29
81.16 Components 29
81.17 Properties 29

82.0 Infant Feeding 31

82.1 Introduction 31
82.2 Architecture and Business Process Overview 32
82.3 Implementation and Maintenance 32
82.4 Routine Descriptions 33
82.5 File List 33
82.6 Cross References 33
82.7 Exported Options 33
82.8 Exported Security Keys 33
82.9 Exported Protocols 33
82.10 Exported Parameters 34
82.11 Exported Mail Groups 34
82.12 Callable Routines 34
82.12.1 RPC: BGOVIF DEL 34
82.12.2 RPC: BGOVIF GET 34
82.12.3 RPC: BGOVIF SET 34
82.13 External Relations 35
82.14 Internal Relations 35
82.15 Archiving and Purging 35
82.16 Components 35
82.17 Properties 35

83.0 Reproductive Factors 36

83.1 Introduction 36
83.2 Architecture and Business Process Overview 37
83.3 Implementation and Maintenance 37

83.4 Routine Descriptions ... 38
83.5 File List ... 38
83.6 Cross References ... 38
83.7 Exported Options ... 38
83.8 Exported Security Keys ... 38
83.9 Exported Protocols ... 38
83.10 Exported Parameters ... 39
83.11 Exported Mail Groups ... 39
83.12 Callable Routines ... 39
83.12.1 RPC: BGOREP DEL ... 39
83.12.2 RPC: BGOREP GET ... 39
83.12.3 RPC: BGOREP SET ... 40
83.12.4 RPC: BGOREP1 CONTALL ... 40
83.12.5 RPC: BGOREP1 DELCONT ... 40
83.12.6 RPC: BGOREP1 SETCONT ... 41
83.13 External Relations ... 41
83.14 Internal Relations ... 41
83.15 Archiving and Purging ... 41
83.16 Components ... 41
83.17 Properties ... 41

84.0 Suicide Form ... 43

84.1 Introduction ... 43
84.2 Architecture and Business Process Overview ... 44
84.3 Implementation and Maintenance ... 44
84.4 Routine Descriptions ... 45
84.5 File List ... 45
84.6 Cross References ... 45
84.7 Exported Options ... 45
84.8 Exported Security Keys ... 45
84.9 Exported Protocols ... 46
84.10 Exported Parameters ... 46
84.11 Exported Mail Groups ... 46
84.12 Callable Routines ... 46
84.12.1 RPC: AMHBH SUICIDE FORM DSP ... 46
84.12.2 RPC: BEHOAMH FORMIENS ... 46
84.13 External Relations ... 46
84.14 Internal Relations ... 47
84.15 Archiving and Purging ... 47
84.16 Components ... 47
84.17 Properties ... 47

85.0 SNOMED Service ... 48

85.1 Introduction ... 48
85.2 Architecture and Business Process Overview ... 48
85.3 Implementation and Maintenance ... 49

85.4 Routine Descriptions ... 49
85.5 File List ... 49
85.6 Cross References ... 49
85.7 Exported Options ... 49
85.8 Exported Security Keys ... 49
85.9 Exported Protocols ... 50
85.10 Exported Parameters ... 50
85.11 Exported Remote Procedures ... 50
85.12 Exported Mail Groups ... 50
85.13 Callable Routines ... 50
85.14 External Relations ... 50
85.15 Internal Relations ... 50
85.16 Archiving and Purging ... 50
85.17 Components ... 50
85.17.1 Execute ... 50
85.17.2 Execute_2 ... 51
85.17.3 ExecuteSubList ... 51
85.17.4 ExecuteSubList_2 ... 51
85.17.5 ExecuteICD9toSNMD ... 51
85.18 Properties ... 52

86.0 eRx Queue Service ... 53

86.1 Introduction ... 53
86.2 Architecture and Business Process Overview ... 53
86.3 Implementation and Maintenance ... 53
86.4 Routine Descriptions ... 54
86.5 File List ... 54
86.6 Cross References ... 54
86.7 Exported Options ... 54
86.8 Exported Security Keys ... 54
86.9 Exported Protocols ... 54
86.10 Exported Parameters ... 54
86.11 Exported Remote Procedures ... 54
86.12 Exported Mail Groups ... 54
86.13 Callable Routines ... 54
86.14 External Relations ... 55
86.15 Internal Relations ... 55
86.16 Archiving and Purging ... 55
86.17 Components ... 55
86.17.1 Execute ... 55
86.17.2 ViewMailbox ... 55
86.18 Properties ... 55

87.0 eRx QueueView ... 56

87.1 Introduction ... 56
87.2 Architecture and Business Process Overview ... 56

87.3 Implementation and Maintenance ... 56
87.4 Routine Descriptions ... 57
87.5 File List ... 57
87.6 Cross References ... 57
87.7 Exported Options ... 57
87.8 Exported Security Keys ... 57
87.9 Exported Protocols ... 58
87.10 Exported Parameters ... 58
87.11 Exported Mail Groups ... 58
87.12 Callable Routines ... 58
87.13 External Relations ... 58
87.14 Internal Relations ... 58
87.15 Archiving and Purging ... 58
87.16 Components ... 58
87.16.1 Properties ... 59

88.0 Designated Primary Provider ... 61

88.1 Introduction ... 61
88.2 Architecture and Business Process Overview ... 61
88.3 Implementation and Maintenance ... 62
88.4 Routine Descriptions ... 62
88.5 File List ... 62
88.6 Cross References ... 62
88.7 Exported Options ... 62
88.8 Exported Security Keys ... 62
88.9 Exported Protocols ... 63
88.10 Exported Parameters ... 63
88.11 Exported Mail Groups ... 63
88.12 Callable Routines ... 63
88.12.1 RPC: BEHOPTPC GETBDP ... 63
88.12.2 RPC: BEHOPTPC GETCATS ... 63
88.12.3 RPC: BEHOPTPC SETBDP ... 63
88.13 External Relations ... 64
88.14 Internal Relations ... 64
88.15 Archiving and Purging ... 64
88.16 Components ... 64
88.17 Properties ... 64

89.0 Level of Intervention (PHN) ... 65

89.1 Introduction ... 65
89.2 Architecture and Business Process Overview ... 66
89.3 Implementation and Maintenance ... 66
89.4 Routine Descriptions ... 67
89.5 File List ... 67
89.6 Cross References ... 67
89.7 Exported Options ... 67

89.8 Exported Security Keys 67
89.9 Exported Protocols 67
89.10 Exported Parameters 68
89.11 Exported Mail Groups 68
89.12 Callable Routines 68
89.12.1 RPC: BGOVPHN CHKPRV 68
89.12.2 RPC: BGOVPHN DEL 68
89.12.3 RPC: BGOVPHN GET 68
89.12.4 RPC: BGOVPHN SET 69
89.13 External Relations 69
89.14 Internal Relations 69
89.15 Archiving and Purging 69
89.16 Components 69
89.17 Properties 70

90.0 Direct Mail Button 71

90.1 Introduction 71
90.2 Architecture and Business Process Overview 71
90.3 Implementation and Maintenance 72
90.4 Routine Descriptions 72
90.5 File List 72
90.6 Cross References 72
90.7 Exported Options 73
90.8 Exported Security Keys 73
90.9 Exported Protocols 73
90.10 Exported Parameters 73
90.11 Exported Mail Groups 73
90.12 Callable Routines 73
90.12.1 RPC: BEHODMA PTEMADR 73
90.13 External Relations 73
90.14 Internal Relations 74
90.15 Archiving and Purging 74
90.16 Components 74

91.0 EPCS Credentialing 75

91.1 Introduction 75
91.2 Architecture and Business Process Overview 75
91.3 Implementation and Maintenance 75
91.4 Routine Descriptions 76
91.5 File List 77
91.5.1 BEH EPCS OE/RR PARAMETERS DATA (#90460.09) 77
91.5.2 BEHO EPCS INCIDENT REPORT VARIABLES (#90460.13) 77
91.5.1 BEH EPCS AUDIT LOG MESSAGES (#90460.14) 78
91.6 Cross References 79
91.7 Exported Options 79
91.8 Exported Security Keys 80

91.9 Exported Protocols 80
91.10 Exported Parameters 80
91.11 Exported Remote Procedures 80
91.12 Exported Mail Groups 81
91.13 Callable Routines 81
91.13.1 $$VRFYPHSH^BEHOEP3(.INP,PROVIEN) 81
91.13.2 RPC: BEHOEP1 ENTRYEP 81
91.13.3 RPC: BEHOEP1 GETPUBKY 81
91.13.4 RPC: BEHOEP1 LDPNDNGV 81
91.13.5 RPC: BEHOEP1 LOACREAD 82
91.13.6 RPC: BEHOEP1 PROV 82
91.13.7 RPC: BEHOEP1 READP200 82
91.13.8 RPC: BEHOEP2 DELPNDVF 82
91.13.9 RPC: BEHOEP2 ENTRYLA 83
91.13.10 RPC: BEHOEP2 INPTRANS 83
91.13.11 RPC: BEHOEP2 LDPNDGLA 83
91.13.12 RPC: BEHOEP2 PENDPROF 83
91.13.13 RPC: BEHOEP2 PROVPRFV 84
91.13.14 RPC: BEHOEP3 AUDTEVTS 84
91.13.15 RPC: BEHOEP5 ADDCHK 84
91.13.16 RPC: BEHOEP5 VRFYPHSH 84
91.14 External Relations 84
91.15 Internal Relations 85
91.16 Archiving and Purging 85
91.17 Components 85

92.0 Two-Factor Authentication Service 86

92.1 Introduction 86
92.2 Architecture and Business Process Overview 86
92.3 Implementation and Maintenance 86
92.4 Routine Descriptions 87
92.5 File List 87
92.5.1 BEH EPCS CERTIFICATE STATUS (#90460.12) 87
92.5.2 BEH EPCS CRL DISTRIBUTION POINTS (#90460.15) 88
92.6 Cross References 88
92.7 Exported Options 88
92.8 Exported Security Keys 88
92.9 Exported Protocols 88
92.10 Exported Parameters 88
92.11 Exported Remote Procedures 89
92.12 Exported Mail Groups 89
92.13 Callable Routines 89
92.13.1 EN^BEHOEPS 89
92.13.2 EN1^BEHOEPS 90
92.13.3 RXVER^BEHOEPS 90
92.13.4 RPC: BEHOEP7 AUDITSVC 90

92.13.5 RPC: BEHOEP7 BUSASVC 90
92.13.6 RPC: BEHOEP7 CERTSTAT 91
92.13.7 RPC: BEHOEP7 CHKCRTST 91
92.13.8 RPC: BEHOEP7 GETCERT 91
92.13.9 RPC: BEHOEP7 KEYHLDRS 91
92.13.10 RPC: BEHOEP7 LISTCERT 92
92.13.11 RPC: BEHOEP7 SETCERT 92
92.13.12 RPC: BEHOEP7 UTC 92
92.13.13 RPC: BEHOEPS GORDIDIG 92
92.13.14 RPC: BEHOEPS STORDSIG 93
92.14 External Relations 93
92.15 Internal Relations 93
92.16 Archiving and Purging 93
92.17 Components 93
92.18 Templates 93

93.0 Surescripts Mailbox 94

93.1 Introduction 94
93.2 Architecture and Business Process Overview 94
93.3 Implementation and Maintenance 94
93.4 Routine Descriptions 95
93.5 File List 95
93.6 Cross References 95
93.7 Exported Options 95
93.8 Exported Security Keys 95
93.9 Exported Protocols 95
93.10 Exported Parameters 96
93.11 Exported Mail Groups 96
93.12 Callable Routines 96
93.12.1 RPC: APSPESM DENIED 96
93.12.2 RPC: APSPESM GETITM 96
93.12.3 RPC: APSPESM ORDERS 96
93.12.4 RPC: APSPESM REQUESTS 97
93.12.5 RPC: APSPESM SSMBCNT 97
93.12.6 RPC: APSPESM1 RPTRPC 97
93.13 External Relations 97
93.14 Internal Relations 98
93.15 Archiving and Purging 98
93.16 Components 98
93.16.1 Properties 98

Appendix A: System Requirements 100

A.1 Minimum System Requirements 100
A.1.1 Windows – RPMS server 100
A.1.2 AIX – RPMS server 100
A.1.3 Windows – Application server 100

A.1.4 Client Workstations 100

Appendix B: Developer Tutorial 102

B.1 Introduction 102
B.2 Using Debug Mode 103
B.3 Using the Trace Log 104
B.4 About Component Support Services 107
B.5 About COM and ActiveX 108
B.6 Component Types 110
B.7 Component Registration 110
B.7.1 COM Registration 110
B.7.2 Framework Registration 110
B.7.3 Runtime Registration 111
B.8 Naming Conventions 111
B.9 Multiple vs. Single Instancing 112
B.10 Remote Procedure Calls 113
B.10.1 Create the M Routine 113
B.10.2 Create a Remote Procedure Definition 113
B.10.3 Register the Remote Procedure 114
B.10.4 Calling a Remote Procedure 115
B.10.5 Synchronous Calls 115
B.10.6 Asynchronous Calls 118
B.11 Context Management 120
B.11.1 Callbacks 120
B.11.2 Requesting a Context Change 121
B.12 Events 121
B.12.1 Defining an Event 122
B.12.2 Firing Events: Local vs. Remote 122
B.12.3 Receiving Events: Callbacks 122
B.12.4 Hierarchical Events 123
B.13 Creating Visual Components with Delphi 123
B.13.1 Creating the Active Form Project 124
B.13.2 Designing the Form 126
B.13.3 Accessing the Session Object 127
B.13.4 Accessing the Patient Context Object 129
B.13.5 Calling a Remote Procedure in Synchronous Mode 130
B.13.6 Testing the Component 131
B.13.7 Subscribing to Patient Context Changes 133
B.13.8 Calling a Remote Procedure in Asynchronous Mode 139
B.13.9 Firing an Event 141
B.13.10 Subscribing and Responding to an Event 141
B.13.11 Summary 143
B.14 Creating Visual Components with Visual Basic 143
B.14.1 Creating the ActiveX Control Project 143
B.14.2 Designing the Form 145
B.14.3 Accessing the Session Object 146

B.14.4 Accessing the Patient Context Object 148
B.14.5 Calling a Remote Procedure in Synchronous Mode 149
B.14.6 Testing the Component 149
B.14.7 Subscribing to Patient Context Changes 152
B.14.8 Calling a Remote Procedure in Asynchronous Mode 154
B.14.9 Firing an Event 156
B.14.10 Subscribing and Responding to an Event 157
B.14.11 Summary 158
B.15 Creating Visual Components with C# 158
B.15.1 Creating the Windows Control Project 159
B.15.2 Accessing the Session Object 164
B.15.3 Accessing the Patient Context Object 165
B.15.4 Calling a Remote Procedure in Synchronous Mode 167
B.15.5 Testing the Component 167
B.15.6 Subscribing to Patient Context Changes 170
B.15.7 Calling a Remote Procedure in Asynchronous Mode 173
B.15.8 Firing an Event 175
B.15.9 Subscribing and Responding to an Event 176
B.15.10 Summary 177
B.16 Creating Services 177
B.17 Creating Services with Delphi 178
B.17.1 Creating the Project 178
B.17.2 Creating the Service Object 178
B.17.3 Accessing the Session Object 180
B.17.4 Modifying the Interface 181
B.17.5 Providing the Implementation 182
B.17.6 Registering the Service 183
B.17.7 Accessing the Service 183
B.17.8 Summary 185
B.17.9 Creating Services with Visual Basic 185
B.17.10 Creating the Project 185
B.17.11 Accessing the Session Object 186
B.17.12 Modifying the Interface 188
B.17.13 Registering the Service 188
B.17.14 Accessing the Service 188
B.17.15 Summary 190
B.18 Creating Services with C# 190
B.18.1 Creating the Project 190
B.18.2 Accessing the Session Object 191
B.18.3 Modifying the Interface 192
B.18.4 Registering the Service 193
B.18.5 Accessing the Service 193
B.18.6 Summary 195
B.19 Deploying Components 195
B.20 Version Control 195

B.20.1 Version Numbers ... 196
B.20.2 Which Version ... 196
B.20.3 Registering Version Information ... 196
B.20.4 Side-by-Side Versioning ... 197
B.20.5 Imbedding Version Information ... 198
B.21 Handling Dependencies ... 200
B.22 Generating KIDS Builds ... 201
B.23 Pitfalls and Special Techniques ... 202
B.23.1 Component Initialization ... 202
B.23.2 Component Destruction ... 202
B.23.3 Other Containers ... 203
B.23.4 Focus Issues ... 203
B.23.5 Deferring Data Fetches ... 203
B.23.6 Intercomponent Communication ... 203
B.23.7 Creating Trace Log Entries ... 204
B.23.8 Embedding Licensed Controls ... 206
B.23.9 Forced Context Changes ... 207
B.23.10 Integrating Help Content ... 207
B.24 Delphi Helper Functions ... 208
B.25 Visual Basic Helper Functions ... 214

Glossary ... 216

Acronym List ... 218

Contact Information ... 219

Document Revision History

Version	Date	Author	Description	Sections
v1.1	9/2007	Doug Martin	v1.1 Release	
v1.1, p17	2/2016	Phillip Salmon	v1.1 Patch 17 Release	
v1.1, p25	8/2019	Phillip Salmon	v1.1 Patch 25 Release	• Preface • 78.4 • 79.4 • 79.12.1 – 79.12.3 • 82.10 • 84.4 • 84.13 • 84.14 • 85.14 • 86.17.1 • 88.3 • 88.4 • 89.4 • 89.12.1 • 91.0 • 92.0 • A.1.4
v1.1, p26	10/2019	Phillip Salmon	v1.1 Patch 26 Release	• 86 • 93.0

Preface

This guide provides information regarding technical aspects of the Indian Health Service RPMS Electronic Health Record (RPMS-EHR) v1.1 software. Its target audience is local and regional information technology support personnel who may be called upon to configure or troubleshoot the application. This guide has been broken into four separate files of which this is the fourth.

78.0 Super-Bills

78.1 Introduction

Super-bills are lists of CPT codes for billing and for documenting services performed. Each super-bill is attached to a visit. The Super-Bills panel (Figure 78-1) shows the super-bill items for the super-bill category below the Super-Bills button.

Figure 78-1: Super-Bills panel

Selecting one or more check boxes above the panel determines which super-bill items display in the right-hand panel.

78.2 Architecture and Business Process Overview

Figure 78-2: Architecture and business process overview

78.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	IHSBGOSUPERBILL.BGOSUPERBILL
Class Identifier	{FCDD0A6B-4DD9-4CF7-BAC5-BD1B5BFBAC00}
Image File	ihsBgoSuperBill.dll
Property Initializations	None
Serializable Properties	None
Required Files	IhsBgoSuperBill.chm
Security Keys	None
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Yes
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

78.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOCPTP." The following routines are distributed:

Routine	Description
BGOCPTPR	Routines used to manage the superbills
BGOCPTP2	Routines used to manage the superbills
BGOCPTP3	Store SNOMED association

78.5 File List

78.5.1 BGO CPT Preferences (# 90362.31)

The CPT preferences file allows the storage of CPT codes and their associations:

Field	Field No	Description
NAME	.01	Free Text

Field	Field No	Description
HOSPITAL LOCATION	.02	Pointer to hospital location (#44) file
CLINIC	.03	Pointer to clinic stop (#40.7) file
PROVIDER	.04	Pointer to file #200
OWNER	.05	Pointer to file #200
DISCIPLINE	.06	Pointer to file #7
TYPE	.07	Set of codes 0:VISIT SERVICES 1:HISTORICAL SERVICES
CPT	1	Subfile of CPT codes
MANAGER	2	Subfile of managers

78.5.1.1 MANAGERS subfile (#90362.313)

Field	Field No	Description
MANAGERS	.01	Pointer to file 200

78.5.1.2 CPT subfile (#90362.312)

Field	Field No	Description
CPT	.01	Pointer to file #81
DISPLAY TEXT	.02	Free Text
FREQUENCY	.03	Numeric
MODIFIER FIRST	.04	Pointer to file 9002274.07
MODIFIER SECOND	.05	Pointer to file 9002274.07
TRANSACTION CODE	.06	Pointer to file 90092.02
ASSOCIATION	1	Subfile of associations

78.5.1.3 ASSOCIATIONS subfile (#90362.3121)

Field	Field No	Description
ASSOCIATION	.01	Variable pointer to 81 CPT 9999999.88CPT MODIFIER 80 ICD DIAGNOSIS 9999999.64HEALTH FACTOR 9999999.09EDUCATION TOPIC 9999999.15EXAM 9999999.14IMMUNIZATION 9999999.28SKIN TEST 90092.02TRANSACTION 80.1 ICD PROCEDURE 81.3 CSV CPT MODIFIER
AUTOMATICALLY ADD	.02	Set of codes, No/Yes
DEFAULT TO ADD	.03	Set of codes, No/Yes
PROHIBIT DUPLICATION	.04	Set of codes, No/Yes
ASK CHARGE AMOUNT	.05	Set of codes, No/Yes
ASK QUANTITY	.06	Set of codes, No/Yes
DEFAULT QUANTITY AMOUNT	.07	Numeric
SNOMED CT CONCEPT CODE	1	Free text
SNOMED CT DESCRIPTIVE CODE	1.1	Free text

78.6 Cross References

File: BGO CPT PREFERENCES (#90362.31)

X-ref	Field No	Name	Desc
AC	.03	Clinic	Regular
ACP	.03	Clinic	Mumps
ACPTOO	.04	Provider	Mumps
AD	.07	Discipline	Regular
AH	.02	Hospital Location	Regular
AHP	.02	Hospital Location	Mumps
AHPTOO	.04	Provider	Mumps
AP	.04	Provider	Regular
B	.01	Name	Regular

78.7 Exported Options

None.

78.8 Exported Security Keys

None.

78.9 Exported Protocols

None.

78.10 Exported Parameters

None.

78.11 Exported Mail Groups

None.

78.12 Callable Routines

This section describes supported entry points for routines exported with this component.

78.12.1 RPC: BGOCPTP3 STORE

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: DFN |1| ^ VIEN |2| ^ SNOMED CT |3| ^ ICD |4|^ LOCATION |5|^ PROVIDER |6|
<return value>	Boolean	Success: True

Stores SNOMED association DXs for a superbill entry. If SNOMED code does not exist on patient's problem list, add it and then add the item as a POV.

78.12.2 RPC: BGOCPTPR CLONE

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Pref IEN (from) |1|^ Pref IEN (to) |2|
<return value>	String	Success: True

Clones a pick list.

78.12.3 RPC: BGOCPTPR CLONEOTH

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: CPT Category IEN |1| ^ Preference Category IEN |2|
<return value>	Boolean	Success: True

Convert a CPT category into a pick list.

78.12.4 RPC: BGOCPTPR DELASSOC

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN |1| ^ Item IEN |2| ^ Element IEN |3|
<return value>	String	Success: True

Delete an association.

78.12.5 RPC: BGOCPTPR GETASSOC

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN |1|^ Item IEN |2|
<return value>	String List	Returned as a list of records in the format: Item IEN [1] ^ Item Name [2] ^ Type [3] ^ Auto Add [4] ^ Auto Default [5] ^ No Dups [6] ^ Amount [7] ^ Association IEN [8] ^ Quantity [9] ^ Code [10] ^ Type ID [11]

Returns a list of associations for the specified pick list and item.

78.12.6 RPC: BGOCPTPR GETCATS

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Hospital Location IEN [2] ^ Provider IEN [3] ^ Manager IEN [4] ^ Show All [5] ^ Historical Flag [6]
<return value>	String List	Returns a list of records in the format: Category Name [1] ^ Category IEN [2] ^ Hosp Loc Name [3] ^ Hosp Loc IEN [4] ^ Clinic Stop Name [5] ^ Clinic Stop IEN [6] ^ Provider Name [7] ^ Provider IEN [8] ^ Owner Name [9] ^ Owner IEN [10] ^ Provider Class Name [11] ^ Provider Class IEN [12]

Returns a list of categories matching the specified criteria.

78.12.7 RPC: BGOCPTPR GETITEMS

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Group [2] ^ Visit IEN [3] ^ Display Freq Order [4]
<return value>	String List	Failure: -n^error text Success: List of records in the format CPT IEN [1] ^ CPT Code [2] ^ CPT Text [3] ^ Short Text [4] ^ Freq [5] ^ VCPT IEN [6] ^ Fee [7] ^ Rank [8] ^ Pref IEN [9] ^ Association [10] ^ Long Text [11]

Returns a list of categories matching the specified criteria.

78.12.8 RPC: BGOCPTPR GETLNAME

Scope: Private

Parameter	Datatype	Description
IEN	Pointer (#81)	IEN of CPT code
<return value>	String	Long narrative text.

Return the long name for the specified CPT code.

78.12.9 RPC: BGOCPTPR GETMGRS

Scope: Private

Parameter	Datatype	Description
CAT	Pointer (#90362.31)	IEN of pick list

Parameter	Datatype	Description
<return value>	String List	List of records in the format: Provider Name ^ Provider IEN

Returns a list of managers associated with the specified pick list.

78.12.10 RPC: BGOCPTPR OTHCATS

Scope: Private

Parameter	Datatype	Description
<return value>	String List	List of records in the format: Category Name [1] ^ Category IEN [2] ^ Class [3] where Class is one of: Med, Surg, Anesth, Rad, Lab

Returns a list of categories from the CPT file.

78.12.11 RPC: BGOCPTPR QUERY

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Provider IEN [2] ^ Clinic IEN [3] ^ Provider Class [4] ^ Hospital Location [5] ^ Start Date [6] ^ End Date [7] ^ Max Hits [8] ^ Med [9] ^ Surg [10] ^ Anest [11] ^ Lab [9] ^ Rad [12] ^ Supply [13] ^ 3rd Party Billing [14] ^ V CPT [15] ^ CHS [16]
<return value>	Boolean	Success: True

Execute a query to update frequencies of use.

78.12.12 RPC: BGOCPTPR SETACHK

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: CPT Preference IEN [1] ^ CPT Subfile IEN [2] ^ Associations Subfile IEN [3] ^ Column Index [4] ^ Value [5]
<return value>	String	Failure: -n^error text Success: IEN

Modify an association of a CPT in a superbill.

78.12.13 RPC: BGOCPTPR SETASSOC

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: CPT Preference IEN [1] ^ CPT Subfile IEN [2] ^ Type [3] ^ Value [4] ^ Association [5] ^ Auto Add [6] ^ Default to Add [7] ^ No Dups [8] ^ Amount [9] ^ Quantity [10]
<return value>	String	Failure: -n^error text Success: IEN of association

Set an association of a CPT in a superbill and returns the IEN of the subfile entry.

78.12.14 RPC: BGOCPTPR SETCAT

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Name [1] ^ Hosp Loc [2] ^ Clinic [3] ^ Provider [4] ^ User [5] ^ Category IEN [6] ^ Delete [7] ^ Discipline [8]
<return value>	String	Failure: -n^error text Success: IEN of pick list

Set field values for a pick list.

78.12.15 RPC: BGOCPTPR SETFREQ

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Item Value (defaults to all) [2] ^ Increment [3] ^ Frequency [4]
<return value>	String	Failure: 0^error text Success: True

Set frequency for a pick list item.

78.12.16 RPC: BGOCPTPR SETITEM

Update the frequency of a CPT code in a category.

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ CPT IEN [2] ^ Display Text [3] ^ Delete [4] ^ CPT Code [5] ^ Frequency [6] ^ Allow Dups [7] ^ Item IEN [8]
<return value>	String	Failure: -n^error text Success: IEN

Returns the new IEN of the CPT code added to the category.

78.12.17 RPC: BGOCPTPR SETMGR

Add a manager to a CPT preferences category.

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Manager IEN (#200) [2] ^ Add [3]
<return value>	String	Failure: -n^error text Success: null

Add or remove a manager from a pick list.

78.12.18 RPC: BGOCPTPR SETNAME

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN [1] ^ Item IEN [2] ^ Display Name [3]
<return value>	String	Failure: -n^error text Success: null

Set display name for a pick list item.

78.12.19 RPC: BGOCPTPR VSTASSOC

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Category IEN ^ Item IEN ^ Visit IEN
<return value>	String List	Returns a list of records in the format: Type ID [1] ^ Type Name [2] ^ Item IEN [3] ^ Item Text [4] ^ V File IEN [5] ^ V File # [6]

Returns all V file entries for a given visit that correspond to all associated entries for the given superbill.

78.13 External Relations

Entity	Name	Description
Component	VCPT	Calls supported APIs
Component	Visit Diagnosis	Calls supported APIs
Component	Health Factors	Calls supported APIs
Component	Patient Education	Calls supported APIs
Component	Exams	Calls supported APIs
Component	Immunization	Calls supported APIs
Component	Skin Test	Calls supported APIs

78.14 Internal Relations

Entity	Name	Description
Package	BGO	Calls supported APIs

78.15 Archiving and Purging

There are no archiving or purging requirements within this software.

78.16 Components

This component supports the following properties and methods:

78.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent

Property	Datatype	Access	Description
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

79.0 Family History

79.1 Introduction

The Family History component (Figure 79-1) displays any information about the patient

Family History | Asthma Zone

Family History List

Use Edit Relation to delete, add, or edit a relative's condition
* Requires update to SNOMED CT

Get SCT | Add Relation | Edit Relation | Delete Relation

Relation	Name	Status	Age At Death	Cause of Death	Multiple Birth	Multiple Birth Type	Provider Narrative	Age at Diagnosis	Date Modified	ICD
BROTHER	Danny	LIVING							07/24/2013	
COUSIN (MALE)	Mike	LIVING			NO				08/09/2013	
COUSIN		LIVING							06/07/2013	V16.0
BROTHER	Jack	LIVING							05/14/2013	V16.3
NATURAL MOTHER	Jeanne	DECEASED	60 and	cancer	NO		*Family History Of Unspecified Malignant Neoplasm. I am testing to		05/02/2013	V16.9
NATURAL MOTHER	Jeanne	DECEASED	60 and	cancer	NO		FH: premature coronary heart disease |		07/31/2013	V17.1
BROTHER	George	LIVING					Family history of myocardial infarction | |397701010		07/17/2013	V17.3
GRANDFATHER	Charles	DECEASED	60 and				FH: Congenital heart disease | testing		07/12/2013	V17.49
GRANDFATHER	Charles	LIVING					FH: Cardiac disorder |		07/12/2013	V17.49
GRANDFATHER	Charles	LIVING					FH: Diabetes mellitus in first degree relative | this is the provider	50	07/12/2013	V18.0
SISTER	Jessica	LIVING					FH: Diabetes mellitus | this is the text|2666212011	15	06/12/2013	V18.0

Figure 79-1: Family History component

79.2 Architecture and Business Process Overview

Figure 79-2: Architecture and business process overview

79.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	IHSBGOFAMHX.BGOFAMHX
Class Identifier	{F275DE03-BF24-4759-B34C-2DA9F792754E}
Image File	bgoFamHx.ocx
Property Initializations	RUNASYNC=FALSE HIDEBUTTONS=FALSE

Entity	Value
Serializable Properties	HIDEBUTTONS – BOOL USELEXICON - BOOL
Required Files	IhsBgoFamHX.chm
Security Keys	PROVIDER
Multiple Instances Allowed	Yes
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	No
.Net Component	No
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

79.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOFHX." The following routines are distributed:

Routine	Description
BGOFHX	Family History component support
BGOREL	Family History relationship support

79.5 File List

None.

79.6 Cross References

None.

79.7 Exported Options

None.

79.8 Exported Security Keys

None.

79.9 Exported Protocols

None.

79.10 Exported Parameters

None.

79.11 Exported Mail Groups

None.

79.12 Callable Routines

79.12.1 RPC: BGOFHX DEL

Scope: Private

Parameter	Datatype	Description
INP	String	INP= Relationship IEN [1] ^ Family HX ien [2] (#9000014.1) (#9000014)
<return value>	Boolean	1 for success or 0^error text

If no family history IEN is included, the entire relationship will be deleted else just delete the family history dx. This is a physical deletion.

79.12.2 RPC: BGOFHX GET

Scope: Private

Parameter	Datatype	Description
DFN	Pointer (#2)	Patient IEN
<return value>	Array	;.RET returned as a list of records in the format ; Relationship IEN [1] Relationship [2] ^ Status [3] ^ Age at Death [4 ^ Cause of Death [5] ^; Multiple Birth [6] ^ Multiple Birth Type [7] ^ Condition [8] ^ Narrative [9] ^ [10] ^ Date Modified [11] ^Description [12] ^Family hx IEN [13]^ Age at DX [14] ^Age at DX Approximate [15] ^ Snomed CT [16] ^ Snomed Desc ID [17] ^ List of Additional ICD codes - ";" delimited [18]

Returns an array of each family history entry with the relationship. A relative who has multiple DXs will have multiple lines in the array.

79.12.3 RPC: BGOFHX ICDLKUP

Scope: Private

Parameter	Datatype	Description
INP	String	Formatted as: Lookup Value^Visit Date^Patient Gender^From^To
<return value>	Array	List formatted as: Descriptive Text^ICD IEN^Narrative Text

ICD Codes used for family history.

79.12.4 RPC: BGOFHX SET

Scope: Private

Parameter	Datatype	Description
DFN	Pointer (#2)	Patient IEN
LIST(1)	Array (1)	Relationship Data REL^ Relationship ien [2] ^ Relationship [3] ^ Relationship Desc [4] ^ Status [5] ^ Age at Death [6] Cause of Death [7] ^ Multiple Birth [8] ^ Multiple Birth Type [9]
LIST(n)	Array	Family Hx DX data FHX^ Family HS ien [2]^ DX [3] ^ Text [4] ^ DX Age [5] ^ DX Age Approximate [6] ^ concept ct [7] ^ DESC CT [8] ^ MULT ICD [9]
<return value>	String	Each item stored in the call with a flag if it's the R(elationship) or F(amily history dx)

Returns an array of each family history entry with the relationship. A relative who has multiple DXs will have multiple lines in the array.

79.13 External Relations

Entity	Name	Description
Package	IHS STANDARD TERMINOLOGY	Uses Standard API calls to look up SNOMED codes
Package	IHS ICD/CPT LOOKUP & GROUPER	Uses standard API calls to lookup ICD codes

79.14 Internal Relations

None.

79.15 Archiving and Purging

There are no archiving or purging requirements within this software.

79.16 Components

This component supports the following properties and methods:

79.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

80.0 Eye Exam

80.1 Introduction

The Eyeglass Prescription component is designed for optometrists to enter the prescription for eyeglasses. The prescription can then be printed and filled at another establishment, or filled locally.

Select a patient and encounter, and then select the Eyeglass Prescription module. If the patient has an eyeglass prescription, the most recent one appears in the dialog.

Figure 80-1: Eyeglass Prescription dialog

80.2 Architecture and Business Process Overview

Figure 80-2: Architecture and business process overview

80.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	IHSBGOEYEEXAM.BGOEYEEXAM
Class Identifier	{D1C64A19-3DCC-4B4C-B2CA-68C8A5030176}
Image File	IhsBgoEyeExam.dll
Property Initializations	none
Serializable Properties	none
Required Files	IhsBgoEyeExam.chm
Security Keys	none
Multiple Instances Allowed	no
Internal Property Editor	no
All Keys Required	no

Entity	Value
Hidden from Property Editor	no
Side-by-Side Versioning	no
Service	no
.Net Component	yes
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

80.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOVEYE." The following routines are distributed:

Routine	Description
BGOVEYE	Used to Get and set eye prescriptions
BGOVEYE1	Used to print rxs and validate data

80.5 File List

None.

80.6 Cross References

None.

80.7 Exported Options

None.

80.8 Exported Security Keys

Key	Description
BGOZ EYE EDIT	Used to Get and set eye prescriptions

80.9 Exported Protocols

None.

80.10 Exported Parameters

None.

80.11 Exported Mail Groups

None.

80.12 Callable Routines

80.12.1 RPC: BGOVEYE DEL

Scope: Private

Parameter	Datatype	Description
IEN	Pointer (#9000010.04)	IEN of V EYE GLASS file
<return value>	Boolean	1 for success or 0^error text

Physically deletes an item in the V EYE GLASS file.

80.12.2 RPC: BGOVEYE GET

Scope: Private

Parameter	Datatype	Description
DFB	Pointer (#2)	Patient IEN
VIEN	Pointer (#9000010)	Visit IEN
<return value>	String List	List formatted as: ; RET(1)=IEN [1] ^ Visit Date [2] ^Facility Name [3] ^Provider IEN [4] ^ Location Name [5] ^ Entered Date [6] ^ Visit IEN [7] ^ Visit Category [8] ^ Visit Locked [9] ; RET(2)=Left sphere [1] ^ left cyl [2] ^ left axis [3] ^ L prism H [4] ^ L Prism HD [5] ^ L Prism V [6] ^ L Prism VD [7] ^ L reading [8] ; RET(3)=Right sphere [1] ^ Right cyl [2] ^ Right axis [3] ^ R prism H [4] ^ R Prism HD [5] ^ R Prism V [6] ^ R Prism VD [7] ^ R reading [8] ; RET(4)=Reading [1] ^ PD Near [2] ^ PD Far [3] ^ LPD [4] ^ RPD [5] ; RET(5)=Comment

Returns an array of data for the last eye exam or the visit eye exam.

80.12.3 RPC: BGOVEYE GETFLD

Scope: Private

Parameter	Datatype	Description
None		
<return value>	String List	Array or help text for the fields in the eye exam file

Returns an array of help text for the fields in the eye exam.

80.12.4 RPC: BGOVEYE SET

Scope: Private

Parameter	Datatype	Description
INP	Array	; DATA(0)=V File IEN (if edit) [1] ^ Patient ien [2] ^ visit ien [3] ^ provider [4] ^Event Date [5] ^ Location IEN [6] ^ Other Location [7] ^ Historical Flag [8] ; DATA(1)=Left sphere [1] ^ left cyl [2] ^ left axis [3] ^ L prism H [4] ^ L Prism HD [5] ^ L Prism V [6] ^ L Prism VD [7] ^ L reading [8] ; DATA(2)=Right sphere [1] ^ Right cyl [2] ^ Right axis [3] ^ R prism H [4] ^ R Prism HD [5] ^ R Prism V [6] ^ R Prism VD [7] ^ R reading [8] ; DATA(3)=Reading [1] ^ PD Near [2] ^ PD Far [3] ^ LPD [4] ^ RPD [5] ; DATA(4)=Comment
<return value>	String	IEN of entry or an error message

Creates a new entry in the V EYE GLASS file or edits an existing entry.

80.12.5 RPC: BGOVEYE1 VAL

Scope: Private

Parameter	Datatype	Description
INP	String	Field name ^ value
<return value>	String	Returns the value or an error message

Validates what is entered for a field against the field definition in PCC.

80.13 External Relations

Entity	Name	Description
Package	IHS PCC DATA	Users standard PCC APIs and Files

80.14 Internal Relations

None.

80.15 Archiving and Purging

There are no archiving or purging requirements within this software.

80.16 Components

This component supports the following properties and methods:

80.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

81.0 Anticoagulation Goal

81.1 Introduction

The Anticoagulation module enables clinicians to establish anticoagulation goals for their patients. This sets and tracks the goals. The goals can be modified as needed, based on anticoagulation results and current medication therapy.

Anticoagulation

Add Edit Delete

Indication	Visit Date	INR Goal	Min	Max	Duration	Start Date	End Date	Entered Date	Category	Comment	ProviderName
NO	8/29/2013	Other	1	1	3 MONTHS	8/30/2013	11/30/2013	8/30/2013	A		USER,DEMO
YES	8/20/2013	2.0 - 3.0			6 MONTHS	8/30/2013	2/28/2014	8/30/2013	A		USER,DEMO

Figure 81-1: Anticoagulation component

81.2 Architecture and Business Process Overview

Figure 81-2: Architecture and business process overview

81.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHANTICOAG.BEHANTICOAG
Class Identifier	{0C8D9769-D849-4D5B-B495-17C9E11DB1B7}
Image File	BEHAntiCoag.dll
Property Initializations	None
Serializable Properties	None
Required Files	BEHAntiCoag.chm
Security Keys	None
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Nes
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

81.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOVCOAG." The following routines are distributed:

Routine	Description
BGOVCOAG	Used to Get and set anticoagulant data

81.5 File List

None.

81.6 Cross References

None.

81.7 Exported Options

None.

81.8 Exported Security Keys

None.

81.9 Exported Protocols

None.

81.10 Exported Parameters

None.

81.11 Exported Mail Groups

None.

81.12 Callable Routines

81.12.1 RPC: BGOVCOAG DEL

Scope: Private

Parameter	Datatype	Description
VFIEN	String	Delete reason is a set of codes INP = V File IEN ^ DELETE REASON ^ OTHER
<return value>	Boolean	1 for success or 0^error text

Logically deletes an item in the V ANTICOAGULATION file. If the delete reason is OTHER, a text comment must be sent.

81.12.2 BGOVCOAG GET

Scope: Private

Parameter	Datatype	Description
INP	String	INP = Patient IEN ^ Number to return
<return value>	Array	.RET = Returned as a list of records: V IEN [1] ^ Indication [2] ^ Visit Date [3] ^ Goal [4] ^ min [5] ^ max [6] ^ duration [7] ^Strt Date [8] ^Facility Name [9] ^ Provider IEN [10] ^ Location IEN [11] ^ Entered Date [12] ^ Visit ^ Visit Locked [15] ^ COMMENT[16] ^ Provider Name [17]

Returns an array of V ANTICOAGULATION for the selected patient.

81.12.3 RPC: BGOVCOAG SET

Scope: Private

Parameter	Datatype	Description
INP	Array	INP = V anticoag IEN (if edit) [1] ^Indication [2] ^ Patient IEN [3] ^ Visit IEN [4] ^ Provider IEN [5] ^ Goal [6] ^ MIN [7] ^ Max [8] ^Duration [9] ^ Strt date [10] ^Event Date [11] ^ Location IEN [12] ^ Other Location [13] ^ Historical Flag [14] ^comment [15]
<return value>	String	IEN of entry or an error message

Creates a new entry in the V anticoagulation file or edits an existing entry.

81.13 External Relations

None.

81.14 Internal Relations

None.

81.15 Archiving and Purging

There are no archiving or purging requirements within this software.

81.16 Components

This component supports the following properties and methods:

81.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent

Property	Datatype	Access	Description
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

82.0 Infant Feeding

82.1 Introduction

The Infant Feeding component is only active for children less than five years old. Otherwise, the Not Applicable message displays.

The Infant Feeding History grid displays the feeding choices, the event dates already entered for the current (infant) patient, and any secondary fluids given to the infant. The records are listed in date order, with the most recent on top. This component requires that a visit is selected in order to update the data.

A red 1 in the column before the Feeding Choice column indicates the visit is locked (and the record cannot be edited), and a zero indicates the visit is not locked and can be edited.

This data is collected because it is used in conjunction with the Childhood Weight Control Government Performance and Results Act (GPRA) measure. This is a long-term measure to support the reduction of the incidence of childhood obesity. Breastfeeding rates are used in the Program Assessment Rating Tool (PART) report for congress and Office of Management and Budget (OMB). Additionally, facilities can use this data to track infant feeding patterns and breastfeeding rates within their own patient populations.

Figure 82-1: Infant Feeding component

82.2 Architecture and Business Process Overview

Figure 82-2: Architecture and business process overview

82.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	IHSBGOINFANTFEED.IHSBGOINFANTFEEDCTRL
Class Identifier	{DAD1EB63-400C-4AB1-834A-DA36E56FB96C}
Image File	IhsbgoInfantFeed.ocx
Property Initializations	None
Serializable Properties	None
Required Files	IhsBgoInfantFeed.chm
Security Keys	None

Entity	Value
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	No
.Net Component	No
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

82.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOVIF." The following routines are distributed:

Routine	Description
BGOVIF	Used to Get and set infant feeding data

82.5 File List

None.

82.6 Cross References

None.

82.7 Exported Options

None.

82.8 Exported Security Keys

None.

82.9 Exported Protocols

None.

82.10 Exported Parameters

None.

82.11 Exported Mail Groups

None.

82.12 Callable Routines

82.12.1 RPC: BGOVIF DEL

Scope: Private

Parameter	Datatype	Description
VFIEN	Pointer (#9000010.44)	IEN of Infant Feeding to be deleted
<return value>	Boolean	1 for success or 0^error text

Physically deletes an item in the infant feeding file.

82.12.2 RPC: BGOVIF GET

Scope: Private

Parameter	Datatype	Description
IMP	String	If only the first piece is sent in, it returns all infant feeding data for this patient INP = Patient IEN [1] ^ V File IEN [2] ^ Visit IEN [3]
<return value>	Array	List of records in the format: IEN [1] ^ Visit Locked [2] ^ Visit [3] ^ Feeding Choice [4] ^Event Date [5] ^ Encounter Provider [6] Where each field except IEN and Visit Locked has the form: External Value | Internal Value

Returns an array of infant feeding data for the selected patient.

82.12.3 RPC: BGOVIF SET

Scope: Private

Parameter	Datatype	Description
INP	String	If the IEN is sent, it is an edit, otherwise it makes a new entry INP = V File IEN ^ Visit IEN ^ Feeding Choice ^ EXTRA EXTRA can be more than one separated by ~
<return value>	String	IEN OF entry or an error message

Creates a new entry in the infant feeding file or edits an existing entry.

82.13 External Relations

None.

82.14 Internal Relations

None.

82.15 Archiving and Purging

There are no archiving or purging requirements within this software.

82.16 Components

This component supports the following properties and methods:

82.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

83.0 Reproductive Factors

83.1 Introduction

The Reproductive Factors application is available for female patients only. After selecting a female patient, the application displays the most recent reproductive factors information, if available. A visit does not need to be selected. The Reproductive Factors application is used for populating PCC data in RPMS.

Figure 83-1: Reproductive Factors

83.2 Architecture and Business Process Overview

Figure 83-2: Architecture and business process overview

83.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	IHSBGOREPFACTORS.IHSBGOREPFACTORSCTRL
Class Identifier	{BCDD2780-00A6-40D5-B5DD-07DEFC1A78E5}
Image File	IHSbgoRepFactors.ocx
Property Initializations	None
Serializable Properties	None

Entity	Value
Required Files	IhsBgoRepFactors.chm
Security Keys	None
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	No
.Net Component	Yes
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

83.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOREP." The following routines are distributed:

Routine	Description
BGOREP	Used to Get reproductive history data
BGOREP1	Utility calls for Reproductive factors and calls for contraceptive data

83.5 File List

None.

83.6 Cross References

None.

83.7 Exported Options

None.

83.8 Exported Security Keys

None.

83.9 Exported Protocols

None.

83.10 Exported Parameters

None.

83.11 Exported Mail Groups

None.

83.12 Callable Routines

83.12.1 RPC: BGOREP DEL

Scope: Private

Parameter	Datatype	Description
DFN	String	Patient's IEN
<return value>	String	Failure: -n^error text Success: null

Delete reproductive factors for a patient.

83.12.2 RPC: BGOREP GET

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Patient IEN |1| ^ Date Obtained |2| ^ Expand History (opt) |3| ^
<return value>	String List	Formatted as: "L" ^ LMP Date |2| "C" ^ Contraception Method |2| ^ Contraception Begun |3| "P" ^ How EDC Determined |2| ^ EDC Date |3| ^ Pregnant |4| "R" ^ Gravida |2| ^ Date |3| ^ Multiple Births |4| ^ Date |5| ^ Full term |6| ^ Date |7| ^ Premature |8| ^ Date |9| ^ Ectopics |10| ^ Date |11| ^ Living Children |12| ^ Date |13| ^ Spontaneous abortions |14| ^ Date |15| ^ Therapeutic abortions |16| ^ Date |17|

Returns reproductive factor information for a patient.

83.12.3 RPC: BGOREP SET

Scope: Private

Parameter	Datatype	Description
INP	String	Specified as: Patient IEN [1] ^ LMP Date [2] ^ Contraceptive Method [3] ^ Contraception Begun [4] ^ EDC Date [5] ^ EDC Determined [6] ^ Gravida [7] ^ Date Updated [8] ^ Multiple Births [9] ^ Date Updated [10] ^ Full term[11] ^ Date Updated [12] ^Premature[13] ^ Date Updated [14] ^Ectopics[15] ^ Date Updated [16] ^Living Children [17] ^ Date Updated [18] ^ Spontaneous abortions[19] ^ Date Updated [20] ^Therapeutic abortions[21]^Date Updated[22] ^Currently pregnant [23]
<return value>	String	Failure: -n^error text Success: null

Supports add/edit reproductive factors for a patient.

83.12.4 RPC: BGOREP1 CONTALL

Scope: Private

Parameter	Datatype	Description
DFN	String	Patient's IEN
<return value>	String List	Formatted as: IEN of subfile [1]^method [2]^date started [3]^date ended [4]^reason DC[5]^ comment[6]

Returns a list of all contraceptive entries for a patient.

83.12.5 RPC: BGOREP1 DELCONT

Scope: Private

Parameter	Datatype	Description
DFN	String	Patient's IEN
IEN	String	Contraceptive Subfile IEN
<return value>	String	Failure: -n^error text Success: null

Removes a contraceptive for a patient.

83.12.6 RPC: BGOREP1 SETCONT

Scope: Private

Parameter	Datatype	Description
DFN	String	Patient's IEN
DATA	String List	Formatted as: IEN of subfile (if edit) |1| ^ Type of contraception |2| ^ Date start |3| ^ Date end |4| ^ reason DC |5| ^ comment |6|
<return value>	String	Failure: -n^error text Success: DFN

Stores contraceptives for a patient.

83.13 External Relations

None.

83.14 Internal Relations

None.

83.15 Archiving and Purging

There are no archiving or purging requirements within this software.

83.16 Components

This component supports the following properties and methods:

83.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent

Property	Datatype	Access	Description
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

84.0 Suicide Form

84.1 Introduction

You use the Suicide Form component to record suicide incidents for the patient on the Suicide Reporting Form.

Figure 84-1: Suicide Form (Patient Centric)

Figure 84-2: Suicide Form (non-Patient Centric)

84.2 Architecture and Business Process Overview

Figure 84-3: Architecture and business process overview

84.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	INDIANHEALTHSERVICE.BEH.IBH.SUICIDE.CONTROLS.CTLSUICIDE_FORM
Class Identifier	{A47DFD65-4878-4635-A4F3-E9122CFCC453}
Image File	SuicideForm.dll
Property Initializations	None

Entity	Value
Serializable Properties	PatientCentric – BOOL
Required Files	Suicide_Form.chm
Security Keys	APCDZ SUICIDE FORMS
Multiple Instances Allowed	Yes
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Yes
Associated Build	BEHO*1.1*013004

There are no specific implementation or maintenance tasks associated with this component.

84.4 Routine Descriptions

This component has been assigned the namespace designation of "BEHOAMH." The following routines are distributed:

Routine	Description
BEHOAMH	Support for Suicide Form

84.5 File List

None.

84.6 Cross References

None.

84.7 Exported Options

None.

84.8 Exported Security Keys

None.

84.9 Exported Protocols

None.

84.10 Exported Parameters

None.

84.11 Exported Mail Groups

None.

84.12 Callable Routines

84.12.1 RPC: AMHBH SUICIDE FORM DSP

Scope: Private

Parameter	Datatype	Description
AMHIEN	String	Suicide Form IEN
<return value>	String List	Suicide Form Display content

Returns Suicide Form display content.

84.12.2 RPC: BEHOAMH FORMIENS

Scope: Private

Parameter	Datatype	Description
IEN	String	MHSS SUICIDE FORMS IEN
<return value>	String	Formatted as: Fld #: IEN; Fld #:IEN; Fld #:IEN Sample: .03:1^.07:2429^.25:2

Returns the pointed to IENs for field data in the MHSS SUICIDE FORM file.

84.13 External Relations

Entity	Name	Description
Library	IHS RPC Broker (BGU)	General Utilities to support calls to RPMS using the CIA Listener
File	MHSS SUICIDE FORMS	Contains content related to an attempt.

84.14 Internal Relations

None.

84.15 Archiving and Purging

There are no archiving or purging requirements within this software.

84.16 Components

This component supports the following properties and methods:

84.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

85.0 SNOMED Service

85.1 Introduction

The SNOMED Service provides access to the SNOMED Search dialog used to select a SNOMED Concept ID.

85.2 Architecture and Business Process Overview

Figure 85-1: Architecture and business process overview

85.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHSNMDSVC.SNMDSVC
Class Identifier	{78BA9201-1BAE-4341-B8F5-54762E12825F}
Image File	BEHSNMDSvc.dll
Property Initializations	None
Serializable Properties	None
Required Files	None
Security Keys	None
Multiple Instances Allowed	N/A
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	Yes
.Net Component	Yes
Associated Build	BEHO*1.1*013004

There are no specific implementation or maintenance tasks associated with this component.

85.4 Routine Descriptions

None.

85.5 File List

None.

85.6 Cross References

None.

85.7 Exported Options

None.

85.8 Exported Security Keys

None.

85.9 Exported Protocols

None.

85.10 Exported Parameters

None.

85.11 Exported Remote Procedures

None.

85.12 Exported Mail Groups

None.

85.13 Callable Routines

None.

85.14 External Relations

Entity	Name	Description
Package	IHS Standard Terminology 2.0	Support APIs for SNOMED search
Dialog	INDIANHEALTHSERVICE. SNOMEDCTSEARCH.DLL	External library that prompts the user for SNOMED Concept ID selection.

85.15 Internal Relations

Entity	Name	Description
Library	BEHCPRS20.bpl	Finds and instantiates the service.

85.16 Archiving and Purging

There are no archiving or purging requirements within this software.

85.17 Components

This component supports the following properties and methods:

85.17.1 Execute

Scope: public

Parameter	Datatype	Description
<Return value>	String	SNOMED DescriptionID^ConceptID^Description^Associated ICD codes delimited by ';'

Invokes the SNOMED Search dialog.

85.17.2 Execute_2

Scope: public

Parameter	Datatype	Description
ValueToSearch	String	Term to restrict list
<Return value>	String	SNOMED DescriptionID^ConceptID^Description^Associated ICD codes delimited by ';'

Invokes the SNOMED Search dialog with a term for lookup.

85.17.3 ExecuteSubList

Scope: public

Parameter	Datatype	Description
Defaults	String	List of subsets to be included on the left portion of the search dialog
Selected	String	List of subsets to be selected on the left portion of the search dialog
<Return value>	String	SNOMED DescriptionID^ConceptID^Description^Associated ICD codes delimited by ';'

Invokes the SNOMED Search dialog.

85.17.4 ExecuteSubList_2

Scope: public

Parameter	Datatype	Description
ValueToSearch	String	Term to restrict list
Defaults	String	List of subsets to be included on the left portion of the search dialog
Selected	String	List of subsets to be selected on the left portion of the search dialog
<Return value>	String	SNOMED DescriptionID^ConceptID^Description^Associated ICD codes delimited by ';'

Invokes the SNOMED Search dialog.

85.17.5 ExecuteICD9toSNMD

Scope: public

Parameter	Datatype	Description
ValueToSearch	String	Term to restrict list
Defaults	String	List of subsets to be included on the left portion of the search dialog
Selected	String	List of subsets to be selected on the left portion of the search dialog
<Return value>	String	SNOMED DescriptionID^ConceptID^Description^Associated ICD codes delimited by ';'

Invokes the SNOMED Search dialog.

85.18 Properties

None

86.0 eRx Queue Service

86.1 Introduction

The eRx Queue Service provides access to the Surescripts refill request form in the BEH EHR support library.

86.2 Architecture and Business Process Overview

Figure 86-1: Architecture and business process overview

86.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHERXQUEUE.QUEUE
Class Identifier	{CF925F24-B500-4E89-B550-75BB84CB4CDB}
Image File	BEHeRxQueue.dll
Property Initializations	none
Serializable Properties	none
Required Files	none
Security Keys	none
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	Yes
.Net Component	No

Entity	Value
Associated Build	BEHO*1.1*062001

There are no specific implementation or maintenance tasks associated with this component.

86.4 Routine Descriptions

None.

86.5 File List

None.

86.6 Cross References

None.

86.7 Exported Options

None.

86.8 Exported Security Keys

None.

86.9 Exported Protocols

None.

86.10 Exported Parameters

None.

86.11 Exported Remote Procedures

None.

86.12 Exported Mail Groups

None.

86.13 Callable Routines

None.

86.14 External Relations

Entity	Name	Description
Library	BEH EHR Support Library	Main OE/RR support library

86.15 Internal Relations

None

86.16 Archiving and Purging

There are no archiving or purging requirements within this software.

86.17 Components

This component supports the following properties and methods:

86.17.1 Execute

Scope: public

Parameter	Datatype	Description
Report	String	OE/RR Report ID^Health Summary Type^Dialog Caption

Displays the eRx Refill Request form.

86.17.2 ViewMailbox

Scope: public

Parameter	Datatype	Description
Report	String	OE/RR Report ID^Health Summary Type^Dialog Caption

Displays the Surescripts Mailbox dialog.

86.18 Properties

None.

87.0 eRx QueueView

87.1 Introduction

The eRx QueueView component is a button allowing the user access to the Surescripts Renewal Request processing dialog.

87.2 Architecture and Business Process Overview

Figure 87-1: Architecture and business process overview

87.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHERXQUEUEVIEW.VIEWQUEUE

Entity	Value
Class Identifier	{9EABE0DB-B058-4160-B19A-4CF81490EE73}
Image File	BEHeRxQueueView.ocx
Property Initializations	none
Serializable Properties	CAPTIONCOLOR1=COLOR, CAPTIONCOLOR2=COLOR, CAPTIONTEXT=TEXT, DISPLAYCOUNT=BOOL, REPORT=TEXT
Required Files	none
Security Keys	none
Multiple Instances Allowed	no
Internal Property Editor	no
All Keys Required	no
Hidden from Property Editor	no
Side-by-Side Versioning	yes
Service	no
.Net Component	no
Associated Build	BEHO*1.1*062001

There are no specific implementation or maintenance tasks associated with this component.

87.4 Routine Descriptions

None.

87.5 File List

None.

87.6 Cross References

None.

87.7 Exported Options

None.

87.8 Exported Security Keys

None.

87.9 Exported Protocols

None.

87.10 Exported Parameters

None.

87.11 Exported Mail Groups

None.

87.12 Callable Routines

None.

87.13 External Relations

Entity	Name	Description
Package	Order Entry/Results Reporting v3.0	Uses supported APIs for orders
Package	IHS Pharmacy Modifications v7	Uses supported APIs for Surescripts renewal requests

87.14 Internal Relations

Entity	Name	Description
Component	CPRS Support Library	Uses supported APIs.

87.15 Archiving and Purging

There are no archiving or purging requirements within this software.

87.16 Components

This component supports the following properties and methods:

87.16.1 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
AUTOSIZE	Boolean	RW	If true, the component automatically resizes itself to accommodate its contents.
BORDERSTYLE	Enum	RW	Sets the style of the border surrounding the component. May be one of: 0 = None 1 = Single 2 = Sunken 3 = Raised
CAPTION	String	RW	Sets the text displayed in the title bar. To justify portions of the caption text, use the "\" character to delimit the left-, center-, and right-justified portions of the caption text.
CAPTIONCOLOR1 CAPTIONCOLOR2	Color	RW	Colors to apply to the title bar. If the two colors differ and a gradient style is set, a gradient effect is created. For a standard title bar style, only the first color is applied.
CAPTIONSTYLE	Enum	RW	Sets the caption style. May be one of: 0 = None – No caption (hides title bar) 1 = Title – Standard title bar 2 = Frame – Framed title bar (group box style) 3 = Left – Left gradient title bar 4 = Right – Right gradient title bar 5 = Center – Center gradient title bar
CAPTIONTEXT	String	RW	Text that appears on button
COLOR	Color	RW	Sets the background color of the component.
DISPLAYCOUNT	Bool	RW	Enables/Disables queue counter

Property	Datatype	Access	Description
FONT	Font	RW	Set the default font used by the component. Some elements of a component may override this setting.
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
HELPFILE	String	RW	Sets the name of the help file associated with the component.
LAYOUT	String	RW	Property representing the internal layout of the form.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

88.0 Designated Primary Provider

88.1 Introduction

The Designated Primary Provider component (Figure 88-1) allows users to view, add, edit, and delete primary providers from several different categories for a patient.

Figure 88-1: Designated Primary Provider component

88.2 Architecture and Business Process Overview

Figure 88-2: Architecture and business process overview

88.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHPCP.PCP
Class Identifier	{666CC312-DE18-4D73-A283-3F5279FEF5D5}
Image File	BEHPCP.dll
Property Initializations	
Serializable Properties	
Required Files	BEHPCP.chm
Security Keys	
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	Yes
Service	No
.Net Component	Yes
Associated Build	BEHO*1.1*063001

There are no specific implementation or maintenance tasks associated with this component.

88.4 Routine Descriptions

None.

88.5 File List

None.

88.6 Cross References

None.

88.7 Exported Options

None.

88.8 Exported Security Keys

None.

88.9 Exported Protocols

None.

88.10 Exported Parameters

None.

88.11 Exported Mail Groups

None.

88.12 Callable Routines

88.12.1 RPC: BEHOPTPC GETBDP

Scope: Private

Parameter	Datatype	Description
DFN	Pointer (#2)	Patient IEN
<return value>	String List	BDPRET(category name)=name of provider^ien of provider ^provider class of provider^date updated

Returns a list of a patient's designated providers.

88.12.2 RPC: BEHOPTPC GETCATS

Scope: Private

Parameter	Datatype	Description
<return value>	String List	Data from BDP DESG SPEC PROV CATEGORY file

Returns an array of categories for designated provider.

88.12.3 RPC: BEHOPTPC SETBDP

Scope: Private

Parameter	Datatype	Description
DFN	Pointer (#2)	Patient IEN
TYPE	String	Name of Category
Prov	Pointer (#200)	Provider IEN
<return value>	Boolean	1 if successful 0^error code if not

Stores a new or edits an existing designated provider for a patient

88.13 External Relations

Entity	Name	Description
Package	DESIGNATED PROVIDER MGT SYSTEM (BDP)	Uses Standard API calls to get and store data in this application.

88.14 Internal Relations

Entity	Name	Description

88.15 Archiving and Purging

There are no archiving or purging requirements within this software.

88.16 Components

This component supports the following properties and methods:

88.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

89.0 Level of Intervention (PHN)

89.1 Introduction

The level of intervention module enables nurses to document PHN visits for their patients. They can document goals and nursing interventions and nursing diagnoses.

Figure 89-1: Level of Intervention component

89.2 Architecture and Business Process Overview

Figure 89-2: Architecture and business process overview

89.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHINTERVENTION.INTERVENTION
Class Identifier	{03BB82FD-F7D4-4E08-84D7-754B4806CE21}
Image File	BEHIntervention.dll
Property Initializations	None
Serializable Properties	None
Required Files	BEHIntervention.chm

Entity	Value
Security Keys	None
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Yes
Associated Build	BGO*1.1*17

There are no specific implementation or maintenance tasks associated with this component.

89.4 Routine Descriptions

This component has been assigned the namespace designation of "BGOVPHN." The following routines are distributed:

Routine	Description
BGOVPHN	Used to Get and set PHN data

89.5 File List

None.

89.6 Cross References

None.

89.7 Exported Options

None.

89.8 Exported Security Keys

None.

89.9 Exported Protocols

None.

89.10 Exported Parameters

None.

89.11 Exported Mail Groups

None.

89.12 Callable Routines

89.12.1 RPC: BGOVPHN CHKPRV

Scope: Private

Parameter	Datatype	Description
IEN	Pointer (#9000010.32)	IEN of file entry
USER	Pointer (#200)	IEN of provider
<return value>	Boolean	1 for they can edit this entry or 0^error text

Returns true if user is a CHIEF,MAS or the encounter provider.

89.12.2 RPC: BGOVPHN DEL

Scope: Private

Parameter	Datatype	Description
INP	Pointer (#9000010.32)	IEN of entry
<return value>	Boolean	1 for success or 0^error text

Physically deletes an item in the V PHN file.

89.12.3 RPC: BGOVPHN GET

Scope: Private

Parameter	Datatype	Description
INP	String	INP = Patient IEN ^ Number to return
<return value>	String List	; .RET = Returned as a list of records: ; RET(1)= "D" ^ IEN [2] ^ Visit Date [3] ^ Date Done [4]^ level of intervention [5] ^Type Decision [6]^Facility Name [7] ^Provider IEN [8] ^ Location IEN [9] ^ Visit IEN [10] ^ Visit Category [11] ^ Visit Locked [12] ; RET(2)= "P"^ IEN [2] ^ PSYCH [3] ; RET(3)= "N" ^ IEN[2] ^ NSG DX [3] ; RET(4)= "S" ^ IEN[2] ^ SHORT TERM GOAL [3] ; RET(5)= "L" ^ IEN[2] ^ LONG TERM GOAL [3]

Returns an array of V PHN for the selected patient.

89.12.4 RPC: BGOVPHN SET

Scope: Private

Parameter	Datatype	Description
INP	Array	; INP(1) = "D" ^ V IEN (if edit) [2] ^Level [3] ^ Type [4] ^ Patient IEN [5] ^ Visit IEN [6] ^ Provider IEN [7] ^Event Date [8] ^ Location IEN [9] ^ Other Location [10]^ Historical Flag [11] ; INP(2)= "P" ^ PSYCH ; INP(3)= "N" ^ NSG DX ; INP(4)= "S" ^ SHORT TERM GOAL ; INP(5)= "L" ^ LONG TERM GOAL
<return value>	String	IEN of entry or an error message

Creates a new entry in the V PHN or edits an existing entry.

89.13 External Relations

None.

89.14 Internal Relations

None.

89.15 Archiving and Purging

There are no archiving or purging requirements within this software.

89.16 Components

This component supports the following properties and methods:

89.17 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

90.0 Direct Mail Button

90.1 Introduction

The Direct Mail button exposes the user dialog and communication layer for sending secure emails.

90.2 Architecture and Business Process Overview

Figure 90-1: Architecture and business process overview

90.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHDIRECT.MAILBUTTON
Class Identifier	{2DFDE9CB-0D04-4C94-A368-CE6D511FF331}
Image File	BEHDirectButton.dll
Property Initializations	none
Serializable Properties	none
Required Files	BEHDirectButton.chm
Security Keys	none
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Yes
Associated Build	BEHO*1.1*069001

There are no specific implementation or maintenance tasks associated with this component.

90.4 Routine Descriptions

This component has been assigned the namespace designation of "BEHODMA." The following routines are distributed:

Routine	Description
BEHODMA	Support routine – Calls the PHR API of BPHRMUPM

90.5 File List

None.

90.6 Cross References

None.

90.7 Exported Options

None.

90.8 Exported Security Keys

None.

90.9 Exported Protocols

None.

90.10 Exported Parameters

Parameter	Instance Type	Value Type	Precedence	Description
BEHODMA BALLOON TIME		Numeric	System, Division, User	The balloon display time in seconds (0-60)
BEHODMA SILENT MODE		Yes/No	System, Division, User	Disable the sent message sound
BEHODMA SUBJECT TEXT		Text	System, Division, User	The Default subject text (1 – 100)

90.11 Exported Mail Groups

None.

90.12 Callable Routines

This section describes supported entry points for routines exported with this component.

90.12.1 RPC: BEHODMA PTEMADR

Scope: public

Parameter	Datatype	Description
DFN	Pointer (#2)	IEN of file entry

90.13 External Relations

Entity	Name	Description
Library	BEHDirectService.dll	DIRECT mail service library. Contains UI and communication layers.

90.14 Internal Relations

None

90.15 Archiving and Purging

There are no archiving or purging requirements within this software.

90.16 Components

This component supports the following properties and methods:

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

91.0 EPCS Credentialing

91.1 Introduction

The EPCS Credentialing component supports a two-step process authorizing a user profile for medication ordering of scheduled medications.

91.2 Architecture and Business Process Overview

Figure 91-1: Architecture and business process overview

91.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHEPCS.CREDENTIALING

Entity	Value
Class Identifier	{2DFDE9CB-0D04-4C94-A368-CE6D511FF331}
Image File	BEHEPCSCredentialing.dll
Property Initializations	none
Serializable Properties	none
Required Files	BEHEPCSCredentialing.chm
Security Keys	none
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	No
Side-by-Side Versioning	No
Service	No
.Net Component	Yes
Associated Build	BEHO*1.1*070001

There are no specific implementation or maintenance tasks associated with this component.

91.4 Routine Descriptions

This component has been assigned the namespace designation of "BEHOEP." The following routines are distributed:

Routine	Description
BEHOEP1	Supporting RPCs for the credentialing component. RPCs for the creation and reading of the provider profiles and logical access requests
BEHOEP2	RPCs for the verification or deletion of pending provider profile. Supporting APIs for the profile hash, profile input transforms and modifications in provider profile and logical access requests i.e. ACTIVATED/REVOKED or other profile modifications
BEHOEP3	Common - BUSA, hash creation, hash verification. Supporting APIs for the credentialing RPMS reports
BEHOEP4	Credentialing RPMS Audit and Hash Mismatch Reports
BEHOEP5	Supporting APIs for hash
BEHOEP6	Credentialing and 2FA Common APIs
BEHOEP7	Certificates validations and checks
BEHOEPAD	Controlled Substance Ad Hoc Report
BEHOEPIC	EPCS Audit Summary Report
BEHOEPIP	Installation support
BEHOEPR1	EPCS and Pharmacy Audit Reports

Routine	Description
BEHOEPR2	EPCS Audit Summary Report
BEHOEPR3	EPCS Audit Summary Report
BEHOEPR4	EPCS Audit Summary Report
BEHOEPR5	EPCS Audit Summary Report
BEHOEPUT	EPCS related utilities

91.5 File List

The component delivers the following FileMan file:

91.5.1 BEH EPCS OE/RR PARAMETERS DATA (#90460.09)

Field Name	#	Datatype	Indexes	Description
NAME	.01	Pointer	B – Standard	Pointer to the NEW PERSON (#200) File.
EDITED BY	.02	Pointer		Pointer to the NEW PERSON (#200) File.
FACILITY	.03	Pointer		Pointer to the INSTITUTION (#4) File.
ENABLED USER	.04	Set of Codes 0:NO 1:YES		Indicates if the user has been enabled or disabled.
DATE/TIME EDITED	.05	Date/Time		Captures last edited date/time.
VERIFIED BY	.06	Pointer		Pointer to the NEW PERSON (#200) File.
DATE/TIME VERIFIED	.07	Date/Time		Captures verified dated/time.

91.5.2 BEHO EPCS INCIDENT REPORT VARIABLES (#90460.13)

Field Name	#	Datatype	Indexes	Description
VARIABLE LONG NAME	.01	Free Text	B – Standard	
VARIABLE SHORT NAME	.02	Free Text		
USER	.03	Pointer		Pointer to the NEW PERSON (#200) File
INCIDENT REPORT VARIABLE	.04	Set of Codes E:EPCS Daily Report P:Pharmacy Daily report		

Field Name	#	Datatype	Indexes	Description
DEFAULT VALUE	.05	Numeric		
SITE VALUE	.06	Numeric		
SITE VALUE TYPE	.07	Set of Codes I:COUNT P:PERCENTAGE		Is the SITE VALUE variable a count (integer) threshold or a percentage threshold.
DESCRIPTION	.08	Free Text		

91.5.1 BEH EPCS AUDIT LOG MESSAGES (#90460.14)

Field Name	#	Datatype	Indexes	Description
LOG ID	.01	Free Text	B – Standard	Assigned Log ID (i.e. EPCS31). Also, it saves as piece 7 of DESC parameter in the BUSA log.
AUDIT MESSAGE	1	Free Text		BUSA log and Mailman message text. Credentialing component, Two Factor Authentication service and Monitoring Services are sending audit related Identifiers and additional information in <INFO1>, <INFO2> and <INFO3>.
STATUS	2	Set of Codes S:SUCCESS F:FAILURE		Holds piece 3 for the DESC parameter of the BUSA log.
ACTION	3	Set of Codes A:AUDIT M:MAILMAN B:BOTH		Indicates if log ID will be used for audit, MailMan or both.
TYPE	4	Set of Codes S:Service PP:Provider Profile P:Pharmacy X:Prescribing E:EPCS		Holds piece 2 for the DESC parameter of the BUSA log.
EVENT	5	Set of Codes E:EPCS P:PHARMACY EP:BOTH		Holds piece 6 for the DESC parameter of the BUSA log.
MAIL GROUP	6	Pointer		Pointer to the MAIL GROUP (3.8) file.
BUSA AUDIT TYPE	7	Free Text		Holds TYPE parameter of the BUSA audit log, i.e. R:RPC Call W:Web Service A:API Call O:Other

Field Name	#	Datatype	Indexes	Description
BUSA AUDIT CATEGORY	8	Free Text		Holds CATEGORY parameter of the BUSA audit log i.e. O: Other Event
BUSA AUDIT ACTION	9	Free Text		Holds ACTION parameter of the BUSA audit log i.e. A:Additions D:Deletions Q:Queries E:Changes C:Copy
BUSA AUDIT CALL	10	Free Text		Holds associated BUSA Group originated the audit.
MESSAGE DESCRIPTION	11	Free Text		Short description for the audit message.

91.6 Cross References

None.

91.7 Exported Options

Name	Description
BEHO EPCS AUDIT LOG SUMMARY	EPCS Audit Log Summary
BEHO EPCS AUDIT REPORT	Controlled substance audit log report
BEHO EPCS AUDIT REPORTS	EPCS Audit Reports
BEHO EPCS AUDIT SUMMARY REPORT	Run the daily EPCS Audit Summary Report
BEHO EPCS CONFIGURATION	EPCS Configuration Menu
BEHO EPCS DIG SIG RX EXPORT	Digitally Signed Rxs Export
BEHO EPCS EXPORT MENU	EPCS Export Reports
BEHO EPCS INCIDENT VARS	EPCS Incident Report Variable Edit
BEHO EPCS MAIN	IHS EPCS Main Menu
BEHO EPCS OR VALID REPORTS	EPCS Validation Reports
BEHO EPCS ORDER EXPORT	Export Orders for Provider
BEHO EPCS ORDER REPORT	Controlled Substance Order Report
BEHO EPCS PHARMACY AUDIT LOG	EPCS Pharmacy Audit Log Summary
BEHO EPCS PHARMACY REPORTS	EPCS Pharmacy Reports
BEHO EPCS PROFILE INTEGRTY RPT	EPCS Integrity Check Report
BEHO EPCS PROV CONFIG CHECK	Check Provider EPCS Configuration
BEHO EPCS PROVIDER AUDIT RPT	Ad hoc Provider Audit Report
BEHO EPCS PROVIDER PROFILE RPT	Provider Profile Integrity Report
BEHO EPCS PRV AD HOC EXPORT	EPCS Ad hoc Provider Order Export

Name	Description
BEHO EPCS SITE CONFIGURATION	EPCS Site Configuration

91.8 Exported Security Keys

Name	Description
BEHOZEPCSREPORT	This key will allow access on credentialing and audit reports

91.9 Exported Protocols

None.

91.10 Exported Parameters

Name	Description
BEHO EPCS CERT NAME THRESHOLD	Holds the number of character difference based on the Levenshtein edit distance algorithm.

91.11 Exported Remote Procedures

Name	Description
BEHOEP1 ENTRYEP	RPC creates pending profile entry.
BEHOEP1 GETPUBKY	RPC returns user public certificate
BEHOEP1 LDPNDNGV	RPC returns all pending profiles.
BEHOEP1 LOACREAD	RPC reads EPCS status of provider.
BEHOEP1 PROV	RPC reads pending provider profile.
BEHOEP1 READP200	RPC reads provider profile.
BEHOEP2 DELPNDVF	RPC deletes the selected pending profile.
BEHOEP2 ENTRYLA	RPC creates pending EPCS status for a provider.
BEHOEP2 INPTRANS	RPC reads Input transforms for VA# and DEA#.
BEHOEP2 LDPNDGLA	RPC reads pending EPCS status of a provider.
BEHOEP2 PENDPROF	RPC returns pending profile for a provider.
BEHOEP2 PROVPRFV	RPC verify pending profile.
BEHOEP3 AUDTEVTS	RPC for audit events.
BEHOEP5 ADDCHK	RPC check addresses for providers/patients.
BEHOEP5 VRFYPHSH	RPC returns current status of hash.

91.12 Exported Mail Groups

Name	Description
BEHO EPCS INCIDENT RESPONSE	People responsible for taking action on incidents. Also, for taking action on: - Provider credentials and any tampering that occurs - Order and Pharmacy Certificates - Server Time Sync

91.13 Callable Routines

91.13.1 $$VRFYPHSH^BEHOEP3(.INP,PROVIEN)

Description: Returns current status of provider profile hash.

Input: Provider IEN

Output: 0 or 1

91.13.2 RPC: BEHOEP1 ENTRYEP

Scope: Private

Parameter	Datatype	Description
DATA	String	DATA= Provider IEN^DUZ^Field Number^Old Data^New Data “~” separated strings.
<return value>	Boolean	Creates a pending profile request for a provider profile. Returns 1 or 0 based on Success/Failure.

91.13.3 RPC: BEHOEP1 GETPUBKY

Scope: Private

Parameter	Datatype	Description
SERIAL	String	The serial number of the certificate to look up.
<return value>	String	Returns the public certificate

91.13.4 RPC: BEHOEP1 LDPNDNGV

Scope: Private

Parameter	Datatype	Description
<return value>	Array	Read all pending profile or pending ACTIVATED/REVOKED requests. Array of strings with the following structure: PROVIDER IEN^PROVIDER NAME^MODIFIED BY IEN^MODIFIED

91.13.5 RPC: BEHOEP1 LOACREAD

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	String	Read current status i.e. ACTIVATED/REVOKED for a provider in EPCS. String with the following structure: Facility IEN^Facility Name^Facility DEA^ENABLED EPCS 0/1

91.13.6 RPC: BEHOEP1 PROV

Scope: Private

Parameter	Datatype	Description
Provider Search Text	String	Provider Name; minimum is three characters.
<return value>	Array	Search providers starting search text and holding ORES security key. Array of strings with the following structure: provider id (DUZ)^provider name

91.13.7 RPC: BEHOEP1 READP200

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	String	Reads current provider profile from NEW PERSON file. String with the following structure of fields from the NEW PERSON (#200) file: 53.1^53.2^ 53.3^53.11^55.1^55.2^55.3^55.4^55.5^55.6^501.2^747.44

91.13.8 RPC: BEHOEP2 DELPNDVF

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	String	Delete both pending profile and logical access request and returns a 1 for success or 0^Message for the failure.

91.13.9 RPC: BEHOEP2 ENTRYLA

Scope: Private

Parameter	Datatype	Description
INPUT	String	INPUT= Provider IEN^DUZ^Facility IEN^ENABLE USER (YES/NO)
<return value>	String	Create logical access requests i.e. ACTIVATED/REVOKED. Returns 1 for success or 0^Message for the failure.

91.13.10 RPC: BEHOEP2 INPTRANS

Scope: Private

Parameter	Datatype	Description
INPUT	String	INPUT IEN^DEA^VA
<return value>	String	DEA#, VA# fields validation returns a 1 or 0^Failed Validation

91.13.11 RPC: BEHOEP2 LDPNDGLA

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	String	Provider pending logical access request. Returns a string with the structure of: Facility IEN^Facility Name^ ^ACTIVATED /REVOKED^MODIFIED BY Name ^DATE/TIME MODIFIED

91.13.12 RPC: BEHOEP2 PENDPROF

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	Array	Returns Pending Provider Profile in an array of strings with the following structure: ProviderIEN^ModifiedByName^FieldNumber^OldData^NewData^DateModified^ModifiedByIEN

91.13.13 RPC: BEHOEP2 PROVPRFV

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	Array	Pending Profile Verification saves the profile to NEW PERSON and pending logical access to OE/RR EPCS PARAMETERS (100.7). Also, mark request as Verified, perform BUSA Logging and Profile hash.

91.13.14 RPC: BEHOEP3 AUDTEVTS

Scope: Private

Parameter	Datatype	Description
INPUT	String	LOGID is mandatory and all other parameters are optional. INPUT= LOGID^Message1^Message 2^Message 3
<return value>	String	Returns a 1 for the successful log or 0 otherwise.

91.13.15 RPC: BEHOEP5 ADDCHK

Scope: Private

Parameter	Datatype	Description
DFN	Pointer	Pointer to PATIENT (#2) file
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
Flag	String	Null-check both addresses 1-check patient address 2-check provider address
<return value>	String	Returns a 1 for a match, 0 for no match

91.13.16 RPC: BEHOEP5 VRFYPHSH

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
<return value>	String	Returns a 1 or 0 indicating if the provider hash has been tampered with.

91.14 External Relations

Entity	Name	Description
Library	BEH2FA	Two Factor Authentication Service

91.15 Internal Relations

None

91.16 Archiving and Purging

There are no archiving or purging requirements within this software.

91.17 Components

This component supports the following properties and methods:

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

92.0 Two-Factor Authentication Service

92.1 Introduction

The service is an intermediary between the user and the external authentication system. The service prompts the user to select certificate from cryptographic token and validates against the two-factor authentication provider.

92.2 Architecture and Business Process Overview

Figure 92-1: Architecture and business process overview

92.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEH2FA.AUTHSVC
Class Identifier	{0516BAC0-D073-40A2-8034-B3DABE7A2DC8}
Image File	BEH2FA.dll
Property Initializations	None
Serializable Properties	None
Required Files	None

Entity	Value
Security Keys	None
Multiple Instances Allowed	No
Internal Property Editor	No
All Keys Required	No
Hidden from Property Editor	Yes
Side-by-Side Versioning	No
Service	Yes
.Net Component	Yes
Associated Build	BEHO*1.1*071001

There are no specific implementation or maintenance tasks associated with this component.

92.4 Routine Descriptions

This component has been assigned the namespace designation of "BEHOEP." The following routines are distributed:

Routine	Description
BEHOEPS	Supports digital signature/hash creation for Order and Pharmacy and Pharmacy hash comparison during medication request processing.
BEHOEP7	Cryptographic certificates validations and checks.

92.5 File List

The service delivers the following FileMan files:

92.5.1 BEH EPCS CERTIFICATE STATUS (#90460.12)

Field Name	#	Datatype	Indexes	Description
SERIAL NUMBER	.01	Free Text	B – Standard	Certificate Serial Number
VERIFIED STATUS	.02	Set Of Codes A:ACTIVE P:PROPOSED R:RETIREDD		
PROVIDER	.03	Pointer to NEW PERSON (#200) file.		
STATUS	.04	Set Of Codes R: REVOKED V: VALID		

Field Name	#	Datatype	Indexes	Description
LAST CHECKED DATE/TIME	.05	Date/Time		
EXPIRATION DATE	.06	Date/Time		
SECURITY HASH	.07	Free Text		
CRL DISTRIBUTION POINT	1	Multiple		
PRIORITY	.01	Numeric		
CRL DISTRIBUTION POINT	.02	Pointer to BEH EPCS CRL DISTRIBUTION POINTS (#90460.15) file.		
USER PUBLIC CERT	2	Word Processing		
CERTIFICATE NAME	4.01	Free Text		
CERTIFICATE USER	4.02	Free Text		

92.5.2 BEH EPCS CRL DISTRIBUTION POINTS (#90460.15)

Field Name	#	Datatype	Indexes	Description
CRL DISTRIBUTION POINT	.01	Free Text	B - Standard	

92.6 Cross References

None.

92.7 Exported Options

None.

92.8 Exported Security Keys

None.

92.9 Exported Protocols

None.

92.10 Exported Parameters

None.

92.11 Exported Remote Procedures

Name	Description
BEHOEP7 AUDITSVC	Audit log events
BEHOEP7 BUSASVC	BUSA Log support
BEHOEP7 CHKCRTST	RPC will return current status of the Certificate.
BEHOEP7 GETCERT	Retrieve certificate registered to provider
BEHOEP7 KEYHLDRS	Return holders of security key
BEHOEP7 LISTCERT	Retrieve certificates for provider
BEHOEP7 SETCERT	Create/Update provider certificate
BEHOEP7 UTC	Returns the current local time of the server in UTC.
BEHOEP7 GETCERT	RPC retrieve the certificate registered to the provider.
BEHOEPS GORDIDIG	Returns information needed for creation of digital signature
BEHOEPS STORDSIG	Stores digital signature for the order.

92.12 Exported Mail Groups

None.

92.13 Callable Routines

92.13.1 EN^BEHOEPS

Scope: private

Parameter	Datatype	Description
ACTION	String	Two-character flag defined by OE/RR
DFN	Numeric	Number representing the IEN to the PATIENT (#2) File
ORNP	Numeric	Number representing the IEN to the NEW PERSON (#200) File
ORIFN	Numeric	Number representing the IEN to the ORDER (#100) File
OETID	Numeric	Number representing the Order Action entry in the ORDER (#100) File

Generates and stores a digital certificate and hash for the order.

92.13.2 EN1^BEHOEPS

Scope: private

Parameter	Datatype	Description
POIEN	Numeric	Number representing the IEN to the PENDING OUTPATIENT ORDERS (#52.41) File

Generates and stores a digital certificate and hash for the pending medication order.

92.13.3 RXVER^BEHOEPS

Scope: private

Parameter	Datatype	Description
POIEN	Numeric	Number representing the IEN to the PENDING OUTPATIENT ORDERS (#52.41) File

Generates a digital certificate and hash and compares to the stored hash for the associated entry in the APSP DEA ARCHIVE INFO (#9009036.1) File

92.13.4 RPC: BEHOEP7 AUDITSVC

Scope: Private

Parameter	Datatype	Description
LOGID	String	LOGID is mandatory and all other parameters are optional.
P1	String	Optional
P2	String	Optional
P3	String	Optional
<return value>	String	Returns a 1 for the successful log or 0 otherwise.

Loads audit log configurations.

92.13.5 RPC: BEHOEP7 BUSASVC

Scope: Private

Parameter	Datatype	Description
ACTION	String	
CALL	String	
MSG	String	
STATUS	String	
EVENT	String	

Parameter	Datatype	Description
<return value>	String	Failure: 0 Success: 1

92.13.6 RPC: BEHOEP7 CERTSTAT

Scope: Private

Parameter	Datatype	Description
SERIAL	String	Serial number
STATUS	Set	R:Revoked V:Valid
<return value>	String	Failure: 0^Error Message Success: 1

92.13.7 RPC: BEHOEP7 CHKCRTST

Scope: Private

Parameter	Datatype	Description
SERIAL	String	Serial Number
<return value>	String	Failure: -n^Error Message Success: 1^Success

92.13.8 RPC: BEHOEP7 GETCERT

Scope: Private

Parameter	Datatype	Description
Provider IEN	Pointer	Pointer to NEW PERSON (#200) file
Flag	String	1 – Return proposed value, Null – Return current value
<return value>	String	Serial Number^Thumbprint (blank for proposed)^Expiration Date

92.13.9 RPC: BEHOEP7 KEYHLDRS

Scope: Private

Parameter	Datatype	Description
KEY	String	Security Key
<return value>	Array	List of users holding the security key formatted as: IEN^Name

92.13.10 RPC: BEHOEP7 LISTCERT

Scope: Private

Parameter	Datatype	Description
<return value>	Array	Returns array of certificates on system for a user.

92.13.11 RPC: BEHOEP7 SETCERT

Scope: Private

Parameter	Datatype	Description
PVIEN	Integer	Provider IEN (File #200)
SERIAL	String	Serial Number
CRL	String	Web address to Certificate Revocation List
EDATE	Date/Time	Expiration Date
CNAME	String	Certificate Name
CISSUER	String	Company issuing certificate
PCERT	String	Public certificate
<return value>	String	Failure: 0^Error Message Success: 1

92.13.12 RPC: BEHOEP7 UTC

Scope: Private

Parameter	Datatype	Description
<return value>	String	Returns local server time in UTC format.

92.13.13 RPC: BEHOEPS GORDIDIG

Scope: Private

Parameter	Datatype	Description
DFN	Integer	IEN to Patient File
ORNP	String	IEN to the New Person file for the ordering provider
ORIFN	Integer	IEN to Order File representing the order
<return value>	String	Returns a string of data to generate a digital signature.

92.13.14 RPC: BEHOEPS STORDSIG

Scope: Private

Parameter	Datatype	Description
PAT	Integer	IEN to Patient File
PRV	Integer	IEN to New Person File
ORD	Integer	IEN to Order File
INPUT	String	Digital signature to be stored^Hash of digital signature
<return value>	String	Failure: -1^Error Message Success: 1

92.14 External Relations

Entity	Name	Description
Library	Cryptographic Token	Commercial Authentication Hardware.

92.15 Internal Relations

None.

92.16 Archiving and Purging

There are no archiving or purging requirements within this software.

92.17 Components

None.

92.18 Templates

None.

93.0 Surescripts Mailbox

93.1 Introduction

The Surescripts Mailbox component is a button allowing the user access to Surescripts related orders and requests and the ability to act on pending requests.

93.2 Architecture and Business Process Overview

Figure 93-1: Architecture and business process overview

93.3 Implementation and Maintenance

This component has the following configuration:

Entity	Value
Programmatic Identifier	BEHERXSSMAILBOX.VIEWMAILBOX
Class Identifier	{33F94CEC-EFCB-4F20-8E6F-1C7130CB91AC}
Image File	BEHeRxSSMailbox.ocx
Property Initializations	none

Entity	Value
Serializable Properties	CAPTIONCOLOR1=COLOR, CAPTIONCOLOR2=COLOR, CAPTIONTEXT=TEXT
Required Files	none
Security Keys	none
Multiple Instances Allowed	no
Internal Property Editor	no
All Keys Required	no
Hidden from Property Editor	no
Side-by-Side Versioning	yes
Service	no
.Net Component	no
Associated Build	BEHO*1.1*072001

There are no specific implementation or maintenance tasks associated with this component.

93.4 Routine Descriptions

None.

93.5 File List

None.

93.6 Cross References

None.

93.7 Exported Options

None.

93.8 Exported Security Keys

None.

93.9 Exported Protocols

None.

93.10 Exported Parameters

None.

93.11 Exported Mail Groups

None.

93.12 Callable Routines

93.12.1 RPC: APSPESM DENIED

Scope: Private

Parameter	Datatype	Description
PRV	Number	NEW PERSON (#200) pointer. Required
STATUS	String	Optional. Defaults to '01'
DAYS	Number	Number of days to go back. Defaults to 30
<return value>	String	Returns a list of requests

Returns a list of denied requests.

93.12.2 RPC: APSPESM GETITM

Scope: Private

Parameter	Datatype	Description
IEN	Number	APSP SURESCRIPTS REQUEST (#9009033.91) pointer. Required
<return value>	String	Returns a string containing details of the related Surescripts request.

Returns information for the requested entry.

93.12.3 RPC: APSPESM ORDERS

Scope: Private

Parameter	Datatype	Description
PRV	Number	NEW PERSON (#200) pointer. Required
DAYS	Number	Number of days back. Defaults to 30
<return value>	String	Returns a list of orders from the PHARMACY EXTERNAL INTERFACE (#52.51) file match criteria.

Returns a string of data for each entry matching the criteria.

93.12.4 RPC: APSPESM REQUESTS

Scope: Private

Parameter	Datatype	Description
PRV	Number	NEW PERSON (#200) pointer. Required
STATUS	String	A string containing one or more status types. Defaults to 1 (processing)
<return value>	String	Returns a list of requests matching the criteria.

Returns a list of requests.

93.12.5 RPC: APSPESM SSMBCNT

Scope: Private

Parameter	Datatype	Description
PRV	Number	NEW PERSON (#200) pointer. Required
<return value>	String	# of Renewal Requests^# of Change Requests^# of Verify Requests^Current patient has requests(0/1)

Returns a string of data used by button to relay request information to the user.

93.12.6 RPC: APSPESM1 RPTRPC

Scope: Private

Parameter	Datatype	Description
PRV	Number	NEW PERSON (#200) pointer.
INP	String	Contains: 'F'<PrescriberAgent Flag: P>Detail Level:' Dn' (where n represents the slider position)
SDT	Date	Start Date
EDT	Date	End Date
<return value>	String	Returns a string of XML formatted information

Returns XML content containing results of the search.

93.13 External Relations

Entity	Name	Description
Package	Order Entry/Results Reporting v3.0	Uses supported APIs for orders
Package	IHS Pharmacy Modifications v7	Uses supported APIs for Surescripts requests
Package	Outpatient Pharmacy v7	References file Pharmacy External Interface (#52.51) File

93.14 Internal Relations

Entity	Name	Description
Component	CPRS Support Library	Uses supported APIs.

93.15 Archiving and Purging

There are no archiving or purging requirements within this software.

93.16 Components

This component supports the following properties and methods:

93.16.1 Properties

Property	Datatype	Access	Description
ALIGN	Enum	RW	Sets the alignment of the component relative to its parent. One of: 0 = None – no alignment occurs 1 = Top – aligns to the top boundary of the parent 2 = Bottom – aligns to the bottom boundary of the parent 3 = Left – aligns to the left boundary of the parent 4 = Right – aligns to the right boundary of the parent 5 = All – expands to the dimensions of the parent 6 = Center – centers itself within the parent
ANCHORS	Flag	RW	Anchors the component's position relative to its parent. Zero or more of: 1 = Top 2 = Left 4 = Right 8 = Bottom
AUTOSIZE	Boolean	RW	If true, the component automatically resizes itself to accommodate its contents.
BORDERSTYLE	Enum	RW	Sets the style of the border surrounding the component. May be one of: 0 = None 1 = Single 2 = Sunken 3 = Raised
CAPTION	String	RW	Sets the text displayed in the title bar. To justify portions of the caption text, use the "\" character to delimit the left-, center-, and right-justified portions of the caption text.

Property	Datatype	Access	Description
CAPTIONCOLOR1 CAPTIONCOLOR2	Color	RW	Colors to apply to the title bar. If the two colors differ and a gradient style is set, a gradient effect is created. For a standard title bar style, only the first color is applied.
CAPTIONSTYLE	Enum	RW	Sets the caption style. May be one of: 0 = None – No caption (hides title bar) 1 = Title – Standard title bar 2 = Frame – Framed title bar (group box style) 3 = Left – Left gradient title bar 4 = Right – Right gradient title bar 5 = Center – Center gradient title bar
CAPTIONTEXT	String	RW	Text that appears on button
COLOR	Color	RW	Sets the background color of the component.
DISPLAYCOUNT	Bool	RW	Enables/Disables queue counter
FONT	Font	RW	Set the default font used by the component. Some elements of a component may override this setting.
HEIGHT	Integer	RW	Sets the height (in pixels) of the component.
HELPFILE	String	RW	Sets the name of the help file associated with the component.
LAYOUT	String	RW	Property representing the internal layout of the form.
LEFT	Integer	RW	Sets the position (in pixels) of the left boundary of the component.
TOP	Integer	RW	Sets the position (in pixels) of the top boundary of the component.
WIDTH	Integer	RW	Sets the width (in pixels) of the top boundary of the component.

Appendix A: System Requirements

A.1 Minimum System Requirements

A.1.1 Windows – RPMS server

- Windows 2008 R2
- Minimum 40 GB RAM
- 2 x 1 TB HDD (over a five year period)
- All Microsoft Updates
- Ensemble 2012.2.5.962.0.13037 (Windows 64 bit version)

A.1.2 AIX – RPMS server

- AIX 7.1
- Minimum 16 GB RAM
- 2 x 1 TB disk space (over a five year period)
- Ensemble 2012.2.5.962.0.13037 (AIX 64 bit version)

A.1.3 Windows – Application server

- Windows 2008 R2
- Minimum 40 GB RAM
- 80+ GB HHD
- All Microsoft Updates
- .Net 4.5.1
- MS Silverlight
- MS Visual C++ 2013 (x86) Redistributable

A.1.4 Client Workstations

- Windows 7
- Minimum 16 GB RAM
- 60 GB HDD
- All Microsoft Updates

- .Net 4.5.1 with .NET 3.5 implemented
- MS Silverlight
- MS Visual C++ 2013 (x86) Redistributable
- Adobe Reader 11.0.04

Appendix B: Developer Tutorial

This tutorial targets software developers wanting to learn the skills and techniques necessary to create components for the VueCentric® Framework. While the document provides examples for the Delphi and Visual Basic user, it should prove to be of use to developers using other programming languages as well. Upon completion of this tutorial, the software developer will have an in-depth understanding of the VueCentric Framework and its major components and how to create visual and non-visual plug-in components.

> **Note:** The examples in this document apply to version 6.0 of Delphi and Visual Basic and Visual Studio .Net 2003. Other versions may differ somewhat from the examples shown.

B.1 Introduction

The first VistA/RPMS applications sported relatively simple, character-based user interfaces. This limited in a significant way the types of user interactions that could be supported. With the advent of client-server programming by way of the remote procedure call broker (aka, RPC Broker), developers were empowered to create much more sophisticated user interfaces supported under the Windows operating system.

However, being of the traditional monolithic design, applications so developed were bulky, integrated poorly with other applications, did little to embrace code reuse, and generally were not very extensible. The VueCentric Framework was developed to address these deficiencies by creating an enabling infrastructure that supports the deployment of applications as reusable components rather than standalone executables. To this end, the Framework provides a number of benefits to the software developer:

- Version-control and Automated Deployment
- User Authentication and Access Control
- Support for Common Tasks
- Debugging Support
- Synchronous and Asynchronous Remote Data Access
- Event Subscription/Propagation
- Choice of Software Development Tools
- Collaborative Software Development Environment
- Context Management
- Visual Design Tool

A component may be either a visual component (ActiveX) or a service (in-process COM server). An example of a visual component might be one the supports management of the problem list. In contrast, a component that maintains information about the currently selected patient would be an example of a service. Note that services may manifest themselves visually (e.g., the patient selection dialog produced by the patient context object), but in their baseline state, they remain invisible to the user. While the initial creation of these two component types differs somewhat, the programming techniques for both are essentially the same.

B.2 Using Debug Mode

Debug mode is activated when the debug command line parameter is specified at application startup. This feature permits debugging of remote procedure calls on the remote host. Normally, when a connection request is made, a dedicated background process is automatically created on the server to handle communications with the client. In contrast, in debug mode when a connection request is made the following Information dialog (Figure B-1) displays.

Figure B-1: Information dialog

1. This dialog instructs you to manually start the listener process on the target machine using the specified entry point. Set any breakpoints needed before doing this if your M environment requires it.
2. At the prompt, type the IP Address (**192.168.1.105**).
3. At the prompt, type the Port (**3335**).
4. Click **OK** to dismiss the dialog. The remainder of the connection process will resume in the usual manner.

Notes: If you close the dialog before manually starting a listener, a background listener will be used instead.

If you use the Serenji debugger, be sure to start the listener after invoking the Serenji shell.

Using debug mode with some network configurations may fail to establish a connection. This is especially true for some VPNs and networks where NAT is utilized. Because debug mode performs a connection callback from the remote host to the client, the remote host must be able to determine the route to the client's IP address. If a VPN is in use, supply the VPN-assigned IP address when starting the listener process rather than the address supplied by the client.

B.3 Using the Trace Log

Trace mode is an extremely useful debugging tool. This mode is activated by starting the VIM with the /trace command line parameter. When activated, a new menu item appears on the system menu called Show Trace Log. Checking this menu item reveals the Trace Log Viewer (see below). The viewer may be used to examine a multitude of activities such as remote procedure calls (synchronous and asynchronous), events, context changes, status messages, as well as custom entries created by running components (see Appendix B.23.7).

The Trace Log is divided into four panes: the trace list, the trace detail, the toolbar, and the status bar. It has the following layout:

Figure B-2: Trace Log

The trace list is located in the upper left. If this pane is not visible, click the minimize bar that separates this from the trace detail pane. The trace list shows a list of all activities logged by the application. Each entry consists of three columns. From left to right these are the **sequence number**, which reflects the chronology of log entries and is incremented as each new entry is logged, the **class**, which represents the category of the entry (e.g., remote procedure call vs. host event), and the **type**, which provides further sub-classification of the entry within its class. The trace list may be sorted by any column by clicking the column header. Clicking the same column header in succession toggles the sort order of that column. Right clicking an entry elicits a pop-up menu that permits further manipulation of the event list.

The upper right pane represents detailed information about the currently selected event. The toolbar permits navigation of the event list and controls other aspects of event logging. The status bar at the bottom of the dialog displays information about the state of the event log.

Figure B-3: Filters menu

The following list describes the actions that may be launched from the menu or the toolbar:

- **Remove Entry**: Removes the selected entry from the event log.
- **Filter this Class**: Removes from the event log all entries belonging to the same class as the selected entry and suppresses logging of future events of the same class.
- **Filter this Type**: Removes from the event log all entries belonging to the same type as the selected entry and suppresses logging of future events of the same type. This affects only event types within the currently selected event class.
- **Clear Filters**: Removes existing filters. Choose the type of filter to remove, or all filters:
  - **Class Filters**
  - **Type Filters**
  - **All Filters**
- Navigation Buttons: Navigate the event log.

Figure B-4: Navigation buttons

  - **First**: Moves to the first entry
  - **Previous**: Moves to the previous entry
  - **Next**: Moves to the next entry
  - **Last**: Moves to the last entry
- **Resume** and **Suspend** buttons:

Figure B-5: Event buttons

Toggle between resuming and suspending event capture.

- **Clear** button: Clears the event log.

Figure B-6: Clear button

- **Hide** button: Hides the event log dialog. This does not affect the capture of event data.

Figure B-7: Hide button

- **Print** button: Prints the contents of the event detail pane.

Figure B-8: Print button

- **Max Entries** field: Sets the maximum number of event log entries. When this limit is reached, the oldest entries are removed.

Figure B-9: Max Entries field

B.4 About Component Support Services

The Component Support Services (CSS) provide the following types of services to the component author:

- Remote Procedure Support (synchronous and asynchronous)
- Event Management
- Context Management
- Dynamic Discovery
- Miscellaneous Utilities

Accessing these services requires acquiring an interface reference to the session object. The session object is shared across all components within the same application process space. It may not be instantiated directly, but instead must be acquired from the server object. The sole purpose of the server object is to create a single instance of the session object and return its reference to the caller. Thus, accessing the session object involves two steps: instantiate the server object and request of it a reference to the session object. Once this is done, the server object is no longer needed and can be released.

The following are examples of how this process is done in Delphi and Visual Basic:

Delphi	Visual Basic

uses CIA_CSS_TLB;
…
var
  vcSession: ICSS_Session;
begin
  vcSession :=
CoCSS_Server.Create.Session;
end;
	Dim vcServer As New
	CIA_CSS.CSS_Server
	Dim vcSession As
	CIA_CSS.CSS_Session
	Set vcSession = vcServer.Session
	Set vcServer = Nothing

Typically, a programmer will acquire the reference to the session object in the component's initializer (the Initialize method in both Delphi and Visual Basic), storing it in an instance variable that persists for the life of the component. This avoids the overhead of repeatedly creating the server object each time the session object needs to be accessed.

A detailed description of the services provided by the CSS and how to use them may be found in the sections that follow.

B.5 About COM and ActiveX

The Component Object Model, or COM, is a Microsoft specification for ensuring interoperability between components regardless of the programming language or development tool used in their creation. Key to this specification is the concept of interfaces.

An interface is a collection of method (i.e., procedures and functions) and property declarations. More than this, an interface represents an immutable contract — a guarantee that any object supporting a particular interface will always implement each and every method or property declared within it. While the implementations may differ, the means for invoking them are identical.

COM builds upon the concept of interfaces by providing a standardized means of invoking interfaces that works within and across process boundaries and adjusts for internal differences in calling conventions and data representation. COM further standardizes on an established set of supported core datatypes (though these can be extended using custom marshaling techniques — a practice that is rarely necessary and very complex).

Also central to the success of COM is the concept of self-description, or the ability to interrogate a component's metadata to determine the interfaces it supports and their declarations. This metadata is provided in the form of a type library. A type library can be supplied separately from the component's executable file, but more typically it is imbedded in that file as a resource. Visual Basic automatically generates type library information for its components based on the program code itself and has no provision for directly accessing the type library itself. Delphi provides a type library editor that may be used to modify an existing type library.

Figure B-10: CSS window

ActiveX, also a Microsoft specification, builds upon the COM specification by defining a specific set of interfaces designed to support the hosting of visual components within container applications. A COM component that supports these interfaces is known as an ActiveX component. In Delphi, wizards exist to create ActiveX components either as wrappers for existing components or as Active Forms, which behave like regular forms in that other components may be placed upon them. Visual Basic has the equivalent of the Active Form in the User Control class. Again, because ActiveX controls are COM objects, it does not matter to the application hosting them what programming language or tool created them.

All components within the VueCentric Framework are COM objects. This includes the VIM, CSS, CMS and all plug-in services and visual components. Further, the visual components are ActiveX components. This means that any visual object created for the VueCentric Framework can be hosted in any ActiveX-compliant container. The VIM is one such container. Internet Explorer is an example of another.

B.6 Component Types

The VueCentric Framework recognizes two basic types of components: **visual components** (ActiveX) and **services** (COM). Visual components may be manipulated in the VIM Designer (services cannot). Services extend the capabilities of the framework and although they may manifest themselves visually, they do not require a container (unlike visual components). The techniques for creating these two component types differ somewhat as does the manner in which they are loaded and executed. However, once created, the coding is very similar for both types. In fact, it is possible to create a visual component and a service within one executable file.

The sections that follow first address programming techniques common to both component types. Following that are sections that describe the procedures specific to a particular component type.

B.7 Component Registration

Component registration has three separate meanings, depending upon the context. They are:

B.7.1 COM Registration

In the COM model, all components must be registered before they may be used. In this sense, information about the component's interfaces and capabilities is stored in the Windows Registry under the HKEY_CLASSES_ROOT key. This registration is typically done by the component itself through a standard published entry point (DllRegisterServer). For Visual Basic and Delphi programmers, this code is automatically generated. When a component is requested from the CSS that has not been previously registered on the target machine, it is retrieved and registered automatically. A tool is also available in Windows called regsvr32.exe (usually located in the system directory) that can be used to manually register and unregister components. Running this tool with no command line options displays information on how it is used. The VueCentric System Management Utility can also be used to register and unregister components. See the online documentation that accompanies this utility for more information.

B.7.2 Framework Registration

Before a component may be utilized within the VueCentric Framework, it must also be registered to the framework. This registration information is stored on the remote host in the VUECENTRIC OBJECT REGISTRY file. This may be done using the VueCentric System Management Utility, or it may be directly entered into the file using VA FileMan. This information determines what objects are available, how they are classified, who can use them, where to get them, and what their capabilities and special requirements are.

B.7.3 Runtime Registration

At execution time under the VueCentric Framework, component registration has yet another meaning. In this context, registration means that the component advertises its presence to the CSS at runtime. The CSS maintains an internal table of all components so registered. It also interrogates the component to determine what context events it handles and connects any handlers it finds to the appropriate context objects.

This registration process is automatic if the component is loaded by the VIM (unless its object registry settings suppress this). An object may also register itself by calling the RegisterObject method, passing a reference to itself. If an object registers itself in this manner, it must also unregister itself prior to unloading by calling the UnregisterObject method. In general, self-registration is only necessary if the object is to be used outside the VIM (e.g., in Internet Explorer). For more information about this topic, see Section B.23.3 (Other Containers).

While the latter form of registration is not required to use an object within the framework, unregistered objects cannot be discovered by other objects nor will they be connected to any context events.

B.8 Naming Conventions

Every COM object has a globally unique identifier (GUID) associated with it and each of its interfaces. While an object can be, and often is, referenced by its GUID, there is a second identifier that is more user friendly: the programmatic identifier (PROGID). The PROGID typically consists of two parts, separated by a period (e.g., “Word.Application”). Occasionally, a third part consisting of a version number is added (the so-called version-dependent PROGID, e.g., “Word.Application.8”). There are no fixed conventions for these identifiers and it should be readily apparent that the risk of naming conflicts is real. In the event that there is a naming conflict, the last object registered is the one referenced. Therefore, some naming convention is needed to minimize this risk.

For Delphi and Visual Basic environments, the PROGID is generated automatically, formed from the type library name (which initially comes from the project name) and the component name. In Delphi, both can later be changed using the ***type library editor*** see COM and ActiveX in the previous section). Thus, the selection of a PROGID is contingent upon your selection of a project name and a component name. We advise beginning your project name with a namespace prefix. We have reserved the prefixes “CIA,” “VC,” and “CW” for our projects. These prefixes refer to framework components, plug-in components, and legacy components, respectively. For example, CIA_VIM.VIM refers to the automation interface for the VIM and vcNotifications.vcNotificationsX refers to the visual notifications component.

PROGIDs are mapped to their GUID equivalents in the Windows Registry under the HKEY_CLASS_ROOT node. For example, if you were to examine the registry key HKEY_CLASSES_ROOT\Word.Application\CLSID, you would find that the default value associated with that key is the GUID corresponding to the object. Knowing this structure makes it easy to understand how two objects with the same PROGID cannot happily coexist on the same machine, even though there GUIDs differ. Remember: it is much harder to change a PROGID after deployment than to give careful thought to naming at the beginning of a project.

B.9 Multiple vs. Single Instancing

Multiple instancing refers to the ability to create more than one copy of a component within the same application space at the same time. For some components, this may clearly not be desirable for a couple of reasons. First, some components that perform a very central function should be limited to a single instance. For example, one would not want to have more than one order-entry component as this would be very confusing to the user. On the other hand, a component that provides a read-only view of the problem list might be useful in several different locations within the user interface. Second, some components are programmatically designed to be singletons (single instance). This would be true if, for example, a component required exclusive use of a shared resource, such as a database file.

In Delphi, the use of global variables (i.e., variables declared outside the scope of a class or method) can cause serious problems because these are shared across multiple instances of the component. A good example of this is the form declarations. The Delphi form wizard automatically generates a global variable to hold the form instance (this is not the case for Active Forms, but does hold true for any additional forms created within an Active Form project). If one instance of a component creates a form and saves the reference in the global form variable, and a second instance does the same, the first instance loses its reference and now refers to the form created by the second. This can result in some very odd behaviors. On the other hand, this can be used to advantage if, for example, the programmer wants to create a shared instance of a form to be used among all instances of the component. Visual Basic, on the other hand, declares all of its variables (even its “global” variables) as instance variables. This eliminates the problem (but also the advantage) of shared variables.

To control whether more than one instance of a component is allowed, set the **Allow Multiple Instances** field in the VUECENTRIC OBJECT REGISTRY file accordingly. If you are uncertain as to a component’s suitability for multiple instancing, disallow multiple instances.

B.10 Remote Procedure Calls

Remote procedure calls are conceptually very simple. Essentially, they permit calling a procedure on a remote host as if it were local. The VueCentric Framework encapsulates the Medsphere Remote Procedure Broker in its Communications Services Layer (CSL). This encapsulation permits all components within the application space to share a single instance of the broker. Interaction with the CSL occurs through a suite of APIs provided by the session object.

Remote procedure creation is very straightforward. First, create and configure the remote procedure on the remote host. Then write the client code to invoke the remote procedure. Remote procedures may be invoked synchronously, where the client pauses until the procedure completes and returns its data, or asynchronously, where control immediately returns to the client after the call and the client is notified when the call has completed through a callback. All of these techniques are described in the sections that follow.

B.10.1 Create the M Routine

First, you must create M routine that contains the code that is to be executed. The entry point must be parameterized, with the first parameter reserved for returning data to the caller. Each subsequent parameter corresponds to a parameter passed by the client application and may be a single scalar value or a local array of values. Up to forty parameters may be specified.

The following example illustrates code implementing a simple remote procedure that accepts a string and a count as input parameters and returns an array of that string duplicated the number of times specified by the count. The return data type is assumed to be a global array (see next section).

Required parameters are missing or incorrect.

B.10.2 Create a Remote Procedure Definition

The next step is to create an entry for the remote procedure in the REMOTE PROCEDURE file. At a minimum, you must define the following fields:

Field	Length
NAME	The name of the remote procedure. This is the name that the client application will use to invoke the remote procedure. The name should be appropriately namespaced and descriptive of its purpose.
TAG	The line label of the entry point in the M code. In the above example, this would be “ENTRY.”
ROUTINE	The name of the routine in which the remote procedure code resides.

Field	Length
RETURN VALUE TYPE	The type of data returned by the remote procedure. This may be one of the following: 1 = Single value, 2 = Local array, 3 = Word processing, 4 = Global array, 5 = Global instance, H = Host file
WORD WRAP ON	Affects only return types 3, 4, and H. If true, a carriage return character is appended to each value in the array or line in the host file. If false, no carriage return is appended.

Though not required, the INPUT PARAMETER multiple should be completed to document the expected parameters and their purpose. The DESCRIPTION field should likewise be completed to document the purpose of the remote procedure.

The RETURN VALUE TYPE merits further elaboration. The Broker daemon passes the return value parameter (the first parameter of the remote procedure entry point) by reference. Therefore, any change to the parameter is returned to the daemon and then to the caller. The simplest return type, single value, returns whatever value is set in the first parameter. Like the single value type, the global instance type returns a single value to the caller. In this case, the return value parameter must be set to a valid global reference. For the local array and word processing return types, all subscripted values are returned to the caller. The global array return type returns all subscripted values beneath a root node specified in the return value parameter. The daemon sets this to a default value prior to calling the remote procedure. The remote procedure may use this value, or assign a different one.

Note: The daemon deletes this global root upon completion.

For all return types that are arrays (global and local), the daemon returns the data to the caller in the collation order of the subscripts (the subscripts themselves are not returned, only the data).

B.10.3 Register the Remote Procedure

Every remote procedure is invoked within an execution context (see the RPCContext property). This context dictates which remote procedures may be executed and rejects any attempts to execute a remote procedure outside the context. Execution contexts are simply entries in the OPTION file of type Broker. To register a remote procedure to an execution context, select the desired entry in the OPTION file (or create a new one with the TYPE field set to Broker), and add the remote procedure to the RPC multiple.

B.10.4 Calling a Remote Procedure

Once the remote procedure has been created and properly registered on the remote host, it may be invoked by the client application using one of a number of method calls provided by the session object. All methods for calling remote procedures begin with “CallRPC” and can be divided into two groups: synchronous and asynchronous, reflecting the two modes in which a remote procedure may be invoked. Synchronous calls do not return until the remote procedure has completed and returned its data. Asynchronous calls return immediately, notifying the caller at a later time when the remote procedure has completed.

Choosing which mode to use depends upon a number of factors:

Requirement	Use
User interaction requires immediate results before application may continue.	Synchronous
Availability of data is not time critical.	Asynchronous
Remote procedure requires considerable time to complete.	Asynchronous
Series of sequenced transactions.	Synchronous

B.10.5 Synchronous Calls

B.10.5.1 RPC Methods

Calling a remote procedure in synchronous mode is very straightforward. There are six method calls that support this mode. They differ only in the type of return data that is expected. They are:

Method Name	Return Type
CallRPCBool	Boolean value
CallRPCDate	Date (Delphi: TDateTime; Visual Basic: Date)
CallRPCInt	32-bit integer
CallRPCList	Multi-valued string list in "comma-text" format. This format encloses each entry in double quotes and separates them with commas.
CallRPCString	String (Delphi: WideString; Visual Basic: BSTR)
CallRPCText	Multi-valued string list in "plain-text" format. This format separates each entry with a <CR><LF> pair.

If the remote procedure return type is multi-valued (either local or global array), you must use either CallRPCList or CallRPCText to retrieve all of the list elements returned. Otherwise, only the first element is returned. The choice between "comma-text" and "plain-text" formats is somewhat arbitrary. If list elements could contain either a <CR> or <LF> character, comma-text format is preferred. Delphi's TStrings class provides a means to convert either of these formats into string lists (the CommaText and Text properties, respectively). Visual Basic requires parsing the result values into a string array.

Regardless of which of the above methods is used, the parameters passed are identical:

Parameter	Datatype	Description
RPCName	String	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the '^' delimiter. To specify a version number, prefix the RPC name with the version number enclosed in vertical bars.
Parameters	Variant	Parameters to be passed to the remote procedure.

B.10.5.2 Specifying the Remote Procedure

The RPCName property specifies the name of the remote procedure to invoke. If the remote procedure needs to be invoked in an execution context other than the default, precede the remote procedure name with the context name and a caret (^) character. If the remote procedure requires a version number, you may include this by prefixing the version number enclosed in vertical bars (|) to the remote procedure name. Consider the following examples:

RPCName Value	Description
CIAV GET PATIENT	Executes the remote procedure named 'CIAV GET PATIENT' in the default framework execution context. No version information is passed.
MYCONTEXT^MYRPC	Executes the remote procedure named 'MYRPC' in the execution context named 'MYCONTEXT.' No version information is passed
|1.5|MYRPC	Executes the remote procedure named 'MYRPC' in the default execution context. The XWBAPVER variable will be set to 1.5.
MYCONTEXT^|2.1|MYRPC	Executes the remote procedure named 'MYRPC' in the execution context named 'MYCONTEXT.' The XWBAPVER variable will be set to 2.1.

B.10.5.3 Specifying the Parameter List

The Parameters property specifies the parameter values to be passed to the remote procedure. These correspond to the second and subsequent parameters of the remote procedure's entry point (recall that the first parameter is reserved for the return value). Because parameter lists typically differ for each remote procedure, the datatype of the Parameters property must accommodate these variations. As a consequence, the datatype of the Parameters property is a variant. The following table describes the techniques for packaging scalar (non-array) parameters in the Parameters property:

# of Parameters	Format
0	Set Parameters to NULL (nothing).
1	Set Parameters to the value of the single parameter.
>1	Set Parameters to an array of variants. Set each element of the array to the value of the corresponding parameter.

B.10.5.4 Specifying Array Parameters

Passing array parameters to your remote procedure requires some additional effort. An array parameter must be passed as an array of variants. If any parameter to be passed is an array type, the Parameters property must be set to an array of variants even if the array parameter is the only parameter. Otherwise, the CSS would not be able to distinguish a parameter list from a single array parameter. Thus, to pass an array parameter, two arrays of variants must be created: one for the parameter list (we will call it the parameter list array) and one for the array parameter (we will call it the array parameter array). Set the reference to the array parameter array into the corresponding entry of the parameter list array. Set each entry of the array parameter array to the values to be passed. By default, array parameters are passed to the remote host with the same subscript values as the original. To override the default subscript value for an entry, prefix the entry with the subscript value (in comma-text format) followed by a character whose ASCII value is 1.

If passing parameters, especially array parameters, sounds confusing, it can be. This process can be greatly simplified by writing helper functions that take parameters in a format native to the programming language used and packages them in the manner described above. CIA has developed a number of Delphi and Visual Basic helper functions for this purpose. See Appendix B.24.

B.10.5.5 Handling Exceptions

If the remote procedure raises an unhandled exception on the remote host, that exception is trapped and raised on the client. Therefore, the caller should be prepared to handle any exceptions that may be raised during the remote procedure invocation.

B.10.6 Asynchronous Calls

Calling a remote procedure in asynchronous mode is not altogether different from doing so in synchronous mode. Naming the remote procedure and packaging the parameter list are identical. However, there is only one method call for invoking a remote procedure in asynchronous mode and its return value is a handle that uniquely identifies the call. So how does the caller know when the remote procedure has completed and how is its data returned? The CSS provides a declaration for a callback interface (ICSS_SessionEvents) that is used to signal all asynchronous activities (including events and asynchronous RPCs). The caller must implement this interface and provide implementations for each of its methods. When the caller invokes a remote procedure asynchronously, it must pass a reference to this callback interface. The session object then invokes methods (callbacks) on this interface to notify the caller when an asynchronous activity has completed.

B.10.6.1 Calling the Remote Procedure Asynchronously

The Session object provides one method, CallRPCAsync, for calling a remote procedure asynchronously. It takes the following parameters:

Parameter	Datatype	Description
RPCName	String	Name of the remote procedure to be invoked. If an execution context other than the default is desired, precede the RPC name with a context name and the caret (^) delimiter. To specify a version number, prefix the RPC name with the version number enclosed in vertical bars.
Parameters	Variant	Parameters to be passed to the remote procedure.
Callback	ICSS_SessionEvents	Callback interface to be invoked on completion of the remote procedure.
PlainText	Boolean	If true, data returned to the callback interface is in plain-text format. Otherwise, format is in comma-text format.
<return value>	Integer	A 32-bit handle that uniquely identifies this asynchronous call.

Note that the first two parameters are identical to those used by the synchronous remote procedure methods. The Callback parameter refers to the callback interface the Session object will use to notify the caller when the remote procedure has completed. PlainText indicates whether the data is to be returned in plain-text or comma-text format. Data returned from asynchronous calls are always formatted as lists in one of these two formats. This is not to imply that only remote procedures that return lists can be called asynchronously. Indeed, a remote procedure that returns, for example, a single Boolean value can be called asynchronously (in fact, any remote procedure may be called asynchronously).

In the case of a remote procedure returning a single Boolean value, the return value would be a list containing a single element and the caller must convert that element from a string to the desired format.

The CallRPCAsync method returns a 32-bit integer handle that uniquely identifies the call. Because a client may have multiple outstanding asynchronous calls, this handle is critical to identify which call is being signaled.

B.10.6.2 Implementing the Callback Interface

The ICSS_SessionEvents interface has the following declaration:

- Delphi

```
 procedure RPCCallback(Handle: Integer; const Data: WideString); safecall;
 procedure RPCCallbackError(Handle: Integer; ErrorCode: Integer; const
ErrorText: WideString); safecall;
 procedure EventCallback(const EventType: WideString; const EventStub:
WideString); safecall;
```

- Visual Basic

```
 Sub ICSS_SessionEvents_RPCCallback(ByVal Handle As Long, ByVal Data As
String)
 Sub ICSS_SessionEvents_RPCCallbackError(ByVal Handle As Long, ByVal
ErrorCode As Long, ByVal ErrorText As String)
 Sub ICSS_SessionEvents_EventCallback(ByVal EventType As String, ByVal
EventStub As String)
```

The first two methods in the ICSS_SessionEvents interface provide support for asynchronous remote procedure calls. The third is used for reporting events and is discussed in the section on ***events***. Note that the client must supply implementations for all three methods, even if only one or two are actually used. This is a requirement for implementing any COM interface. If a method is not needed, its implementation can be left empty (an "NOP" implementation).

The Session object invokes the caller's RPCCallback method when an asynchronous remote procedure has completed successfully. The Handle parameter is the same value returned by the CallRPCAsync method. The Data parameter is the data returned by the remote procedure in either plain-text or comma-text format.

If an asynchronous remote procedure raises an unhandled exception on the remote host, that exception is trapped and the RPCCallbackError method is called instead. This method includes the Handle parameter as expected, but also supplies ErrorCode and ErrorText parameters that provide information about the exception.

B.10.6.3 Aborting a Pending Call

Occasionally, the caller may want to abort an asynchronous call that has not yet completed. This might occur, for example, if a patient context change has occurred and the pending remote procedure no longer applies. To abort an asynchronous remote procedure, call the CallRPCAbort method, passing the handle of the call that is to be aborted.

If the remote procedure has already completed, the request is ignored. If the remote procedure is pending execution, it is removed from the execution queue. If the remote procedure is in progress, it is requested to abort. Even if the remote procedure ignores the abort request, any data it may return is discarded and its completion is not signaled to the caller.

B.11 Context Management

The VueCentric Framework provides support for special services known as context objects. A context object is used to establish a common context for a specific entity, such as patient or user. For context objects to be useful, a mechanism must exist to notify interested parties when the context is about to change. The Framework accomplishes this through the use of a callback interface.

Every context object defines a callback interface that is a descendant of the ICSS_ContextEvents interface. When a context object registers itself as a service with the CSS, the CSS recognizes that it is a context object and enters its callback interface into an internal table. When a component (visual or non-visual) registers itself with the CSS, that component is interrogated to determine if it implements any of the callback interfaces in this table. If it does, the component is registered as a subscriber to the callback. This means that all a component must do to be notified of context changes for a particular context object is to implement the callback interface for that context object. The CSS takes care of the rest.

B.11.1 Callbacks

Every context object defines a callback interface that is a descendant of the ICSS_ContextEvents interface. This interface has the following declaration:

- Delphi

```
function Pending(Silent: WordBool): WideString; safecall;
procedure Canceled; safecall;
procedure Committed; safecall;
```

- Visual Basic

```
Function ICSS_ContextEvents_Pending(ByVal Silent As Boolean) As String
Sub ICSS_ContextEvents_Canceled()
Sub ICSS_ContextEvents_Committed()
```

Context change is a democratic process and occurs in two phases. In the first phase, known as the polling phase, every subscriber to the context change is interrogated for its acceptance of the change. This is done through a call to the Pending method. Subscribers indicate their acquiescence to the proposed change by returning a null value. By returning a non-null value, the subscriber is indicating that it does not want the context change to occur. The Silent parameter indicates whether or not the subscriber is permitted to interact with the user during this decision-making process. If Silent is true, no user interaction is permitted. This will be the case under one of two conditions: the context change was initiated by an external application through the CCOW interface, or the context change was initiated during a forced shutdown of the application.

If all subscribers acquiesce to the proposed context change, the context is changed and each subscriber is notified of the change through a call to the Committed method. If any subscriber rejects the context change, the proposed change is aborted and subscribers are notified through the Canceled method. Note that in the case where a silent context change is occurring, the context change may occur even when a subscriber rejects the change. This will always be the case in a forced shutdown and can also occur in the case of a CCOW-initiated context change if the requester decides to force the change despite the objection. Therefore, a component must be prepared to react to a context change even if it has rejected the request.

B.11.2 Requesting a Context Change

How a context change request is made is dependent upon the individual context object. For example, setting the Handle property of the patient context object initiates a context change request. Depending on whether or not the context change is accepted, the value of the Handle property may or may not be changed. The Select method of the patient context object is another means for changing the patient context. Yet a third method, is through a call to the SETCTX^BEHOPTCX method on the remote host. Thus, there is no “standard” mechanism for initiating a context change request.

B.12 Events

Events are an extremely powerful tool for communicating information. The VueCentric Framework implements events utilizing a subscribe/publish (consumer/producer) model. In this model, clients may subscribe to a particular event. When an event is generated (published), the Framework forwards that event to all subscribers of that type of event. Some important features of the VueCentric event model are:

- Events are fully extensible. New event types may be defined at any time.
- Event notification is asynchronous. Subscribers may be notified at any time that an event has occurred.

- Sequence of event delivery depends upon order of event subscription. Generally one should not rely upon a particular sequence of event delivery.
- Event subscription and publication can be restricted using security keys.
- Event publication can be local (current process) or remote (all active sessions).
- Events are hierarchical. This allows the publishing of events at one level of detail and subscribing of events at another.
- Events may be generated by the client (local or remote) or by the remote host (remote only).
- Distribution of remote events may be restricted to a subset of subscribers (at user or session level).

B.12.1 Defining an Event

Events should be defined by creating an entry in the CIA EVENT TYPE file. While this is technically not necessary in order to use an event, it is strongly encouraged to do so for two reasons. First, this file provides a place to formally document each event and helps avoid naming collisions between events. Second, this file permits applying business rules that can restrict publication and subscription rights.

B.12.2 Firing Events: Local vs. Remote

Events fired locally are distributed only to subscribers within the current session. Events fired remotely are sent to the host system which then distributes them to all subscribing sessions. A receiver of an event handles both types identically.

The Session object provides two methods for firing (signaling) events: EventFireLocal and EventFireRemote, for firing an event to local and remote subscribers, respectively. They are defined elsewhere in this document.

B.12.3 Receiving Events: Callbacks

When an event is fired, the Session object notifies each subscriber through the EventCallback method of the ICSS_SessionEvents callback interface. This callback interface is specified at the time the subscription is established in the call to the EventSubscribe method. This callback interface is the same ICSS_SessionEvents interface described in the section on asynchronous remote procedure calls. Any component wanting to subscribe to an event must implement this interface and provide implementations for all three methods declared within it.

The EventCallback method has the following parameters:

B.12.4 Hierarchical Events

There are times when an event publisher may want to specify an event at one level of granularity, but an event subscriber may want to subscribe at a less granular level. For example, the publisher of a NOTIFY event might want to specify the type of notification (for example, critical lab value vs. order needing signature) as part of the event type. While one subscriber might want to know about only notifications of a particular type, another subscriber might want to know about all NOTIFY events. To reconcile differing event granularity requirements between producers and consumers, the Broker implements a hierarchical schema for event naming and subscription.

Hierarchical events are named using the following convention:

<Event name>.<Event subtype>.<Event sub-subtype>.<…>

The convention uses a period to separate each subtype specifier. Subtypes appear in order of increasing specificity, from left to right. A consumer may subscribe to a hierarchical event at any level of specificity. In the above example, the event producer may publish the following two events:

- NOTIFY.CRITICALVALUE
- NOTIFY.ORDERNEEDSSIG

A consumer caring only about critical value notifications may subscribe to the NOTIFY.CRITICALVALUE event while a second consumer wanting to know about all notification events may subscribe to NOTIFY events. Both subscribers would receive NOTIFY.CRITICALVALUE events, but only the second would receive NOTIFY.ORDERNEEDSSIG events.

When defining hierarchical events in the CIA EVENT TYPE file, one may create an entry for only the top level event type or for each subtype or both. Setting the DISABLE field to YES for an event also disables all events of greater specificity below it in the hierarchy. In the above example, setting the DISABLE field of the NOTIFY entry to YES would disable the NOTIFY.CRITICALVALUE and NOTIFY.ORDERNEEDSSIG events even if those events had separate entries in the file with their DISABLE field set to NO. This inheritance pattern also holds true for security keys associated with events. Thus, it is only necessary to create entries for event subtypes if one desires to apply business rules to those subtypes that differ from the parent entry.

B.13 Creating Visual Components with Delphi

Delphi offers two project types for creating visual components: Active Form and ActiveX Control. The latter type generates a simple wrapper for an existing control and cannot be used for creating more complex controls. Therefore, its use will not be addressed here. The Active Form, on the other hand, allows one to create a form as an ActiveX control. This project type is much more useful.

B.13.1 Creating the Active Form Project

To create an Active Form project, select **File | New | Other...**.This will display the project selection dialog as shown in Figure B-11.

Figure B-11: New Items dialog

Select the **ActiveX** tab, pick the Active Form project type, and click **OK**. The **Active Form Wizard** dialog (Figure B-12) displays.

Figure B-12: Active Form wizard

Change the first three editable entries to the desired names. Note that the New ActiveX Name will become the name of your component (and the second part of the programmatic identifier) and the Project Name will become the name of the type library (and the first part of the programmatic identifier). Changing these later can be done, but can be problematic, so choose wisely here. Leave the **Threading Model** field as Apartment. Select the **Include Version Information** option check box. The other two options are at the component author's discretion, but we will leave them unchecked.

After making these changes, the **Active Form Wizard** dialog now looks like this:

Figure B-13: Active Form wizard

Click **OK** and the wizard will create your project. The project will consist of a single form and its class declaration (class name TDemoControlX) and a type library.

Figure B-14: Created project

The component class declaration already has a number of event handlers and methods defined. These are standard entries created by the Active Form Wizard. While the declarations for these should not be modified, their implementations may be. Among the events of particular interest are the DestroyEvent and the PaintEvent, whose implementations may be modified to accomplish tasks that need to occur during object destruction and painting, respectively.

For tasks that need to occur immediately following object creation, complete the implementation of the Initialize method that may be found in the public declarations for the Active Form class. Do not use the CreateEvent for this purpose as our experience has shown that this is not reliably invoked during object creation.

B.13.2 Designing the Form

Next, we will add a TMemo control and four buttons to the form. First, double-click the TMemo component on the component palette to add it to the form. Set its Align property to alTop and adjust its height as desired. Add four TButton components to the form using the same technique, and arrange them at the bottom of the form as shown below. Set the Anchors property of each to [akLeft,akBottom].

Figure B-15: Designing the form

Now modify the Caption property of each button, from left to right, to **Sync RPC**, **Async RPC**, **Fire Event**, and **Clear**. Double-click the far-right button that is now labeled **Clear** to generate a handler for its click event and complete it as follows:

```
Procedure TDemoControl1X.Button4Click(Sender: TObject);
Begin
  Memo1.Clear;
End;
```

B.13.3 Accessing the Session Object

Before we can perform a remote procedure call, we must obtain access to the Session object within the CSS. For Delphi to access any COM object, it must first generate a type declaration for the object's type library. This type declaration consists of an automatically generated unit that contains code that is the Delphi equivalent of the type library's object and interface declarations. This allows the Delphi compiler to understand a type library declaration in its own native format. To do this, select **Project | Import Type Library...** from the Delphi menu. Select **CIAI Component Support Services (Version x)** from the list box.

If there are multiple versions listed, select the highest version. If this entry does not exist, it means that that CSS has never been registered on your machine (see the section on COM registration to see how this is done). Deselect the **Generate Component Wrapper** check box as shown below:

Figure B-16: Import Type Library dialog

Finally, click **Create Unit**. This will create a file called CIA_CSS_TLB.pas in the Imports folder under the Delphi installation directory. Note that you only need to do this procedure the first time you need to reference a COM object. The file created in this manner is then available to all projects. It also should be done whenever an object's type library changes to ensure that Delphi recognizes the changes.

Now that you have a Delphi type declaration for the CSS, we can add code to access the Session object. Select the DemoControl1 unit in the code editor and add a reference to the CIA_CSS_TLB unit to the uses clause at the top as shown:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, DemoControl_TLB, StdVcl, StdCtrls, CIA_CSS_TLB;
```

Next, add an instance variable to the private section of the TDemoControlX class declaration call FSession. This will be used to hold a reference to the session object.

```
  private
    { Private declarations }
    FSession: ICSS_Session;
    FEvents: IDemoControlXEvents;
    procedure ActivateEvent(Sender: TObject);
```

Finally, add the following line of code to the Initialize method to obtain the Session object reference:

```
procedure TDemoControlX.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
  FSession := CoCSS_Server.Create.Session;
end;
```

B.13.4 Accessing the Patient Context Object

Before we can access the Patient Context object, its type library must be imported in the same manner as shown above. From the **Import Type Library** dialog, select the **CIAI Patient Context Object (Version x)** entry and click **Create Unit**. This will create the **CSS_Patient_TLB.pas** file which contains the Delphi declarations necessary for accessing the Patient Context object.

Add a reference to this unit to the uses clause of the DemoControl1 unit:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ActiveX, AxCtrls, DemoControl_TLB, StdVcl, StdCtrls, CIA_CSS_TLB, CSS_Patient_TLB;
```

Declare an instance variable to hold the reference:

```
  private
    { Private declarations }
    FPatient: ICSS_Patient;
    FSession: ICSS_Session;
    FEvents: IDemoControlXEvents;
```

Now, we need to obtain a reference to the Patient Context object. Because this object is just a special kind of service, we use the Session object to retrieve a reference to it. Insert the following line of code in the Initialize method to do this:

```
procedure TDemoControlX.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
  FSession := CoCSS_Server.Create.Session;
  FPatient := FSession.FindServiceByCLSID(CLASS_Patient) as ICSS_Patient;
end;
```

This code will cause the Session to locate (and start if not already started) the indicated service, in this case the Patient Context object. Because this function returns a reference to the default IUnknown interface, it must be cast to the desired interface, in this case ICSS_Patient.

B.13.5 Calling a Remote Procedure in Synchronous Mode

Now we are ready to add code that will invoke a remote procedure and return its data in the TMemo control. First, we will add a click handler to the first button (labeled **Sync RPC**) to execute a remote procedure that returns detailed information about the currently selected patient and populates the TMemo control with the results.

Double-click the far-left button in the form designer and add the following code to its click event handler:

```
procedure TDemoControlX.Button1Click(Sender: TObject);
begin
  if FPatient.Handle = 0
  then Memo1.Lines.Text := 'No patient is currently selected'
  else Memo1.Lines.Text := FSession.CallRPCText('BEHOPTCX PTINQ',FPatient.Handle);
end;
```

This code calls the remote procedure named BEHOPTCX PTINQ, passing it a single parameter corresponding to the patient's internal entry number (DFN), and places the return text in the TMemo control. If the patient context is empty, it displays a message to that effect instead.

B.13.6 Testing the Component

Before we proceed to add additional functionality, let us test what we currently have. To do this, we must first compile the project. Select **Project | Build DemoControl** from Delphi's menu. The project should compile without errors. Next, we have to do the COM registration. Select **Run | Register ActiveX Server** from Delphi's menu to do this. You should receive confirmation of successful registration.

Finally, we need to register our new component to the VueCentric Framework. The easiest way to do this is to use the VueCentric System Management Utility. Run the tool and login to the remote host. Select the **Object Registry** tab and in the **Restrict List To** box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework). The list of objects will refresh. Now search the list for the programmatic identifier of our newly created component, DemoControl.DemoControlX, and select that entry.

In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the Name field with the display name for the control. The choice of name is arbitrary. We'll call it Demo Control. Finally fill in the height and width fields with 100 and 200, respectively.

Click **Apply** at the bottom. The display should look something like this:

Figure B-17: VueCentric System Management window

You may now close the utility. To test the component:

1. Start the Visual Interface Manager with the following flags: **vim.exe /noupdate /blank /trace**.
2. After logging in, enter Design Mode.
3. Right-click on the desktop and select **Add Object**.
4. Expand the **Name** node and locate and add the Patient Identification Header.
5. Top align this control (right-click it in Design Mode to do this).
6. Find and add the Demo Control.
7. Set the alignment of this control to all.
8. Save this as a template named **%DEMO** for later retrieval.
9. Exit design mode and click **Sync RPC**.

You should see text in the memo control. Try clicking the **Patient Identification Header** control and changing the patient selection. The contents of the memo control are unaffected since we have not yet subscribed to patient context change events. However, if we click the **Sync RPC** button again, the text that appears in the memo control now pertains to the newly selected patient.

Figure B-18: VueCentric memo control

Now close the application.

> **Note:** If you do not close the application now, you will receive an error the next time you try to compile the project because the control's executable image is locked.

B.13.7 Subscribing to Patient Context Changes

As we noticed, our component retrieves information based on the currently selected patient, but it does not respond when the patient selection changes. Let us fix this deficiency by having our component subscribe to patient context changes and modify the memo control's contents when the context change occurs. To do this, we need to use the type library editor.

To view the type library editor, choose the **View | Type Library** menu option from Delphi's main menu. You should now see something similar to Figure B-19.

Figure B-19: DemoControl window

Your GUIDs will be different, but otherwise the dialog should look the same as this one. On the left, you see several entries. The IDemoControlX entry refers to the default interface for our control.

Expanding that entry would reveal all of the properties and methods that are exposed on the COM interface. IDemoControlXEvents is the standard event interface for our component, which we will not make use of. DemoControlX is the actual component itself. To make a component respond to context changes, the component must implement the context change callback interface that is declared by the context object itself.

To add this interface declaration to our component, we must first reference the context object's type library in our component's type library. To do this, select the top node labeled DemoControl. This node corresponds to our type library. When it is selected, several tabs appear on the right.

Select the one labeled **Uses**. You should see the following:

Figure B-20: DemoControl window

This view shows the type libraries that are currently referenced by this type library. To view all known type libraries, right-click the right pane and select **Show All Type Libraries** from the pop-up menu.

You will now see a much longer list. Scroll down until you find the **Patient Context** object and check that entry:

Figure B-21: DemoControl window

Now add the Patient Context object's context change interface to the component. Select the **DemoControlX** node on the left and the Implements tab on the right. Right-click the right pane and choose **Insert Interface** from the pop-up menu. A list of known interfaces displays (Figure B-22).

Figure B-22: Insert Interface dialog

Select the **ICSS_PatientEvents** interface from this list and click **OK**. The type library editor should now look like this:

Figure B-23: DemoControl window

Now we need to synchronize our program code with the changes we have made in the type library editor. To do this, click the refresh button in the toolbar to refresh your code. You should now see three new methods in your components class declaration and three empty implementations in the code section:

```
function TDemoControlX.Pending(Silent: WordBool): WideString;
begin

end;

procedure TDemoControlX.Canceled;
begin

end;

procedure TDemoControlX.Committed;
begin

end;
```

We will add code to each of these implementations to populate the memo control with text indicating that each method has been invoked. Complete the implementations with the code shown below:

```
function TDemoControlX.Pending(Silent: WordBool): WideString;
begin
  Memo1.Clear;
end;

procedure TDemoControlX.Canceled;
begin
  Memo1.Lines.Text := 'Context Change Canceled.';
end;

procedure TDemoControlX.Committed;
begin
  Memo1.Lines.Text := 'Context Change Committed.';
end;
```

When a context change is initiated, the Pending method will be called first. In this method, we simply clear the memo control's contents. Because we are not setting the return value for the Pending method, we are essentially voting YES to the context change. Assuming nothing else cancels the pending change, the Committed method is called next. In that method, we set the memo control's text to indicate that the context change was committed. Now compile the project by selecting **Project | Build DemoControl** from the menu. Since we made changes to the type library, we need to re-register the control by selecting **Run | Register ActiveX Server**.

Restart the Visual Interface Manager, this time with the following command line options:

vim.exe /noupdate /trace /template=%DEMO

Once you login, you should see your component much as it looked before. Click the **Sync RPC** button to verify that this still works. Now click the patient identification header and select a different patient. Notice how the memo control's contents have now changed. We have now responded to a context change event.

Figure B-24: VueCentric dialog

B.13.8 Calling a Remote Procedure in Asynchronous Mode

Our next task is to support calling our remote procedure asynchronously. To do this, we need to implement the ICSS_SessionEvents interface that is defined in the CSS type library. This is done in an identical manner to the ICSS_PatientEvents interface we implemented in the previous section. First, bring up the type library editor by selecting **View | Type Library**. Select the **DemoControl** node on the left and the Uses tab on the right.

If only checked entries are showing, right-click the right pane and select **Show All Type Libraries** from the pop-up menu. Find the CIA Component Support Services library in the list (select the highest version number if you see more than one) and check it. Next select the **DemoControlX** node on the left and the **Implements** tab on the right. Right-click the right pane and select **Insert Interface** from the pop-up menu. Select the **ICSS_SessionEvents** interface and click **OK**. Finally, click the refresh button to refresh your code. You should now see three new methods in your code, all with empty implementations:

```
procedure TDemoControlX.EventCallback(const EventType,
  EventStub: WideString);
begin

end;

procedure TDemoControlX.RPCCallback(Handle: Integer;
  const Data: WideString);
begin

end;

procedure TDemoControlX.RPCCallbackError(Handle, ErrorCode: Integer;
  const ErrorText: WideString);
begin

end;
```

At this point, we will ignore the EventCallback method, but will return to it later. Let us add the following code to the other two methods:

```
procedure TDemoControlX.RPCCallback(Handle: Integer;
  const Data: WideString);
begin
  FHandle := 0;
  Memo1.Lines.Text := Data;
end;

procedure TDemoControlX.RPCCallbackError(Handle, ErrorCode: Integer;
  const ErrorText: WideString);
begin
  FHandle := 0;
  Memo1.Lines.Text := 'An error occurred: ' + ErrorText;
end;
```

Here, we add the data returned by the asynchronous call to the memo control if it completed normally, or the text of the reported error if it did not.

Next, we will add code to the **Async RPC** button to call our remote procedure in asynchronous mode. To do this, double-click that button in the form designer and complete the click event handler as shown:

```
procedure TDemoControlX.Button2Click(Sender: TObject);
begin
  if FHandle <> 0
  then FSession.CallRPCAbort(FHandle);

  Memo1.Lines.Text := 'Asynchronous Remote Procedure Invoked.';

  FHandle := FSession.CallRPCAsync('BEHOPTCX PTINQ',FPatient.Handle,self,True);
end;
```

Note that we are first aborting any asynchronous remote procedure in progress before we call it again.

Now add a declaration to the private section of the component's class for the FHandle variable used to store the returned handle:

```
  private
    { Private declarations }
    FHandle: Integer;
    FPatient: ICSS_Patient;
    FSession: ICSS_Session;
```

Now, since we are already equipped to respond to patient context changes, let us add code to abort an asynchronous call in progress when a context change occurs. We will add this code to the Committed method:

```
procedure TDemoControlX.Committed;
begin
  Memo1.Lines.Text := 'Context Change Committed.';

  if FHandle <> 0
  then begin
    FSession.CallRPCAbort(FHandle);
    FHandle := 0;
  end;
end;
```

Now it is time to test our component again. Recompile the project by selecting **Project | Build DemoControl** and, because we have again made type library changes, re-register the component by selecting **Run | Register ActiveX Server**. Now restart the Visual Interface Manager as before. This time, click the **Async RPC** button. You should at first see the text "Asynchronous Remote Procedure Invoked" in the memo control. After a small delay, you should see the results of the remote procedure appear. Note that TaskMan must be running to invoke a remote procedure asynchronously, so if you do not receive data from the call, make certain TaskMan is running.

B.13.9 Firing an Event

Let us now add the capability of firing an event. We are going to fire a local event called "STATUS." The Visual Interface Manager subscribes to this event and displays the data associated with it in its status bar. Double-click the **Fire Event** button in the form designer and complete the click event handler as follows:

```
procedure TDemoControlX.Button3Click(Sender: TObject);
begin
  FSession.EventFireLocal('STATUS','Status event fired by DemoControl.');
end;
```

Recompile the project and test in the Visual Interface Manager as before. Click the **Fire Event** button and you should see the event data appear in the status bar:

Figure B-25: Status bar

B.13.10 Subscribing and Responding to an Event

Finally, we will enable our component to respond to STATUS events. This requires two steps. First, we must implement the callback interface for responding to events. Since this is the same interface (ICSS_SessionEvents) we implemented earlier for responding to asynchronous remote procedures, we have already done this. All we need to do is to fill in the implementation for the EventCallback method.

Find this method in your component's implementation section and complete as follows:

```
procedure TDemoControlX.EventCallback(const EventType,
  EventStub: WideString);
begin
  Memo1.Lines.Text := EventType + ': ' + EventStub;
end;
```

This will display the event name and data in the memo control.

Next, we need to subscribe to the STATUS event. To do this, return to the Initialize method and add the following line of code:

```
procedure TDemoControlX.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
  FSession := CoCSS_Server.Create.Session;
  FPatient := FSession.FindServiceByCLSID(CLASS_Patient) as ICSS_Patient;
  FSession.EventSubscribe('STATUS',self);
end;
```

Now recompile and test your component as before. Now, clicking the **Fire Event** button also displays the event data in the memo control:

Figure B-26: VueCentric dialog

B.13.11 Summary

You have learned how to create an Active Form control, edit type libraries, reference the Session and Patient Context objects, call remote procedures synchronously and asynchronously, and fire and receive events. You should now be well prepared to create components on your own.

B.14 Creating Visual Components with Visual Basic

Visual Basic offers the ActiveX Control project type for creating visual components. This control is the equivalent to Delphi's Active Form in that it is essentially a form hosted within an ActiveX wrapper.

B.14.1 Creating the ActiveX Control Project

To create an ActiveX Control project, select **File | New Project** from Visual Basic's main menu. This will bring up the project selection dialog as show below:

Figure B-27: New Project dialog

Select the **ActiveX Control** project type and click **OK**. This creates a project called Project1 (which will form the first part of the component's programmatic identifier) containing an ActiveX control called UserControl1 (which will form the second part of the component's programmatic identifier):

Figure 93-2: Visual Basic Design pane

Since we would like our programmatic identifier for this control to be DemoControl.DemoControl2, we need to rename the project and the control. To rename the project, select the project in the Project pane (upper right in the above illustration), and edit its Name property in the Properties pane (lower right) to be DemoControl. To rename the control, select the control name in the Project pane and edit its Name property in the Properties pane to be DemoControl2.

Our project should now look like this:

Figure B-28: Visual Basic Design pane

There is one final setting we should change for our project. Select **Project | DemoControl Properties** from the main menu. Select the **Make** tab and check the **Auto Increment** check box under Version Number. Close the dialog by clicking **OK**.

B.14.2 Designing the Form

Next, we are going to add a TextBox control and four buttons to the form. First, double-click the TextBox component on the component palette to add it to the form. Resize the control in the form designer to fill the upper 3/4 of the form. In the properties pane, change its MultiLine property to True. Add four CommandButton controls to the form using the same technique, and arrange them at the bottom of the form as shown in Figure B-29

Figure B-29: Visual Basic Design pane

Now modify the Caption property of each button, from left to right, to **Sync RPC**, **Async RPC**, **Fire Event**, and **Clear**. Double-click the far-right button that is now labeled **Clear** to generate a handler for its click event and complete it as shown below:

```
Private Sub Command4_Click()
    Text1.Text = ""
End Sub
```

B.14.3 Accessing the Session Object

> **Note:** Visual Basic can have difficulty locating components that are registered using side-by-side versioning. You may find the need to copy the required components to the system directory before adding them as references to the project.

Before we can perform a remote procedure call, we must obtain access to the Session object within the CSS. For Visual Basic to access any COM object, it must first reference that object within its project. To do this, select **Project | References...** from the main menu. In the reference list, find the **CIAI Component Support Services** entry and select it:

Figure B-30: References dialog

Now click **OK** to close the form. The project now has a reference to the CSS type library and can access objects within it.

Next, we need to declare a global variable to hold our reference to the Session Object. To do this, select **(General)** in the code editor and add the following code:

Figure B-31: DemoControl2

Now we need to store a reference to the Session object in our global variable, gSession. Select the **UserControl** class in the left drop-down box of the code editor and its Initialize method in the right drop-down box. This will create an empty implementation for the Initialize method. Next, complete the implementation by adding the following code:

Figure B-32: DemoControl – Private Sub UserControl Initialize

This code obtains a temporary reference to the Server object, retrieves a reference to its Session object, and releases the Server object. At this point, you now have a reference to the Session object stored in the global variable, gSession.

B.14.4 Accessing the Patient Context Object

Before we can access the Patient Context object, we must add a reference to it to our project in the same manner as the Session object. To do this, select **Project | References...** from the main menu and locate "CIAI Patient Context Object" in the reference list. Check the box next to the entry and close the dialog by clicking **OK**.

Next, we need to add a global variable to hold the reference to the Patient Context object, just as we did for the Session object. Return to the (General) section of the code editor and add a global variable declaration as shown:

```
Dim gPatient As CSS_Patient.Patient
Dim gSession As CIA_CSS.CSS_Session
```

Now return to the Initialize method of the UserControl and add the following code to initialize the variable:

B.14.5 Calling a Remote Procedure in Synchronous Mode

Now we are ready to add code that will invoke a remote procedure and return its data in the TextBox control. First, we will add a click handler to the first button (labeled **Sync RPC**) to execute a remote procedure that returns detailed information about the currently selected patient and populates the TextBox control with the results. Double-click the far-left button in the form designer and add the following code to its click event handler:

```
Private Sub Command1_Click()
    If gPatient.Handle = 0 Then
        Text1.Text = "No patient is currently selected"
    Else
        Text1.Text = gSession.CallRPCText("BEHOPTCX PTINQ", gPatient.Handle)
    End If
End Sub
```

This code calls the remote procedure named BEHOPTCX PTINQ, passing it a single parameter corresponding to the patient's internal entry number (DFN), and places the return text in the TextBox control. If the patient context is empty, it displays a message to that effect instead.

B.14.6 Testing the Component

Before we proceed to add additional functionality, let us test what we currently have. To do this, we must first compile the project. Select **File | Make DemoControl.ocx...** from the main menu. The project should compile without errors.

Next, we need to register our new component to the VueCentric Framework. The easiest way to do this is to use the VueCentric System Management Utility. Run the tool and login to the remote host. Select the **Object Registry** tab and in the **Restrict List To** box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework).

The list of objects will refresh. Now search the list for the programmatic identifier of our newly created component, DemoControl.DemoControl2, and select that entry. In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the **Name** field with the display name for the control. The choice of name is arbitrary. We'll call it Demo Control.

Finally fill in the height and width fields with 100 and 200, respectively. Now click **Apply** at the bottom. The display should look something like this:

Figure B-33: VueCentric System Management dialog

You may now close the utility. To test the component, start the Visual Interface Manager with the following flags:

vim.exe /noupdate /blank /trace

After logging in, enter Design Mode, right-click the desktop and select **Add Object**. Expand the Name node and locate and add the Patient Identification Header. Top align this control (right-click it in Design Mode to do this).

Next find and add the Demo Control. Set the alignment of this control to all. Save this as a template named %DEMO for later retrieval. Now exit design mode and click the **Sync RPC** button. You should see text in the TextBox control. Try clicking the Patient Identification Header control and changing the patient selection.

Note that the contents of the TextBox control is unaffected, since we have not yet subscribed to patient context change events. However, if we click the **Sync RPC** button again, the text that appears in the TextBox control now pertains to the newly selected patient.

Figure B-34: VueCentric dialog

Now close the application (note: if you do not close the application now, you will receive an error the next time you try to compile the project because the control's executable image is locked).

B.14.7 Subscribing to Patient Context Changes

As we noticed, our component retrieves information based on the currently selected patient, but it does not respond when the patient selection changes. Let us fix this deficiency by having our component subscribe to patient context changes and modify the TextBox control's contents when the context change occurs. To do this, we need to direct our component to implement the callback interface for patient context changes, ICSS_PatientEvents. Return to the (General) section of the code editor and enter the following line of code:

```
Implements ICSS_PatientEvents

Dim gPatient As CSS_Patient.Patient
Dim gSession As CIA_CSS.CSS_Session
```

From the left drop-down box in the code editor, find and select the "ICSS_PatientEvents" entry. In the right drop-down box, you will see three methods: Pending, Canceled, and Committed. Select each one in turn to generate implementations for each. Your code should look like this:

We will add code to each of these implementations to populate the TextBox control with text indicating that each method has been invoked. Complete the implementations with the code shown below:

When a context change is initiated, the Pending method will be called first. In this method, we simply clear the TextBox control's contents. Because we are not setting the return value for the Pending method, we are essentially voting YES to the context change. Assuming nothing else cancels the pending change, the Committed method is called next. In that method, we set the TextBox control's text to indicate that the context change was committed. Now compile the project by selecting **File | Make DemoControl.ocx...** from the menu. Now restart the Visual Interface Manager, this time with the following command line options:

vim.exe /noupdate /trace /template=%DEMO

Once you login, you should see your component much as it looked before. Click the **Sync RPC** button to verify that this still works. Now click the patient identification header and select a different patient. Notice how the TextBox control's contents have now changed. We have now responded to a context change event.

Figure B-35: VueCentric dialog

B.14.8 Calling a Remote Procedure in Asynchronous Mode

Our next task is to support calling our remote procedure asynchronously. To do this, we need to implement the ICSS_SessionEvents interface that is defined in the CSS type library. This is done in an identical manner to the ICSS_PatientEvents interface we implemented in the previous section. Return again to the (General) section of the code editor and add the following:

```
Implements ICSS_SessionEvents
Implements ICSS_PatientEvents

Dim gPatient As CSS_Patient.Patient
Dim gSession As CIA_CSS.CSS_Session
```

As we did with the ICSS_PatientEvents interface, select "ICSS_SessionEvents" from the left drop-down box and each of its three methods in turn from the right drop-down box. This will create blank implementations for each method.

```
Private Sub ICSS_SessionEvents_EventCallback(ByVal EventType As String, ByVal EventStub As String)

End Sub

Private Sub ICSS_SessionEvents_RPCCallback(ByVal Handle As Long, ByVal Data As String)

End Sub

Private Sub ICSS_SessionEvents_RPCCallbackError(ByVal Handle As Long, ByVal ErrorCode As Long, ByVal ErrorText As String)

End Sub
```

At this point, we will ignore the EventCallback method, but will return to it later. Let us add the following code to the other two methods:

```
Private Sub ICSS_SessionEvents_RPCCallback(ByVal Handle As Long,
    gHandle = 0
    Text1.Text = Data
End Sub

Private Sub ICSS_SessionEvents_RPCCallbackError(ByVal Handle As
    gHandle = 0
    Text1.Text = "An error occurred: " + ErrorText
End Sub
```

Here, we add the data returned by the asynchronous call to the TextBox control if it completed normally, or the text of the reported error if it did not.

Next, we will add code to the **Async RPC** button to call our remote procedure in asynchronous mode. To do this, double-click that button in the form designer and complete the click event handler as shown:

```
Private Sub Command2_Click()
    If gHandle <> 0 Then gSession.CallRPCAbort (gHandle)
    Text1.Text = "Asynchronous Remote Procedure Invoked."
    gHandle = gSession.CallRPCAsync("BEHOPTCX PTINQ", gPatient.Handle, Me, True)
End Sub
```

> **Note:** We are first aborting any asynchronous remote procedure in progress before we call it again.

Now add the global variable declaration to the (General) section of the code editor to store the returned handle:

```
Dim gHandle As Integer
Dim gPatient As CSS_Patient.Patient
Dim gSession As CIA_CSS.CSS_Session
```

Now, since we are already equipped to respond to patient context changes, let us add code to abort an asynchronous call in progress when a context change occurs. We will add this code to the Committed method:

```
Private Sub ICSS_PatientEvents_Committed()
    Text1.Text = "Context Change Committed."
    If gHandle <> 0 Then
        gSession.CallRPCAbort (gHandle)
        gHandle = 0
    End If
End Sub
```

Now it is time to test our component again. Recompile the project by selecting **File | Make DemoControl.ocx...** Now restart the Visual Interface Manager as before. This time, click the **Async RPC** button. You should at first see the text "Asynchronous Remote Procedure Invoked" in the TextBox control. After a small delay, you should see the results of the remote procedure appear. Note that TaskMan must be running to invoke a remote procedure asynchronously, so if you do not receive data from the call, make certain TaskMan is running.

B.14.9 Firing an Event

Let us now add the capability of firing an event. We are going to fire a local event called "STATUS." The Visual Interface Manager subscribes to this event and displays the data associated with it in its status bar. Double-click the **Fire Event** button in the form designer and complete the click event handler as follows:

```
Private Sub Command3_Click()
    gSession.EventFireLocal "STATUS", "Status event fired by DemoControl."
End Sub
```

Recompile the project and test in the Visual Interface Manager as before. Click the **Fire Event** button and you should see the event data appear in the status bar:

Figure B-36: Status bar

B.14.10 Subscribing and Responding to an Event

Finally, we will enable our component to respond to STATUS events. This requires two steps. First, we must implement the callback interface for responding to events. Since this is the same interface (ICSS_SessionEvents) we implemented earlier for responding to asynchronous remote procedures, we have already done this. All we need to do is to fill in the implementation for the EventCallback method.

Find this method in your component's implementation section and complete as follows:

```
Private Sub ICSS_SessionEvents_EventCallback(ByVal EventType
    Text1.Text = EventType + ": " + EventStub
End Sub
```

This will display the event name and data in the TextBox control.

Next, we need to subscribe to the STATUS event. To do this, return to the Initialize method and add the following line of code:

```
Private Sub UserControl_Initialize()
    Dim Server As New CIA_CSS.CSS_Server
    Set gSession = Server.Session
    Set Server = Nothing
    Set gPatient = gSession.FindServiceByProgID("CSS_PATIENT.PATIENT")
    gSession.EventSubscribe "STATUS", Me
End Sub
```

Now recompile and test your component as before. Now, clicking the **Fire Event** button also displays the event data in the TextBox control:

Figure B-37: VueCentric dialog

B.14.11 Summary

You have learned how to create an ActiveX control, reference the Session and Patient Context objects, implement interfaces, call remote procedures synchronously and asynchronously, and fire and receive events. You should now be well prepared to create components on your own.

B.15 Creating Visual Components with C#

.NET Window Controls may be used in the VIM through a COM-interoperability feature of the .NET framework known as the COM Callable Wrapper (CCW). This wrapper transparently exposes a .NET Window Control as an ActiveX object and requires little effort on the part of the developer. While any .NET-compatible language may be used, we choose C# for this example.

B.15.1 Creating the Windows Control Project

To create an Active Form project, select **File | New | Project...**. This will display the project selection dialog as shown below:

Figure B-38: New Project dialog

Select Visual C# Projects in the left pane and Windows Control Library in the right pane. Modify the Name to **DemoControl** and click **OK**.

The form designer will display:

Figure B-39: DemoControl window

Next, right-click the node named "DemoControl" from the Solution Explorer pane (upper right) and select **Properties** from the pop-up menu. Select **Configuration Properties | Build** from the left pane and change the "Register for COM Interop" property in the right pane to True. Click **OK** to close the dialog.

Figure B-40: DemoControl window

Next, rename the component from UserControl1 to **DemoControlCS** by first clicking the control's form in the Form Designer (upper left) and then modifying its **Name** property in the **Properties** pane (lower right) to **DemoControlCS**.

Also, change the **BackColor** property to "**ControlLight**." (If you do not change the default color, it will appear black in the VIM.) Then resize the form on the form designer to accommodate the controls you will be placing on it by grabbing the lower right sizing grip and dragging it to its desired position. Your project should now look like this:

Figure B-41: DemoControl window

Now we are going to add a text box and four buttons to the form. First, open the component palette by hovering the mouse pointer over the ToolBox icon to the left of the Form Designer. Locate the TextBox control (you may need to scroll the component list to find it) and double-click to add it to your form. Next, from the Form Designer, select the newly added TextBox control by clicking it. In the Properties pane, edit its Dock property to Top and its MultiLine property to True. Then resize it to fill most of the form (allowing space at the bottom for buttons) by grabbing its resizing grip and dragging to the desired position.

Next, add four buttons to the form by opening the component palette as before and double-clicking the Button control. Repeat this three more times. You will see four buttons on your form. Drag them to the desired positions at the bottom of the form. Set the Anchor property for each button to (Bottom,Left). An easy way to do this is to select the first button by clicking it, then clicking the remaining three while holding down the Shift key. This will select all four buttons. Then modify the Anchor property to the desired value. This will change the property values for all four buttons.

While we are here, we are going to add an empty handler for the Load Event. To do this, switch the property editor view to events by clicking the lightning bolt icon in the toolbar. Find the Load event in the list and double-click in the empty box to the right of it. You should now see the code editor with your empty event handler:

```
private void DemoControlCS_Load(object sender, System.EventArgs e)
{

}
```

We will complete this event handler in a moment. For now, return to the design view by clicking the appropriate tab. Your form should now look like this:

Figure B-42: DemoControl window

Now modify the Text property of each button, from left to right, to **Sync RPC**, **Async RPC**, **Fire Event**, and **Clear**. Double click the far-right button that is now labeled **Clear** to generate a handler for its click event and complete it as shown below:

```
        private void button4_Click(object sender, System.EventArgs e)
        {
            textBox1.Clear();
        }
```

B.15.2 Accessing the Session Object

Before we can perform a remote procedure call, we must obtain access to the Session object within the CSS. For C# to access any COM object, we must first add its reference information to the project. This is done by selecting **Project | Add Reference...** from the menu. Select the **COM** tab and locate the entry named **CIAI Component Support Services** from the list. Select that entry by clicking it, the click the **Select** button. The list item should now appear in the bottom pane of the dialog as shown:

Figure B-43: Add Reference dialog

Finally, click **OK** to close the dialog and add the reference to your project. You will now see a new entry in the Solution Explorer pane under the References node named CIA_CSS.

Now that you have added reference information for the CSS, we can add code to access the Session object. First, add an instance variable to the class declaration of your control and name it "Session." This will be used to hold a reference to the session object.

```
public class DemoControlCS : System.Windows.Forms.UserControl
{
    private CIA_CSS.ICSS_Session session;
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Button button2;
    private System.Windows.Forms.Button button3;
    private System.Windows.Forms.Button button4;
```

To obtain a reference to the Session object, add the following line of code to the Load event handler we created earlier:

```
private void DemoControlCS_Load(object sender, System.EventArgs e)
{
    session=new CIA_CSS.CSS_ServerClass().Session;
}
```

We must also be sure to properly release this reference when the object is destroyed. To do this, locate the Dispose method implementation for the control and add the following lines of code:

```
protected override void Dispose( bool disposing )
{
    if(session!=null)
    {
        Marshal.ReleaseComObject(session);
        session = null;
    }

    if( disposing )
    {
        if( components != null )
            components.Dispose();
    }
    base.Dispose( disposing );
}
```

B.15.3 Accessing the Patient Context Object

Before we can access the Patient Context object, its reference information must be added to the project in the same manner as you did with the Session object. From the Add Reference dialog, double-click the entry “CIAI Patient Context Object” to add it and click **OK** to close the dialog. A new reference should appear in the Solution Explorer pane named CSS_Patient.

Next, declare an instance variable to hold the reference:

```
public class DemoControlCS : System.Windows.Forms.UserControl
{
    private CIA_CSS.ICSS_Session session;
    private CSS_Patient.ICSS_Patient patient;
    private System.Windows.Forms.TextBox textBox1;
    private System.Windows.Forms.Button button1;
    private System.Windows.Forms.Button button2;
    private System.Windows.Forms.Button button3;
    private System.Windows.Forms.Button button4;
```

Now, we need to obtain a reference to the Patient Context object. Because this object is just a special kind of service, we use the Session object to retrieve a reference to it. Add the following line of code to the Load event handler to do this:

```
private void DemoControlCS_Load(object sender, System.EventArgs e)
{
    session=new CIA_CSS.CSS_ServerClass().Session;
    patient=session.FindServiceByProgID("CSS_Patient.Patient") as CSS_Patient.ICSS_Patient;
}
```

The above code will cause the Session to locate (and start if not already started) the indicated service, in this case the Patient Context object. Because this function returns a reference to the default IUnknown interface, it must be cast to the desired interface, in this case ICSS_Patient.

As before, we must release all object references when the control is destroyed. To do this, add the following lines of code to the Dispose method:

```
protected override void Dispose( bool disposing )
{
    if(session!=null)
    {
        Marshal.ReleaseComObject(session);
        session = null;
    }

    if (patient!=null)
    {
        Marshal.ReleaseComObject(patient);
        patient = null;
    }

    if( disposing )
    {
        if( components != null )
            components.Dispose();
    }
    base.Dispose( disposing );
}
```

B.15.4 Calling a Remote Procedure in Synchronous Mode

Now we are ready to add code that will invoke a remote procedure and display its data in the TextBox control. First, we will add a click handler to the first button (labeled **Sync RPC**) to execute a remote procedure that returns detailed information about the currently selected patient and populates the TextBox control with the results. Double-click the far-left button in the form designer and add the following code to its click event handler:

```
private void button1_Click(object sender, System.EventArgs e)
{
    if(patient.Handle==0)
        textBox1.Text="No patient is currently selected";
    else
        textBox1.Text=session.CallRPCText("BEHOPTCX PTINQ",patient.Handle);
}
```

This code calls the remote procedure named BEHOPTCX PTINQ, passing it a single parameter corresponding to the patient's internal entry number (DFN), and displays the return text in the TextBox control. If the patient context is empty, it displays a message to that effect instead.

B.15.5 Testing the Component

Before we proceed to add additional functionality, let us test what we currently have. Before compiling the project, we must first assign a unique GUID to our new control. If we do not do this, the control will be assigned a new GUID every time it is registered, which is not desirable. First, we must add a reference to the COM Interop Services namespace to our project as shown:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Runtime.InteropServices;
```

Next, we must obtain a unique GUID to assign to our component. To do this, select **Tools | Create GUID** from the menu. The following dialog appears:

Figure B-44: Create GUID dialog

Select the format labeled "Registry Format," click the **Copy** button to copy the GUID to the clipboard, then click **Exit** to close the dialog. To assign a static GUID to our control, insert the following attribute declaration just prior to the class declaration for the control. Where the GUID appears in the declaration, paste the contents of the clipboard and delete the curly braces.

```
[GuidAttribute("048FBBD9-65F0-473b-888A-909FAD6225C8")]
public class DemoControlCS : System.Windows.Forms.UserControl
{
    private CIA_CSS.ICSS_Session session;
    private CSS_Patient.ICSS_Patient patient;
```

Now we are ready to compile the project. Select **Build | Build Solution** from the menu.

Finally, we need to register our new component to the **VueCentric** Framework. The easiest way to do this is to use the **VueCentric** System Management Utility. Run the tool and login to the remote host. Select the **Object Registry** tab and in the **Restrict List To** box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework). The list of objects will refresh. Now search the list for the programmatic identifier of our newly created component, DemoControl.DemoControlCS, and select that entry. In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the Name field with the display name for the control. The choice of name is arbitrary. We'll call it C# Demo Control. Next, fill in the height and width fields with 100 and 200, respectively. Finally, switch to the **Special Settings** tab and check the box labelled **.NET Component**. Switch back to the **General Settings** tab and click **Apply** at the bottom. The display should look something like this:

Figure B-45: VueCentric System Management dialog

You may now close the utility. To test the component, start the Visual Interface Manager with the following flags:

vim.exe /noupdate /blank /trace

After logging in, enter design mode, right click the desktop and select **Add Object**. Expand the Name node and locate and add the Patient Identification Header. Top align this control (right-click it in design mode to do this). Next find and add the Demo Control. Set the alignment of this control to all. Save this as a template named %DEMO for later retrieval. Now exit design mode and click the **Sync RPC** button. You should see text in the text box control. Try clicking the Patient Identification Header control and changing the patient selection. The contents of the text box control are unaffected, since we have not yet subscribed to patient context change events. However, if we click the **Sync RPC** button again, the text that appears in the text box control now pertains to the newly selected patient.

Figure B-46: VueCentric dialog

Now close the application.

> **Note:** If you do not close the application now, you will receive an error the next time you try to compile the project because the control's executable image is locked.

B.15.6 Subscribing to Patient Context Changes

As we noticed, our component retrieves information based on the currently selected patient, but it does not respond when the patient selection changes. Let us fix this deficiency by having our component subscribe to patient context changes and modify the text box control's contents when the context change occurs. To do this, we need to add a callback interface to our component by modifying its class declaration:

```
public class DemoControlCS : System.Windows.Forms.UserControl, CSS_Patient.ICSS_PatientEvents
{
```

Press TAB to implement stubs for interface 'CSS_Patient.ICSS_PatientEvents'

Visual Studio .NET prompts you to press the Tab key to implement method stubs when you add a new interface. This is a convenience feature that can be a real timesaver. Press the tab key now to generate these stubs. To view them, scroll to the bottom of the program code. You should see a collapsed region named "ICSS_PatientEvents Members." Expand it to reveal the method stubs:

```
#region ICSS_PatientEvents Members

public void Committed()
{
    // TODO:  Add DemoControlCS.Committed implementation
}

public string Pending(bool Silent)
{
    // TODO:  Add DemoControlCS.Pending implementation
    return null;
}

public void Canceled()
{
    // TODO:  Add DemoControlCS.Canceled implementation
}

#endregion
```

We will add code to each of these stub entries to populate the text box control with text indicating that each method has been invoked. Complete the implementations with the code shown below:

```
#region ICSS_PatientEvents Members

public void Committed()
{
    textBox1.Text="Context change committed.";
}

public string Pending(bool Silent)
{
    textBox1.Clear();
    return "";
}

public void Canceled()
{
    textBox1.Text="Context change canceled.";
}

#endregion
```

When a context change is initiated, the Pending method will be called first. In this method, we simply clear the text box control's contents. Because we are returning a null string for the Pending method, we are essentially voting YES to the context change. Assuming nothing else cancels the pending change, the Committed method is called next. In that method, we set the text box control's text to indicate that the context change was committed. Now compile the project by selecting **Build | Build Solution** from the menu. Now restart the Visual Interface Manager, this time with the following command line options:

vim.exe /noupdate /trace /template=%DEMO

Once you log in, you should see your component much as it looked before. Click the **Sync RPC** button to verify that this still works. Now click the patient identification header and select a different patient. Notice how the memo control's contents have now changed. We have now responded to a context change event.

Figure B-47: VueCentric dialog

B.15.7 Calling a Remote Procedure in Asynchronous Mode

Our next task is to support calling our remote procedure asynchronously. To do this, we need to implement the ICSS_SessionEvents interface that is defined in the CSS type library. This is done in an identical manner to the ICSS_PatientEvents interface we implemented in the previous section. First, add the interface to the component's class declaration:

```
, CSS_Patient.ICSS_PatientEvents, CIA_CSS.ICSS_SessionEvents
```

Be sure to press the Tab key when prompted to create the method stubs:

```
#region ICSS_SessionEvents Members

public void RPCCallback(int Handle, string Data)
{
    // TODO:  Add DemoControlCS.RPCCallback implementation
}

public void EventCallback(string EventType, string EventStub)
{
    // TODO:  Add DemoControlCS.EventCallback implementation
}

public void RPCCallbackError(int Handle, int ErrorCode, string ErrorText)
{
    // TODO:  Add DemoControlCS.RPCCallbackError implementation
}

#endregion
```

At this point, we will ignore the EventCallback method, but will return to it later. Let us add the following code to the other two methods:

```
public void RPCCallback(int Handle, string Data)
{
    rpcHandle=0;
    textBox1.Text=Data;
}

public void RPCCallbackError(int Handle, int ErrorCode, string ErrorText)
{
    rpcHandle=0;
    textBox1.Text="An error occurred: "+ErrorText;
}
```

Here, we add the data returned by the asynchronous call to the text box control if it completed normally, or the text of the reported error if it did not.

Next, we will add code to the **Async RPC** button to call our remote procedure in asynchronous mode. To do this, double-click that button in the form designer and complete the click event handler as shown:

```
private void button2_Click(object sender, System.EventArgs e)
{
    if(rpcHandle!=0)
        session.CallRPCAbort(rpcHandle);

    textBox1.Text="Asynchronous Remote Procedure Invoked.";
    rpcHandle=session.CallRPCAsync("BEHOPTCX PTINQ",patient.Handle,this,true);
}
```

Note that we are first aborting any asynchronous remote procedure in progress before we call it again.

Now add an instance variable declaration to the component's class for the rpcHandle variable used to store the returned handle:

```
private CIA_CSS.ICSS_Session session;
private CSS_Patient.ICSS_Patient patient;
private int rpcHandle;
private System.Windows.Forms.TextBox textBox1;
```

Now, since we are already equipped to respond to patient context changes, let us add code to abort an asynchronous call in progress when a context change occurs. We will add this code to the Committed method:

```
public void Committed()
{
    textBox1.Text="Context change committed.";

    if(rpcHandle!=0)
    {
        session.CallRPCAbort(rpcHandle);
        rpcHandle=0;
    }
}
```

Now it is time to test our component again. Recompile the project by selecting **Build | Build Solution**. Now restart the Visual Interface Manager as before. This time click the **Async RPC** button. You should at first see the text "Asynchronous Remote Procedure Invoked." in the memo control. After a small delay, you should see the results of the remote procedure appear. Note that TaskMan must be running to invoke a remote procedure asynchronously, so if you do not receive data from the call, make certain TaskMan is running.

B.15.8 Firing an Event

Let us now add the capability of firing an event. We are going to fire a local event called "STATUS." The Visual Interface Manager subscribes to this event and displays the data associated with it in its status bar. Double-click the **Fire Event** button in the form designer and complete the click event handler as follows:

```
private void button3_Click(object sender, System.EventArgs e)
{
    session.EventFireLocal("STATUS","Status event fired by DemoControl.");
}
```

Recompile the project and test in the Visual Interface Manager as before. Click the **Fire Event** button and you should see the event data appear in the status bar:

Figure B-48: Status bar

B.15.9 Subscribing and Responding to an Event

Finally, we will enable our component to respond to STATUS events. This requires two steps. First, we must implement the callback interface for responding to events. Since this is the same interface (ICSS_SessionEvents) we implemented earlier for responding to asynchronous remote procedures, we have already done this. All we need to do is to fill in the implementation for the EventCallback method. Find this method in your component's implementation section and complete as follows:

Import Type Library dialog:

```
public void EventCallback(string EventType, string EventStub)
{
    textBox1.Text=EventType+": "+EventStub;
}
```

This will display the event name and data in the text box control.

Next, we need to subscribe to the STATUS event. To do this, return to the Load event handler and add the following line of code:

```
private void DemoControlCS_Load(object sender, System.EventArgs e)
{
    session=new CIA_CSS.CSS_ServerClass().Session;
    patient=session.FindServiceByProgID("CSS_PATIENT.PATIENT") as CSS_Patient.ICSS_Patient;
    session.EventSubscribe("STATUS",this);
}
```

Now recompile and test your component as before. Now, clicking the **Fire Event** button also displays the event data in the memo control:

Figure B-49: VueCentric dialog

B.15.10 Summary

You have learned how to create a .NET Windows Control, expose it as an ActiveX object, reference the Session and Patient Context objects, call remote procedures synchronously and asynchronously, and fire and receive events. You should now be well prepared to create components on your own.

B.16 Creating Services

Services, like visual components, are COM objects. Unlike visual components, they are not ActiveX controls, they cannot be manipulated at design time nor do they manifest themselves visually in their baseline state. Despite these differences, the programming techniques are very similar.

In the Delphi and Visual Basic examples that follow, we will be creating a simple service that implements a function to return information about the remote host. We will modify the visual component we created in the previous tutorial to use this service.

B.17 Creating Services with Delphi

This tutorial will demonstrate how to use Delphi to create a simple service object that implements a single method and how to access that service from within another component. Delphi offers several project types that can be used to create COM objects. We will create an automation-compatible COM object, which will afford the most flexibility for use in different settings.

B.17.1 Creating the Project

Creating a project to produce an automation-compatible COM object requires two steps. First, we will create an ActiveX Library project by selecting **File | New | Other...** from the main menu. From the dialog that appears, select the **ActiveX** tab as shown below.

Figure B-50: New Items dialog

Select **ActiveX Library** and click **OK**.

B.17.2 Creating the Service Object

Next, we need to add a COM object that will be our service to our newly created project. To do this, return to the project type dialog by selecting **File | New | Other...**, select the **ActiveX** tab again, but this time select **Automation Object** and click **OK**. In the **Automation Object Wizard** that appears, fill in the CoClass Name as shown below. Leave the other settings as they are.

Figure B-51: Automation Object Wizard

Click **OK** and you should see the type library editor appear with our newly created type library and its automation object, DemoService1, with its default interface, IDemoService1:

Figure B-52: Type Library editor

Select the top node labeled Project1 and rename the type library to DemoService by changing the entry in the **Name** edit box in the right pane. Also change the entry in the **Help String** edit box to DemoService as well. Click the refresh button in the toolbar to synchronize your program code with the change. Now save the project to a folder of your choice, naming the project file DemoService1 (this will give us an executable filename of DemoService1.dll when we are done).

B.17.3 Accessing the Session Object

Since we will be making a remote procedure call, we need to access the Session object. This is done in the exact same manner as we did with the visual component we created earlier. Since we have already imported the type library for the CSS, we do not have to repeat this step (see Appendix B.13). We do need to add a reference to the CSS type declaration unit to the uses clause of the unit containing the Session object. In the code editor, select the Unit1 unit and modify the uses clause as shown:

```
uses
  ComObj, ActiveX, Project1_TLB, StdVcl, CIA_CSS_TLB;
```

Now create a private section for the service object's class and add a declaration for the variable that will hold a reference to the Session object:

```
type
  TDemoService1 = class(TAutoObject, IDemoService1)
  private
    FSession: ICSS_Session;
  protected
    { Protected declarations }
  end;
```

As with our visual component created earlier, we will initialize the FSession variable in the Initialize method. Since the Automation Object Wizard does not automatically generate this for us, we will need to add it manually. Create a public section and add an override declaration for the Initialize method:

```
type
  TDemoService1 = class(TAutoObject, IDemoService1)
  private
    FSession: ICSS_Session;
  protected
    { Protected declarations }
  public
    procedure Initialize; override;
  end;
```

To generate an implementation for the Initialize method, we will use a Delphi shortcut. Right-click the service object's class name (TDemoService1) and select **Complete class at cursor** from the pop-up menu. This will produce a default implementation for the Initialize method:

```
procedure TDemoService1.Initialize;
begin
  inherited;

end;
```

Now add the code to initialize the FSession variable:

```
procedure TDemoService1.Initialize;
begin
  inherited;
  FSession := CoCSS_Server.Create.Session;
end;
```

B.17.4 Modifying the Interface

Now we want to add a method to the interface of our service object. This will be a function that will receive one argument that specifies the type of information requested and will return the requested information as a string. To modify the interface, we will use the type library editor. If it is not already visible, make it visible by selecting **View | Type Library** from the main menu. Select the node labeled IDemoService1. This corresponds to the default interface for the service object. It currently has no methods or properties associated with it.

To create a method, click the refresh button on the toolbar. This will generate a method with the default name of Method1. Rename the method to GetInfo by either changing the label associated with the node or by changing the Name edit box in the right pane. Now switch to the **Parameters** tab. From the **Return Type** drop-down box, select either **WideString** or **BSTR** (the list content depends upon whether you have configured the editor to display Delphi or C syntax).

We also need to add the parameter that will contain the type of information we are requesting. At the bottom of the **Parameters** tab, click **Add**. This will create a parameter with a default name of Param1 and a datatype of Integer. Change the name of the parameter to InfoType by selecting the name in its cell and modifying. Leave the datatype as it is. Now click the refresh button on the toolbar to synchronize your program code with these changes. At this point, the type library editor should look like this:

Figure B-53: Type Library editor

Your program code should now contain a new method:

```
type
  TDemoService1 = class(TAutoObject, IDemoService1)
  private
    FSession: ICSS_Session;
  protected
    function GetInfo(InfoType: Integer): WideString; safecall;
    { Protected declarations }
  public
    procedure Initialize; override;
  end;
```

B.17.5 Providing the Implementation

Now that we have created our method, we need to provide its implementation. Our method will perform a synchronous remote procedure call, passing its single parameter to indicate the type of information we are requesting. It will return the result of the call in its return value. To do this, complete the implementation for the GetInfo method as shown:

```
function TDemoService1.GetInfo(InfoType: Integer): WideString;
begin
  Result := FSession.CallRPCText('CIANBRPC GETINFO',InfoType);
end;
```

B.17.6 Registering the Service

Now we need to register the service. First, let us compile the project by selecting **Project | Build DemoService1** from the menu. It should compile without errors. Now, register the type library by selecting **Run | Register ActiveX Server** from the menu. You should receive confirmation of successful registration.

Next, we need to register our service to the **VueCentric** Framework. This is done in the same way we registered our visual component - using the **VueCentric** System Management Utility. Run the tool and login to the remote host. Select the Object Registry tab and in the Restrict List To box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework). The list of objects will refresh. Now search the list for the programmatic identifier of our newly created service, DemoService.DemoService1, and select that entry. In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the Name field with the display name for the control. The choice of name is arbitrary. We will call it "Demo Service 1." Now switch to the special settings tab and check the box labeled **Service**. Now click **Apply** at the bottom. This completes the Framework registration process.

Finally, let's import the service's type library so that it is available to be used by other projects. Select **Project | Import Type Library...** from the main menu. Find the entry "DemoService (Version 1.0)" and select it. Make sure **Create Component Wrapper** is unchecked and click **Create Unit**. You have now created a type declaration unit in the Delphi Imports folder (if you receive a message that the unit is already in the project, just ignore it and continue).

B.17.7 Accessing the Service

Next, we need to provide a means to test the GetInfo method of our service object. To do this, we will modify the DemoControl we created in the earlier tutorial. Open the DemoControl project now (be sure to save any changes to the existing one). Modify the uses clause in the DemoControl1 unit to add a reference to our service's type library declaration unit.

```
trols, Forms, Dialogs,
, CIA_CSS_TLB, CSS_Patient_TLB, DemoService_TLB;
```

Next, add a declaration for the variable that is to hold the reference to our service in the private section of our TDemoControlX class.

```
  private
    { Private declarations }
    FService: IDemoService1;
    FHandle: Integer;
    FPatient: ICSS_Patient;
```

Add the code to initialize the FService variable to the Initialize method:

```
procedure TDemoControlX.Initialize;
begin
  inherited Initialize;
  OnActivate := ActivateEvent;
  OnClick := ClickEvent;
  OnCreate := CreateEvent;
  OnDblClick := DblClickEvent;
  OnDeactivate := DeactivateEvent;
  OnDestroy := DestroyEvent;
  OnKeyPress := KeyPressEvent;
  OnPaint := PaintEvent;
  FSession := CoCSS_Server.Create.Session;
  FPatient := FSession.FindServiceByCLSID(CLASS_Patient) as ICSS_Patient;
  FSession.EventSubscribe('STATUS',self);
  FService := FSession.FindServiceByProgID('DemoService.DemoService1') as IDemoService1;
end;
```

Note that we are using the FindServiceByProgID here. We could just as easily have used the FindServiceByCLSID and passed the GUID for the service object.

Finally, add a fifth button to the form and position it as desired. Rename the caption to “Demo Service” and set its Anchors property to [akLeft,akBottom]. Double-click the button and complete its Click event handler as shown:

```
procedure TDemoControlX.Button5Click(Sender: TObject);
begin
  Memo1.Lines.Text := FService.GetInfo(2);
end;
```

Compile the project by selecting **Project | Build DemoControl** and load it in the Visual Interface Manager as before. Now click the **Demo Service** button. You should see something similar to this:

Figure B-54: VueCentric dialog

B.17.8 Summary

You have learned how to create a service object and access that service within another component. The techniques for performing other programming tasks, such as making synchronous and asynchronous remote procedure calls and firing and receiving events, are identical to those used in creating visual components.

B.17.9 Creating Services with Visual Basic

This tutorial will demonstrate how to use Visual Basic to create a simple service object that implements a single method and how to access that service from within another component.

B.17.10 Creating the Project

To create the project for our service component, select **File | New Project** from Visual Basic's main menu. Select the **ActiveX DLL** project type and click **Open**.

Figure B-55: New Project dialog

This creates a project containing a single automation-compatible COM object named Class1 by default. Before continuing, let us rename our project and object to better reflect their function. In the project pane, select the project node and change its name in the property pane from Project1 to DemoService. Returning to the project pane, select the object node labeled Class1 and change its name in the property pane to DemoService2. This will provide a programmatic identifier for our service object of DemoService.DemoService2. Now save the project to a folder of your choice.

B.17.11 Accessing the Session Object

Since we will be making a remote procedure call, we need to access the Session object. This is done in the exact same manner as we did with the visual component we created earlier. To do this, select **Project | References...** from the main menu, locate and check the entry "CIA Component Support Services," and click **OK** to close the dialog.

Next, we need to declare a global variable to hold our reference to the Session Object. To do this, select (**General**) in the code editor and add the following code:

Now we need to store a reference to the Session object in our global variable, gSession. Select the **Class** entry in the left drop-down box of the code editor and its Initialize method in the right drop-down box. This will create an empty implementation for the Initialize method. Next, complete the implementation by adding the following code:

This code obtains a temporary reference to the Server object, retrieves a reference to its Session object, and releases the Server object. At this point, you now have a reference to the Session object stored in the global variable, gSession.

B.17.12 Modifying the Interface

Now we want to add a method to the interface of our service object. This will be a function that will receive one argument that specifies the type of information requested and will return the requested information as a string. Return to the code editor and add the following public function at the end as shown:

B.17.13 Registering the Service

Now we need to register the service. First, let us compile the project by selecting **File | Make DemoService.dll** from the menu. It should compile without errors. Next, we need to register our service to the **VueCentric** Framework. This is done in the same way we registered our visual component - using the **VueCentric** System Management Utility. Run the tool and login to the remote host. Select the **Object Registry** tab and in the **Restrict List To** box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework). The list of objects will refresh. Now search the list for the programmatic identifier of our newly created service, DemoService.DemoService2, and select that entry. In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the Name field with the display name for the control. The choice of name is arbitrary. We'll call it "Demo Service 2." Now switch to the special settings tab and check the box labeled **Service**. Now click **Apply** at the bottom. This completes the Framework registration process.

B.17.14 Accessing the Service

Now we need to provide a means to test the GetInfo method of our service object. To do this, we will modify the DemoControl we created in the earlier tutorial. Open the DemoControl project now (be sure to save any changes to the existing one). Add a reference to our service object by selecting **Project | References...** from the main menu. Find the entry "DemoService" in the list, check it, and click **OK** to close the dialog.

Next, create a global variable to hold the reference to our service object in the (General) section as shown:

```
Dim gService As DemoService2
Dim gHandle As Integer
Dim gPatient As CSS_Patient.Patient
Dim gSession As CIA_CSS.CSS_Session
```

Now initialize the gService variable in the Initialize method of the UserControl:

```
Private Sub UserControl_Initialize()
    Dim Server As New CIA_CSS.CSS_Server
    Set gSession = Server.Session
    Set Server = Nothing
    Set gPatient = gSession.FindServiceByProgID("CSS_PATIENT.PATIENT")
    gSession.EventSubscribe "STATUS", Me
    Set gService = gSession.FindServiceByProgID("DemoService.DemoService2")
End Sub
```

Finally, add a fifth button to the form and position it as desired. Rename the caption to “Demo Service.” Double-click the button and complete its Click event handler as shown:

```
Private Sub Command5_Click()
    Text1.Text = gService.GetInfo(2)
End Sub
```

Compile the project by selecting **File | Make DemoControl.ocx...** and load it in the Visual Interface Manager as before. Now click the **Demo Service** button. You should see something similar to this:

Figure B-56: VueCentric dialog

B.17.15 Summary

You have learned how to create a service object and access that service within another component. The techniques for performing other programming tasks, such as making synchronous and asynchronous remote procedure calls and firing and receiving events, are identical to those used in creating visual components.

B.18 Creating Services with C#

This tutorial will demonstrate how to use Visual Studio 2013 C# to create a simple service object that implements a single method and how to access that service from within another component.

B.18.1 Creating the Project

To create the project for our service component, select **File | New | Project** from Microsoft Visual Studio main menu. Select the **Class Library** under C#

Figure B-57: New Project window

This creates a project containing a single automation-compatible COM object named ClassLibrary1 by default. Before continuing, let us rename our project and object to better reflect their function. In the project pane, select the Name control entry change its name in the property pane from ClassLibrary1 to DemoService. Now save the project to a folder of your choice. Go to the Solution Explorer pane, select the object node labeled Class1 and change its name in the property pane to DemoService2. This will provide a programmatic identifier for our service object of DemoService.DemoService2.

B.18.2 Accessing the Session Object

Since we will be making a remote procedure call, we need to access the Session object. This is done in the exact same manner as we did with the visual component we created earlier. To do this, select **Project | Add Reference...** from the main menu, locate and check the entry "Interop.CIA_CSS," and click **OK** to close the dialog.

Next, we need to declare a local global variable to hold our reference to the Session Object. To do this, select the **Class Library (DemoService)** in the code editor and add the following code:

```
DemoService
DemoService2.cs
DemoService | DemoService.DemoService2 | Dispose()
public class DemoService2 : IDisposable
{
    CSS_Session _session;
100 %
```

Now we need to store a reference to the Session object in our global variable, _session. Select the Class entry in the left drop-down box of the code editor and its Initialize method in the right drop-down box. This will create an empty implementation for the Initialize method. Next, complete the implementation by adding the following code:

```
DemoService - DemoService2.cs
DemoService2.cs
DemoService | DemoService.DemoService2 | GetText()
    static CSS_Session _session;

    public static void Init ()
    {
        var svr = new CSS_Server();
        _session = svr.Session;
        Marshal.ReleaseComObject(svr);
    }
100 %
```

This code obtains a temporary reference to the Server object, retrieves a reference to its Session object, and releases the Server object. At this point, you now have a reference to the Session object stored in the global variable, _session.

B.18.3 Modifying the Interface

Now we want to add a method to the interface of our service object. This will be a function that will receive one argument that specifies the type of information requested and will return the requested information as a string. Return to the code editor and add the following public function at the end as shown:

B.18.4 Registering the Service

Now we need to register the service. First, let us compile the project by selecting **Build | Build DemoService** from the menu. It should compile without errors. Next, we need to register our service to the **VueCentric** Framework. This is done in the same way we registered our visual component - using the **VueCentric** System Management Utility. Run the tool and login to the remote host. Select the **Object Registry** tab and in the **Restrict List To** box, check **Local Registry** (this allows us to see local COM objects that are not yet registered to the Framework). The list of objects will refresh. Now search the list for the programmatic identifier of our newly created service, DemoService.DemoService2, and select that entry. In the upper right pane, click **Copy** to copy the local COM registration information into the VueCentric Settings pane. In that pane, fill in the Name field with the display name for the control. The choice of name is arbitrary. We'll call it "Demo Service 2." Now switch to the special settings tab and check the box labeled **Service**. Now click **Apply** at the bottom. This completes the Framework registration process.

B.18.5 Accessing the Service

Now we need to provide a means to test the GetInfo method of our service object. To do this, we will modify the DemoControl we created in the earlier tutorial. Open the DemoControl project now (be sure to save any changes to the existing one). Add a reference to our service object by selecting **Project | References...** from the main menu. Find the entry "DemoService" in the list, check it, and click **OK** to close the dialog.

Next, create a global variable to hold the reference to our service object in the (General) section as shown:

```
private DemoService2.DemoService _demoservice = null;
int Handle = 0;
private CSS_Session _session;
private readonly CSS_Patient.ICSS_Patient _patient;
```

Now initialize the _demoservice variable in the Initialize method of the UserControl:

```
public CCDACtl()
{
    InitializeComponent();
    var svr = new CSS_Server();
    _session = svr.Session;
    Marshal.ReleaseComObject(svr);
    _patient = _session.FindServiceByProgID("CSS_Patient.Patient") as ICSS_Patient;
    _service = _session.FindServiceByProgID("DemoService.DemoService2");
}
```

Finally, add a fifth button to the form and position it as desired. Rename the caption to "Demo Service." Double-click the button and complete its Click event handler as shown:

```
private void command5_Click(object sender, EventArgs e)
{
    textBox1.Text = _demoservice.GetInfo(2);
}
```

Compile the project by selecting **File | Build DemoControl.dll...** and load it in the Visual Interface Manager as before. Now click the **Demo Service** button. You should see something similar to this:

Figure B-58: VueCentric pane

B.18.6 Summary

You have learned how to create a service object and access that service within another component. The techniques for performing other programming tasks, such as making synchronous and asynchronous remote procedure calls and firing and receiving events, are identical to those used in creating visual components.

B.19 Deploying Components

Successful deployment of a component relies on a thorough understanding of version control and dependency control as implemented by the VueCentric Framework. In addition, the VueCentric SDK can greatly facilitate the task of creating server-side builds for your components.

B.20 Version Control

Proper version control is essential to ensuring that your components are updated properly and that the correct version of your component is loaded at run-time. The **VueCentric** Framework (specifically, the CMS) automatically recognizes that a newer version of a requested component is available and retrieves and installs it (so-called, just-in-time update). For this to work properly, a basic understanding of how this is done is necessary.

B.20.1 Version Numbers

Version numbers consist of up to four numbers separated by periods. These numbers represent, from left to right, the major, minor, release, and build versions. These numbers have a hierarchical relationship, so whenever a number is incremented, all numbers below it (i.e., to the right) should be reset to 0. Build numbers should be incremented every time a component is recompiled (most compilers can be configured to do this automatically). The guidelines for when to increment the major, minor, and release numbers are less clear. We typically increment the release number whenever we add or change functionality, but do not break compatibility with a previous version. We increment the minor version number when we break compatibility with a previous version. We increment the major version number when there is a major change to the functionality.

B.20.2 Which Version

COM allows associating version numbers with the type library and each imbedded interface and object. While this has certain advantages, COM makes no provision for supporting multiple versions of an object on a given machine. In addition, COM versioning has no applicability to files that are not COM objects. Therefore, a version control mechanism independent of COM is necessary. The CMS uses two different mechanisms for file versioning. For binary files that have an imbedded version resource, the CMS uses this information to determine the file version. Version resources are a standard means for imbedding version information within binary files. You may examine this resource in Windows by viewing a binary file's properties. If the binary file has a version resource, you will see a tab labeled Version and on that tab you will find the file version. Most compilers can be configured to include version information.

For non-binary files, the CMS uses the file's modification timestamp to generate a pseudo-version number. Like standard version numbers, this version number consists of four hierarchically arranged numbers. From left to right, these numbers are the year, month, day, and time. Because the timestamp uses universal time format, the version number generated in this manner is not sensitive to varying time zones.

B.20.3 Registering Version Information

When a component is requested, the CMS searches the application directory of the local machine to determine the version of the component currently residing there. It does this by directly examining the component's imbedded version resource. It then compares the local version number to the version number specified for the component in its VUECENTRIC OBJECT REGISTRY file entry. If the local version number is less than the one specified in the VUECENTRIC OBJECT REGISTRY file, or if the component was not found in the application directory, it is retrieved from the object repository and installed in the application directory.

Note that the CMS does not directly examine the file in the object repository to determine its version. This means that the version entered in the VUECENTRIC OBJECT REGISTRY file must truly reflect the version residing in the object repository for the update mechanism to work. The reason the CMS does not directly examine files in the object repository is twofold. First, if the repository resides on a remote share, examining a file's version resource can be very time consuming (in fact, Windows makes a local copy of the file to do this). Second, if the repository resides on a web or ftp server, there is no means to examine the file directly without downloading it first. Therefore, be certain that you correctly update the version information for your components.

B.20.4 Side-by-Side Versioning

Side-by-side versioning refers to the ability to have multiple versions of the same component residing on a machine at the same time. When most COM objects are registered, the full path to the executable is included in the Windows registry. Because this registration overwrites any previous entry, it is not possible to register more than one version at a time. Thus, the default behavior of COM is to share a single copy of an object across all applications. This is rarely desirable, especially in the situation where an object is not backward-compatible with previous versions. To overcome this limitation, Microsoft has offered three possible solutions:

- Store only the object's file name, not its path information, in the Windows registry. In the absence of path information, Windows will search for the file in the application directory first, then in the System and Windows directories. In this manner, one can partition different object versions in the respective application directories and be certain that the correct one is loaded. Modifying the default COM registration behavior can be done in one of two ways. Since COM objects register themselves through the DLLRegisterServer method, one can override the default implementation of this method and modify the registration process. A second option is to set the SIDE-BY-SIDE field of the corresponding VUECENTRIC OBJECT REGISTRY file entry to true. When this field is true, the CMS inspects the Windows registry entry for the COM object and removes path information if it exists. This allows forcing side-by-side versioning on an object-by-object basis.
- Microsoft offers a second solution to the versioning problem. If the COM subsystem detects a file with the same name as the main application with ".local" appended to it, it will ignore all path information in the Windows registry. For example, if the file VIM.exe.local is placed in the application directory, all objects loaded by the VIM application will be treated as if they had been registered without path information. This option has three disadvantages. First, it is all-or-nothing as far as the application is concerned. Second, this capability only applies to versions of Windows including and subsequent to Windows 98 SE. Third, in the presence of an application manifest (see below), this feature is disabled.

- The third, and most recent, solution to versioning is in the form of an application manifest, which may either be imbedded in the application or present in the form of a file with the same name as the application with a ".manifest" appended to it. The purpose of the manifest is to allow the application developer to specify which version of a particular component is to be used. In contrast to the ".local" file described above which affects the search behavior for all components, the manifest affects search behavior only for those components listed. Furthermore, the use of a manifest precludes the use of the ".local" file (except in Windows 2000 environments where the manifest is not supported). The EHR uses a manifest to enable theme support for the application.

The choice of versioning techniques depends on many factors. Because of the multiple run-time environments supported by the EHR, all three techniques are utilized to ensure the correct components are used for a given application instance.

For a detailed treatment of this subject, see the MSDN article titled *Implementing Side-by-Side Component Sharing in Applications* (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsetup/html/sidebyside.asp).

B.20.5 Imbedding Version Information

Both Delphi and Visual Basic can imbed version information in the binary files they generate.

To include version information in Delphi projects, choose the **Project | Options** menu. From the Project Options dialog that appears, select the **Version Info** tab. Select the **Include version information in project** and **Auto-increment build number** check boxes. You may optionally fill in other information located at the bottom of the dialog.

Figure B-59: Project Options dialog

To imbed version information using Visual Basic, select the **Project | Properties** menu. From the **Project Properties** dialog that appears, select the **Make** tab. Select the **Auto Increment** check box. You may optionally fill in other version information as desired.

Figure B-60: Project Properties dialog

B.21 Handling Dependencies

It is not at all uncommon that a component depends upon the presence of other components or files to function. So how does one insure that all of the necessary pieces are in place when a component is executed? There are several possible approaches to this problem.

If a given file is required by many components, it may make sense to deploy that file using the same mechanism to deploy the core Framework. VueCentric provides a configurable installer utility that installs and updates core components from a shared directory, independent of the Framework. Alternatively, if the core files are installed using a third-party installer, one could add the required file(s) to the installation.

The VUECENTRIC OBJECT REGISTRY file provides two options for handling dependent files. Using the VueCentric System Management Utility, one can register dependencies between entries in this file in the Dependencies section of the Object Registry tab. When an object is requested, all listed dependencies are requested automatically. This method has the advantage of applying version control over each dependency. In addition, if a dependent file is marked as a service, the service is automatically started. Dependencies are recursive so that any dependencies listed for a dependent file are also requested.

The Required Files section of the Object Registry tab provides an alternate means for declaring dependencies. Whenever an object is installed or updated, any required files listed here are also retrieved. This has the advantage of not requiring that these dependent files be entered into the VUECENTRIC OBJECT REGISTRY file. The main disadvantage is that no version control is applied to these files. They are only retrieved when the object needs to be installed or updated.

B.22 Generating KIDS Builds

All components require some kind of installation on the remote host. At a minimum, a component requires registration information to be entered into the VUECENTRIC OBJECT REGISTRY file. Most will also require supporting code and remote procedure declarations. Some will also define events and parameters. To facilitate the delivery of these required elements to the remote host, the VueCentric SDK augments the KIDS system by providing a means to easily package these elements into the build and also provides support for common tasks such as registering remote procedures.

The VueCentric SDK is delivered as a KIDS build. Once installed, the SDK provides a template for creating component builds and a configuration file for controlling the behavior of the build. The template is a KIDS build called VUECENTRIC DUMMY. Do not modify this build directly. Rather, create a copy of the build and modify it. This build is delivered with pre-installation, post-installation, and pre-transportation methods that should be modified. In addition, it is also configured to deliver entries from several key files based on information in the configuration file. You may add additional files to the build, but you should not remove the existing entries. You may also add any additional elements that are to be delivered (e.g., remote procedures, options, etc.).

The configuration file delivered with the SDK is called the VUECENTRIC DISTRIBUTION file. By associating an entry with your component build created from the VUECENTRIC DUMMY template, you may control the elements delivered with your build. The VUECENTRIC DISTRIBUTION file has the following fields:

Field	#	Datatype	Description
NAME	.01	Text	This is typically the display name given to the component.
BUILD	.5	Pointer (#9.6)	This is the build with which this entry is to be associated.
OBJECT (multiple)	1	Pointer (#19930.2)	Any VUECENTRIC OBJECT REGISTRY entries to be included in the build.
PARAMETER DEFINITION (multiple)	2	Pointer (#8989.51)	Any parameter definitions to be included in the build. The subfile also has fields for initializing parameter values.

Field	#	Datatype	Description
PARAMETER TEMPLATE (multiple)	3	Pointer (#8989.52)	Any parameter templates to be included in the build.
TEMPLATE (multiple)	4	Pointer (#19930.3)	Any VUECENTRIC TEMPLATE REGISTRY entries to be included in the build.
EVENT(multiple)	5	Pointer (#19941.21)	Any CIA EVENT TYPE entries to be included in the build.
PREINIT CODE	50	M Code	Any code to be executed during the pre-installation phase.
POSTINIT CODE	51	M Code	Any code to be executed during the post-installation phase.
PRETRANS CODE	52	M Code	Any additional pretransportation code to be executed when generating the build.
COMMENTS	99	Word Processing	Used for documentation purposes.

When you generate your build, you will see the message "Target distribution: <name>" where <name> is the name of the associated entry in the VUECENTRIC DISTRIBUTION file. If more than one entry is associated with the build, you will be prompted to select the one to use.

B.23 Pitfalls and Special Techniques

This section discusses potential pitfalls that the component developer may encounter and special techniques that will facilitate component development.

B.23.1 Component Initialization

Both Delphi and Visual Basic define an Initialize method for its ActiveX controls. This method is executed when an object is first created and should be used to perform any necessary initializations. In Delphi, this is preferred over overriding the object's constructor since ActiveX objects typically have more than one constructor and the Initialize method is guaranteed to be executed regardless of which constructor is invoked.

B.23.2 Component Destruction

Under Delphi, accessing a component's COM interface within the destructor may cause a stack overflow. This happens because the destructor has been invoked because the object's reference count has reached zero. When the object's COM interface is referenced, the reference count increments and then decrements back to zero, causing re-entry of the destructor. If there is a need to access an object's COM interface in its destructor, first make a call to the object's _AddRef method. This increments the reference count and ensures that it will not return to zero. Do not call the _Release method since the object is already in the process of being released and doing so would cause the reference count to return to zero.

B.23.3 Other Containers

Because visual components are ActiveX-compliant, any visual component can potentially be hosted in any ActiveX-compliant container (e.g., Internet Explorer). Any component used in this manner should register itself with the CSS by calling the Session object's RegisterObject method. If this is not done, the component will not be notified of context changes nor will it be accessible to other components.

B.23.4 Focus Issues

ActiveX controls created with Visual Basic have one anomaly associated with control focus. The first mouse click a Visual Basic ActiveX control sets focus to the control. Once the control has focus, subsequent mouse clicks perform as expected. This means, for example, that if you click a button on a Visual Basic ActiveX control that does not yet have focus, the click will not generate a button press event. Subsequent clicks will.

One solution to this anomaly is to implement the MouseMove method for the UserControl and make a call to SetFocus in the implementation. This causes the control to automatically receive focus when the mouse moves over it.

B.23.5 Deferring Data Fetches

Retrieving data from the remote host can be time consuming and hamper application performance. This burden can be lessened by devising intelligent strategies for deferring data fetches until the data is actually needed. One common strategy is for a component to defer populating its display until it becomes visible. For example, it makes little sense to retrieve a patient's problem list data into a component immediately after a patient context change if the user never views that component. A useful technique for deferring data fetches in this scenario would be to set a flag indicating that a data fetch is required when the context change occurs and then invalidate the component's main window. In the component's painting logic, one could check for this flag and perform the fetch if it is set. By invalidating the component's main window, one insures that if the component is already visible, the fetch will occur immediately. Otherwise, the fetch will occur only if the component becomes visible.

B.23.6 Intercomponent Communication

Intercomponent communication refers to the technique of one component communicating information to a second component in the application. This can be very useful if the action of one component affects another or if one component needs to make use of a service provided by another.

The VueCentric Framework provides two methods for implementing intercomponent communication. The first uses local events to send information to local subscribers. This method is useful when a component wants to communicate information to multiple targets. It has the disadvantage of being unidirectional and anonymous.

A second method provides a much more tightly coupled communication link between components. This method utilizes a technique called dynamic discovery to locate other components in the environment and establish an interface reference to them. The Session object provides a number of methods that support dynamic discovery. They are divided into two groups, one for discovering visual components and another for discovering services. Each returns a reference to the IUnknown interface of the requested object which may then be cast to the desired interface. Using this reference, an object can then invoke any method or property on the interface. The methods are:

FindObjectByCLSID – This function searches the list of registered objects to find one that implements the class identified by CLSID. If the Last parameter is not nil (Nothing), the search begins following that object's entry in the list. In this manner, one can iterate through multiple object instances of the same class.

FindObjectByIID – This function searches the list of registered objects to find one that implements the interface identified by IID. If the Last parameter is not nil (Nothing), the search begins following that object's entry in the list. In this manner, one can iterate through all objects implementing a particular interface.

FindObjectByProgID – This function searches the list of registered objects to find one that possesses the programmatic identifier specified by ProgID. If the Last parameter is not nil (Nothing), the search begins following that object's entry in the list. In this manner, one can iterate multiple object instances of the same class.

FindServiceByCLSID – Request a reference to the service identified by CLSID. If the service is not already running, the CSS starts the service. If the service is not located, a nil (Nothing) value is returned. Otherwise, the return value is a reference to the service's default interface.

FindServiceByProgID – Request a reference to the service identified by ProgID. If the service is not already running, the CSS starts the service. If the service is not located, a nil (Nothing) value is returned. Otherwise, the return value is a reference to the service's default interface.

B.23.7 Creating Trace Log Entries

Component authors may create entries in the trace log for debugging purposes using the API suite provided by the session object of the CSS. This suite consists of the following:

Method or Property	Return Type	Description
TraceMode	Boolean	This property indicates whether or not trace mode is active. If trace mode is not active, calling any of the trace methods will have no effect. While it is not required to check the state of trace mode before invoking one of the trace methods, it is generally advisable to do so to avoid the overhead of sending trace information when trace mode is inactive.
TraceBegin(TraceClass, TraceType)	Integer	Call this method to initiate a trace log entry. TraceClass and TraceType determine how the entry is displayed in the trace log viewer. TraceClass defines an overall category for the log entry and TraceType defines a subcategory within the TraceClass. This method returns a handle that uniquely identifies the log entry being created.
TraceAdd(Handle,Value, IsHeader)	none	Call this method to add to the newly created log entry. Handle refers to the unique handle returned by the TraceBegin method. Value is the text to be added to the entry. If IsHeader is true, this indicates that the Value represents header information. When displayed in the log viewer, headers are centered, underlined, and red in color.
TraceEnd(Handle)	none	Call this method to complete the entry. Once this is done, the CSS fires a TRACE event which is intercepted by the log viewer and entered into the trace log.

Consider the following Delphi code example:

```
var
  Handle: Integer;
begin
  if vcSession.TraceMode
  then begin
    Handle := vcSession.TraceBegin('RPC', 'BEHOPTCX LAST');
    vcSession.TraceAdd(Handle, 'Parameters', True);
    vcSession.TraceAdd(Handle, '#1 : 733', False);
    vcSession.TraceAdd(Handle, 'Results', True);
    vcSession.TraceAdd(Handle, '733', False);
    vcSession.TraceEnd(Handle);
end;
```

This code would generate a trace log entry that would appear as follows in the trace log viewer:

Figure B-61: Trace Log pane

B.23.8 Embedding Licensed Controls

The use of licensed third-party controls in the development of a component sometimes produces an unauthorized license exception when it is imbedded within another component. This is a problem that has been reported with ActiveX controls produced in Visual Basic that contain licensed controls on the primary form. It is unclear whether this is a “feature” or a “bug,” but it has received much discussion in the technical newsgroups. It is not a container issue, but rather a problem with the way Visual Basic passes license information to the imbedded control. Evidently, for the primary form this does not occur. Interestingly, for secondary forms it does. Therefore, imbedding a licensed component on a secondary form causes no problems. This phenomenon lends itself to an interesting, albeit somewhat inelegant, workaround:

Create a dummy form in your Visual Basic project. Place one copy of each licensed component that you will be imbedding in your primary form onto the dummy form. Load the dummy form in the Initialize method of your control. This registers the license information for the controls. Unload the dummy form in the very next statement (it is no longer needed). The licensed components will now function properly on the primary form.

B.23.9 Forced Context Changes

While context changes are a mostly democratic process, there are two situations where a context change may be forced even when one of the participants rejects the request. The first situation occurs during a forced shutdown of the application when the context of each context object is cleared. The second situation occurs when a CCOW client requests a context change and elects to override a participant's objection. In either event, the programmer should be prepared to react to a forced context change and not assume that a rejection of the request will always abort the change.

B.23.10 Integrating Help Content

The Visual Interface Manager interrogates each visual component as it is loaded to determine which ones provide on-line help. It then provides access to the help documentation by means of the **Help | Help On** menu. The VIM attempts to acquire three pieces of information when it interrogates a component: the help file name, the display name of the component, and the help context identifier. It first examines the type library to determine if a Help File attribute is defined for the type library and Help String and Help Context attributes are defined for the component's default interface. If these attribute values are found, they are used for the help file name, display name, and help context identifier, respectively.

For example, the type library shown below defines a Help File attribute of "vcPatientID.chm," a Help String attribute of "Patient Selection," and a Help Context attribute of 124.

Figure B-62: Type Library pane

Figure B-63: Type Library pane

If the VIM does not obtain the required information from the type library, it then examines the default interface of the component for a HelpFile and HelpContext property. If it finds these, it uses those values for the help file name and help context identifier, respectively, and uses the component's Name property from the VUECENTRIC OBJECT REGISTRY file as the display name.

If the VIM can obtain a value for the component's help file name by one of these methods, that component's display name will appear as a submenu under the **Help | Help On** menu. Invoking the submenu will load the specified help file (both Windows and HTML help formats are supported) at the point referenced by the context identifier (if found).

B.24 Delphi Helper Functions

```
unit vcWrappers;

{=========================================================================
==========
  Helper Functions

  This unit contains wrappers for certain calls to the Session that
simplify the task of converting Delphi parameters to COM-compliant
parameters.

  Declarations        Description
  ---------------------------------------------------------------------
----------
```

```
  TMultList           TStringList descendant that supports specifying
subscript values when storing data.
  TWPList             TStringList descendant that passes its subscripts
values in word processing format.

  Modification History
  ---------------------------------------------------------------------------
----------

  Copyright © 2001-2003 by Clinical Informatics Associates, Inc.  All
Rights Reserved.
==============================================================================
=========}

interface

uses
  Classes, SysUtils, ActiveX, Variants, CIA_CSS_TLB;

type
  TMultList = class(TStringList)    // Used to pass explicitly subscripted
data to broker
  public
    function Add(const Subscript, Data: String): Integer; reintroduce;
overload;
    function Add(const Subscript: String; Data: Extended): Integer;
reintroduce; overload;
    function Add(const Subscript: String; Data: Integer): Integer;
reintroduce; overload;
    function Add(const Subscript: array of const; Data: String): Integer;
reintroduce; overload;
    function Add(const Subscript: array of const; Data: Extended): Integer;
reintroduce; overload;
    function Add(const Subscript: array of const; Data: Integer): Integer;
reintroduce; overload;
  end;

  TWPList = class(TStringList);     // Used to force WP formatting of
broker data

function CallRPCList(const Session: ICSS_Session; const RPCName: String;
const Args: array of const; Retlist: TStrings): TStrings; overload;
function CallRPCInt(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): Integer; overload;
function CallRPCStr(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): String; overload;
function CallRPCBool(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): Boolean; overload;
function CallRPCDate(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): TDateTime; overload;
function CallRPCAsync(const Session: ICSS_Session; const RPCName: String;
const Args: array of const; const Callback: ICSS_SessionEvents): Integer;
overload;
procedure CallRPC(const Session: ICSS_Session; const RPCName: String; const
Args: array of const); overload;

function ConstToVar(const Args: array of const): OleVariant;
function ListToVar(Strings: TStrings): OleVariant;

implementation

uses
```

```
  MFunctions, Utility;

const
  SUBSCRIPT_DELIMITER = #1;

function CallRPCList(const Session: ICSS_Session; const RPCName: String;
const Args: array of const; Retlist: TStrings): TStrings;
var
  DidCreate: Boolean;
begin
  try
    ShowBusy;
    DidCreate:=RetList=nil;
    if DidCreate
    then RetList:=TStringList.Create;

    try
      Retlist.CommaText:=Session.CallRPCList(RPCName,ConstToVar(Args));
    except
      if DidCreate
      then FreeAndNil(RetList);
      raise;
    end;

  finally
    Result:=RetList;
    ShowBusy(False);
  end;
end;

function CallRPCInt(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): integer;
begin
  try
    ShowBusy;
    Result:=Session.CallRPCInt(RPCName,ConstToVar(Args));
  finally
    ShowBusy(False);
  end;
end;

function CallRPCStr(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): String;
begin
  try
    ShowBusy;
    Result:=Session.CallRPCStr(RPCName,ConstToVar(Args));
  finally
    ShowBusy(False);
  end;
end;

function CallRPCBool(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): Boolean;
begin
  try
    ShowBusy;
    Result:=Session.CallRPCBool(RPCName,ConstToVar(Args));
  finally
    ShowBusy(False);
  end;
end;
```

```
function CallRPCDate(const Session: ICSS_Session; const RPCName: String;
const Args: array of const): TDateTime;
begin
  try
    ShowBusy;
    Result:=Session.CallRPCDate(RPCName,ConstToVar(Args));
  finally
    ShowBusy(False);
  end;
end;

function CallRPCAsync(const Session: ICSS_Session; const RPCName: String;
const Args: array of const; const CallBack: ICSS_SessionEvents): Integer;
begin
  try
    ShowBusy;
    Result:=Session.CallRPCAsync(RPCName,ConstToVar(Args),CallBack,False);
  finally
    ShowBusy(False);
  end;
end;

procedure CallRPC(const Session: ICSS_Session; const RPCName: String; const
Args: array of const);
begin
  try
    ShowBusy;
    Session.CallRPCStr(RPCName,ConstToVar(Args));
  finally
    ShowBusy(False);
  end;
end;

function ConstToVar(const Args: array of const): OleVariant;
{ This converts an array of constants to a variant array of variants
suitable
  for passing to a COM object.  Note that the variant array is locked and
  referenced via a pointer to improve performance.
}
type
  TVarArray = array [0..$FFFFFF] of OleVariant;
  PVarArray = ^TVarArray;
var
  i: integer;
  p: PVarArray;
begin
  if High(Args)=-1
  then begin
    Result:=Unassigned;
    exit;
  end;

  Result:=VarArrayCreate([0,High(Args)],varVariant);
  p:=VarArrayLock(Result);

  try
    for i:=0 to High(Args) do
      with Args[i] do begin
        p^[i]:=Unassigned;
        case VType of
          vtInteger:    p^[i]:=VInteger;
```

```
          vtBoolean:    p^[i]:=Ord(VBoolean);
          vtChar:       p^[i]:=VChar;
          vtExtended:   p^[i]:=VExtended^;
          vtString:     p^[i]:=VString^;
          vtPChar:      p^[i]:=VPChar^;
          vtObject:     if VObject is TStrings
                        then p^[i]:=ListToVar(TStrings(VObject));
          vtWideChar:   p^[i]:=VWideChar;
          vtPWideChar:  p^[i]:=VPWideChar^;
          vtAnsiString: p^[i]:=AnsiString(VAnsiString);
          vtCurrency:   p^[i]:=VCurrency^;
          vtVariant:    p^[i]:=VVariant^;
          vtInterface:  p^[i]:=IUnknown(VInterface);
          vtWideString: p^[i]:=WideString(VWideString);
          vtInt64:      p^[i]:=IntToStr(VInt64^);
        end;
        if VarIsEmpty(p^[i])
        then raise Exception.Create('Unrecognized datatype');
      end;
  finally
    VarArrayUnlock(Result);
  end;
end;

function ListToVar(Strings: TStrings): OleVariant;
{ Convert a TStrings to a variant array with base subscript starting at
one.
  Passes data in format <subscript>~<value>.
  If the TStrings list is empty, returns a null string instead.
  If the TStrings is a TMultList, this call assumes the entries are already
  formatted.  Otherwise, the data are automatically formatted.
}
type
  TListType = (ltMult, ltWP, ltOther);
  TVarArray = array [0..$FFFFFF] of WideString;
  PVarArray = ^TVarArray;
var
  i: Integer;
  p: PVarArray;
  ListType: TListType;
begin
  if Strings.Count=0
  then Result:=''
  else try
    if Strings is TMultList
    then ListType:=ltMult
    else if Strings is TWPList
    then ListType:=ltWP
    else ListType:=ltOther;

    Result:=VarArrayCreate([1,Strings.Count],varOleStr);
    p:=VarArrayLock(Result);
    for i:=0 to Strings.Count-1 do
      case ListType of
        ltMult:   p^[i]:=WideString(Strings[i]);
        ltWP:
p^[i]:=WideString(IntToStr(i+1)+',0'+SUBSCRIPT_DELIMITER+Strings[i]);
        ltOther:
p^[i]:=WideString(IntToStr(i+1)+SUBSCRIPT_DELIMITER+Strings[i]);
      end;
  finally
    VarArrayUnlock(Result);
```

```
  end;
end;

{ TMultList is used to pass explicitly subscripted data to the broker as a
string
  list.  Entries in the string list should be of the format:
<subscript>~<data>
}

function TMultList.Add(const Subscript, Data: String): Integer;
{ This is an overloaded version of the TStrings Add function which is
provided
  to hide the underlying representation of the subscripted data and to
perform
  validity checks on the subscript.  An exception is raised if the
subscript is
  null or contains the subscript delimiter.
}
begin
  if Subscript=''
  then raise Exception.Create('Null subscript in RPC call')
  else if Pos(SUBSCRIPT_DELIMITER,Subscript)<>0
  then raise Exception.Create('Invalid subscript in RPC call: '+Subscript);

  Result:=Add(Subscript+SUBSCRIPT_DELIMITER+Data);
end;

function TMultList.Add(const Subscript: String; Data: Integer): Integer;
{ Overloaded form of above to support integer datatypes.
}
begin
  Result:=Add(Subscript,IntToStr(Data));
end;

function TMultList.Add(const Subscript: String; Data: Extended): Integer;
{ Overloaded form of above to support floating point datatypes.
}
begin
  Result:=Add(Subscript,FloatToStr(Data));
end;

function TMultList.Add(const Subscript: array of const; Data: String):
Integer;
{ Overloaded form of above to support string datatypes and subscript passed
as array.
}
begin
  Result:=Add(BuildSubscript(Subscript),Data);
end;

function TMultList.Add(const Subscript: array of const; Data: Extended):
Integer;
{ Overloaded form of above to support floating point datatypes and
subscript passed as array.
}
begin
  Result:=Add(BuildSubscript(Subscript),Data);
end;

function TMultList.Add(const Subscript: array of const; Data: Integer):
Integer;
```

```
{ Overloaded form of above to support integer datatypes and subscript
passed as array.
}
begin
  Result:=Add(BuildSubscript(Subscript),Data);
end;

end.
```

B.25 Visual Basic Helper Functions

```
Attribute VB_Name = "RPCUtilities"
Option Explicit

'==============================================================
' This function creates an RPC parameter list (variant array)
' from any number of arguments or data types passed to it.
'
' For example, the following code creates an RPC parameter
' list from 5 totally different data types, then calls
' CallRPCList.
'   dim params as variant
'   params = CreateRPCParamList("abc", 1, today, 4/5, x)
'   result = gSession.CallRPCText("Some RPC Name", params)
'==============================================================
Public Function CreateRPCParamList(ParamArray params() As Variant) As
Variant
    ' Passed parameters are automatically converted to variants array
members
    CreateRPCParamList = params
End Function

'==============================================================
' This function creates a single, pipe-delimited RPC parameter
' string from any number of arguments or data types passed to it.
'
' For example:
'   dim params as string
'   params = CreateRPCParamString("abc", 1, today, 4/5, x)
'   result = gSession.CallRPCText("Some RPC Name", params)
'==============================================================
Public Function CreateRPCParamString(ParamArray params() As Variant) As
String
    Dim i As Integer, result As String

    result = ""
    For i = 0 To UBound(params)
        If i > 0 Then result = result & "|"
        result = result & params(i)
    Next i

    CreateRPCParamString = result
End Function

'==============================================================
' These helper functions makes it easier to parse data that is
' passed back from CallRPCText and CallRPCList.  It parses the
' string using the appropriate delimiter, producing a string
' array that can then be accessed directly.
'
```

```
' For Example:
'   dim result as string, ResultArray() as string
'   result = gSession.CallRPCText("Some RPC Name", params)
'   ResultArray=ParseRPCText(result)
'
'   msg = UBound(ResultArray) + 1 & " results returned:" & vbCrLf
'   For i = LBound(ResultArray) To UBound(ResultArray)
'       msg = msg & vbCrLf & ResultArray(i)
'   Next i
'==============================================================

Public Function ParseRPCText(str As String) As String()
   ParseRPCText = Split(str, vbCrLf)
End Function

Public Function ParseRPCList(str As String) As String()
    Dim QuoteChar As String, tmp As String
    Dim quote As Integer, quote2 As Integer, comma As Integer
    Dim count As Integer, ResultArray() As String
    Dim piece As String, index As Integer

    tmp = str & ","
    QuoteChar = Chr(34)
    count = 0

    Do Until tmp = ""
        comma = InStr(1, tmp, ",")
        quote = InStr(1, tmp, QuoteChar)
        If quote = 0 And comma = 0 Then
            piece = tmp
            tmp = ""
        ElseIf quote < comma Then
            quote2 = InStr(quote + 1, tmp, QuoteChar)
            If quote2 = 0 Then quote2 = Len(tmp) + 1
            piece = Mid(tmp, quote + 1, quote2 - quote - 1)
            If Mid(tmp, quote2 + 1, 1) = "," Then quote2 = quote2 + 1
            tmp = Mid(tmp, quote2 + 1)
        Else
            piece = Left(tmp, comma - 1)
            tmp = Mid(tmp, comma + 1)
        End If

        index = count
        count = count + 1
        ReDim Preserve ResultArray(index)
        ResultArray(index) = Trim(piece)
    Loop
    ParseRPCList = ResultArray
End Function

Public Function ParseRPCResult(str As String, delim As String) As String()
    ' This function is an expanded version of the previous ParseRPC
routines
    ' and allows the developer to specify the delimiting character.
    ParseRPCResult = Split(str, delim)
End Function
```

Glossary

Application Context

This is an entry in the Option file that provides access control at the client application level. The Broker passes this information on the initial connection request. A user must have access to the given option to establish a broker connection.

Broker

Software that marshals remote procedure requests between a client application and a remote host. Also known as RPC Broker.

Common Context Object Workgroup

An HL7-sponsored workgroup responsible for specifying standards for context exchange among applications.

Context Object

A specialized service that maintains a common context for a specific entity (e.g., patient or encounter) and supports the context change event interface.

Daemon

A background process that performs a specified service.

GPRA

Government Performance and Results Act.

Listener

A daemon that monitors a specified TCP port and handles communications between the client and remote host.

Primary Listener Daemon

The daemon that listens for the initial connection request. The primary listener daemon immediately passes the connection to a secondary listener daemon.

Remote Procedure

A procedure residing on a remote host that may be invoked by a client process through a broker intermediary.

Remote Procedure Call

The act of invoking a remote procedure. This is often used interchangeably with the term “Remote Procedure,” especially using its abbreviated form “RPC.”

RPC Context

This is an entry in the Option file that provides access control at the remote procedure level. The Broker passes this information with each remote procedure request. A user must have access to the given option to invoke the associated remote procedure.

Secondary Listener Daemon

The daemon that handles communication interchange between the client and remote host processes.

Session Context

This refers to the persistent state information associated with an establish session.

Acronym List

Acronym	Meaning
API	Application Programming Interface
CCOW	Clinical Context Object Workgroup
CMS	Component Management Service
CSL	Communication Service Layer
CSS	Component Support Services
GUID	Global Unique Identifier
IHS	Indian Health Service
RPC	Remote Procedure Call
RPMS	Resource and Patient Management System
VIM	Visual Interface Manager

Contact Information

If you have any questions or comments regarding this distribution, please contact the IHS IT Service Desk.

Phone: (888) 830-7280 (toll free)

Web: http://www.ihs.gov/helpdesk/

Email: itsupport@ihs.gov