



IBM MQSeries Workflow for OS/390

Programming Guide

Version 3 Release 2



IBM MQSeries Workflow for OS/390

Programming Guide

Version 3 Release 2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xiii.

Second Edition (December 1999)

This edition applies to Version 3, Release 2, Modification Level 1 of IBM MQSeries Workflow for OS/390 (product number 5565-A96) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for your comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

FAX (Germany): 07031+16-3456
FAX (Other Countries): (+49)+7031-16-3456

IBM Mail Exchange: DEIBMBM9 at IBMMAIL
Internet: s390id@de.ibm.com

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1999. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables ix

About this book xi

Who should read this book xi

How to get additional information xi

How to send your comments xi

How this book is organized xi

How to read the syntax diagrams xii

Notices xiii

Trademarks xv

Summary of changes xvii

Chapter 1. MQSeries Workflow programming concepts 1

Understanding Workflow programming 1

 The role of the programmer in modeling a process 1

Programming interfaces 2

Prerequisites for using a programming language API 3

Overview of the Runtime API 3

Building an MQSeries Workflow application 7

 Overview 7

 Handling errors 8

Object and memory management 11

The result object 12

Client/server communication and data access

models 16

 Synchronous client/server communication 16

 Asynchronous client/server communication 16

 The push data access model 16

 Receiving information 17

An MQSeries Workflow session 19

Querying data 19

 Persistent lists 20

 Using filters, sort criteria, and thresholds 20

 Handling collections 20

 C and COBOL vector accessor functions 21

 Java arrays 30

Handling containers 30

 Data structure/container type 30

 Data member/container element 30

 Predefined data members 32

 Determining the structure of an unknown

 container 37

 Analyzing a container element 41

 Accessing a known container element 47

 Accessing a value of a container 47

 Accessing a value of a container element 54

 Setting a value of a container 60

 Return codes/FmcException 65

Monitoring a process instance 65

 Obtaining a process instance monitor 66

 Ownership of monitors 67

Authorization considerations 67

Types of API calls 70

 Basic API calls 70

 Accessor API calls 84

 Action API calls 121

 Activity implementation API calls 121

Chapter 2. Language interfaces 123

C and C++ interface 123

 Coding an MQSeries Workflow client

 application in C or C++ 123

 Coding an MQSeries Workflow activity

 implementation in C or C++ 124

 Compiling and linking 125

Java interface 126

 The Java CORBA Agent 126

 The communication layer 127

 The locator methods 127

 The Java API Beans 128

 Coding an MQSeries Workflow client

 application in Java 129

 Coding an MQSeries Workflow activity

 implementation in Java 130

 Compiling 132

 Object management 132

 Garbage collection when using Java API Beans 132

COBOL interface 133

 Calling the API 133

 String handling 133

 Coding an MQSeries Workflow client

 application in COBOL 134

 Coding an MQSeries Workflow activity

 implementation in COBOL 135

 Compiling and linking 136

 Mapping C to COBOL data types 137

 Name changes between COBOL and C 138

 Example of the use of strings 148

XML message interface 149

 The MQSeries Workflow message 149

 Sending requests to MQSeries Workflow 152

 Invoking an activity implementation 153

 The MQSeries Workflow XML message format 157

Chapter 3. Interfacing with the Program Execution Server 163

CICS considerations 163

IMS considerations 163

Program mapping via the Program Execution

Server 163

 Introduction 163

 Program mapping definitions 165

 Mapping algorithm 168

 Supported program mapping definition element

 types 173

| | |
|---|-----|
| Grammar | 177 |
| Usertype | 188 |
| Size of program mapping interface definition elements | 190 |
| Activation of program mapping definitions | 191 |
| Troubleshooting | 192 |
| Additional mapping examples. | 192 |
| Program execution server exits | 198 |
| Introduction | 198 |
| Interfaces for all exits. | 199 |
| Special considerations for exit programming | 201 |
| Program mapping exit | 201 |
| Program invocation exit | 205 |

Chapter 4. API classes and objects 215

| | |
|---|-----|
| Summary | 215 |
| API calls by class | 217 |
| ActivityInstance | 217 |
| ActivityInstanceNotification | 221 |
| ActivityInstanceNotificationVector | 222 |
| ActivityInstanceVector | 223 |
| Agent | 223 |
| BlockInstanceMonitor | 224 |
| Container. | 225 |
| ContainerElement | 228 |
| ContainerElementVector | 230 |
| ControlConnectorInstance | 231 |
| ControlConnectorInstanceVector | 232 |
| Date and Time (FmcDateTime/FmcjCDateTime/Calendar) | 232 |
| DllOptions | 233 |
| ExecutionAgent | 233 |
| ExecutionData | 234 |
| ExecutionService | 235 |
| ExeOptions | 237 |
| ExternalOptions | 238 |
| FmcError/FmcjError | 240 |
| FmcException | 241 |
| Global. | 241 |
| ImplementationData | 242 |
| Item | 243 |
| ItemVector | 245 |
| Message | 245 |
| PersistentList | 246 |
| Person. | 247 |
| Point | 250 |
| PointVector | 251 |
| ProcessInstance. | 251 |
| ProcessInstanceList | 255 |
| ProcessInstanceListVector | 255 |
| ProcessInstanceMonitor | 255 |
| ProcessInstanceNotification. | 256 |
| ProcessInstanceNotificationVector. | 256 |
| ProcessInstanceVector | 256 |
| ProcessTemplate | 257 |
| ProcessTemplateList | 259 |
| ProcessTemplateListVector | 260 |
| ProcessTemplateVector | 260 |
| ProgramData | 260 |
| ProgramTemplate | 261 |
| ReadOnlyContainer | 263 |

| | |
|------------------------------|-----|
| ReadWriteContainer | 263 |
| Result | 265 |
| Service | 265 |
| StringVector | 266 |
| SymbolLayout | 267 |
| WorkItem | 268 |
| WorkItemVector | 270 |
| Worklist | 270 |
| WorklistVector | 271 |

Chapter 5. API action and activity implementation calls 273

| | |
|--|-----|
| ActivityInstance actions | 273 |
| ObtainProcessInstanceMonitor() | 273 |
| SubProcessInstance() | 275 |
| ActivityInstanceNotification actions | 277 |
| PersistentObject() | 278 |
| StartTool() | 280 |
| BlockInstanceMonitor actions | 282 |
| ObtainBlockInstanceMonitor() | 282 |
| ObtainProcessInstanceMonitor() | 284 |
| Refresh() | 287 |
| Container activity implementation calls. | 289 |
| InContainer() | 289 |
| OutContainer() | 291 |
| RemoteInContainer() | 293 |
| RemoteOutContainer() | 294 |
| SetOutContainer() | 296 |
| SetRemoteOutContainer() | 298 |
| ExecutionService actions. | 300 |
| CreateProcessInstanceList() | 301 |
| CreateProcessTemplateList() | 306 |
| CreateWorklist() | 312 |
| Logoff() | 319 |
| Logon() | 320 |
| Passthrough() | 325 |
| QueryActivityInstanceNotifications() | 327 |
| QueryItems() | 333 |
| QueryProcessInstanceLists() | 339 |
| QueryProcessInstanceNotifications() | 341 |
| QueryProcessInstances() | 347 |
| QueryProcessTemplateLists() | 352 |
| QueryProcessTemplates() | 354 |
| QueryWorkitems() | 358 |
| QueryWorklists() | 365 |
| Receive() | 367 |
| RemotePassthrough() | 370 |
| TerminateReceive() | 372 |
| Item actions | 373 |
| Delete() | 374 |
| ObtainProcessInstanceMonitor() | 376 |
| ProcessInstance() | 378 |
| Refresh() | 380 |
| SetDescription() | 382 |
| SetName() | 384 |
| Transfer() | 386 |
| PersistentList actions | 388 |
| Delete() | 389 |
| Refresh() | 390 |
| SetDescription() | 392 |
| SetFilter() | 394 |

| | |
|-------------------------------------|-----|
| SetSortCriteria() | 396 |
| SetThreshold() | 398 |
| Person actions | 400 |
| Refresh() | 400 |
| SetAbsence() | 402 |
| SetSubstitute() | 403 |
| ProcessInstance actions | 405 |
| Delete() | 406 |
| InContainer() | 407 |
| ObtainMonitor() | 410 |
| PersistentObject() | 412 |
| Refresh() | 414 |
| Restart() | 416 |
| Resume() | 417 |
| SetDescription() | 419 |
| SetName() | 421 |
| Start() | 423 |
| Suspend() | 425 |
| Terminate() | 427 |
| ProcessInstanceList actions | 429 |
| QueryProcessInstances() | 429 |
| ProcessInstanceNotification actions | 432 |
| PersistentObject() | 432 |
| ProcessTemplate actions | 434 |
| CreateAndStartInstance() | 434 |
| CreateInstance() | 439 |
| Delete() | 442 |
| ExecuteProcessInstance() | 444 |
| InitialInContainer() | 451 |
| PersistentObject() | 453 |
| ProgramTemplate() | 455 |
| Refresh() | 457 |
| ProcessTemplateList actions | 459 |
| QueryProcessTemplates() | 459 |
| ProgramTemplate actions | 461 |
| Execute() | 462 |
| Service actions | 466 |
| Refresh() | 466 |
| SetPassword() | 467 |
| UserSettings() | 469 |
| Workitem actions | 471 |
| CancelCheckOut() | 473 |
| CheckIn() | 475 |
| CheckOut() | 477 |
| Finish() | 481 |
| ForceFinish() | 483 |
| ForceRestart() | 485 |
| InContainer() | 486 |
| OutContainer() | 488 |
| PersistentObject() | 490 |
| Restart() | 492 |

| | |
|--------------------------------------|-----|
| Start() | 494 |
| StartTool() | 495 |
| Terminate() | 497 |
| Worklist actions | 499 |
| QueryActivityInstanceNotifications() | 500 |
| QueryItems() | 502 |
| QueryProcessInstanceNotifications() | 505 |
| QueryWorkitems() | 507 |

Chapter 6. Examples 511

| | |
|--|-----|
| How to create persistent lists | 511 |
| Create a process instance list (C) | 512 |
| Create a process instance list (C++) | 513 |
| Create a process instance list (Java) | 514 |
| Create a process instance list (COBOL) | 517 |
| How to query persistent lists | 521 |
| Query worklists (C) | 522 |
| Query worklists (C++) | 524 |
| Query worklists (Java) | 525 |
| Query worklists (COBOL) | 528 |
| How to query a set of objects | 535 |
| Query process instances (C) | 536 |
| Query process instances (C++) | 538 |
| Query process instances (Java) | 539 |
| Query process instances (COBOL) | 542 |
| Query work items from a worklist (C) | 546 |
| Query work items from a worklist (C++) | 548 |
| Query work items from a worklist (Java) | 550 |
| Query work items from a worklist (COBOL) | 553 |
| How to code an activity implementation | 558 |
| Programming an activity implementation (C) | 559 |
| Programming an activity implementation (C++) | 560 |
| Programming an activity implementation (COBOL) | 561 |

Glossary 565

Bibliography 571

| | |
|---|-----|
| MQSeries Workflow for OS/390 publications | 571 |
| MQSeries Workflow publications | 571 |
| Workflow publications | 571 |
| MQSeries publications | 571 |
| Other useful publications | 571 |
| Licensed books | 572 |

Index 573

Readers' Comments — We'd Like to Hear from You 581

Figures

| | | | |
|---|-----|--|-----|
| 1. MQSeries Workflow Client API hierarchy | 2 | 40. Backward mapping with constants. | 172 |
| 2. Setting up client/server communication | 3 | 41. Relationship between mapping elements. | 177 |
| 3. Querying objects | 4 | 42. Usertype exit | 189 |
| 4. Dealing with process instances and (work) items | 5 | 43. Process instance states | 405 |
| 5. Monitoring a process instance. | 6 | 44. Work item states - process instance state running | 472 |
| 6. Handling data sent by an MQSeries Workflow server | 7 | 45. Work item states - process instance state suspending or suspended | 472 |
| 7. Accessing a result object in C | 12 | 46. Work item states - process instance state terminating or terminated | 473 |
| 8. Accessing a result object in C++. | 13 | 47. Sample C program to create a process instance list | 512 |
| 9. Accessing a result object in COBOL (via PERFORM) | 14 | 48. Sample C++ program to create a process instance list | 513 |
| 10. Accessing a result object in COBOL (via CALL) | 15 | 49. Sample Java program to create a process instance list | 514 |
| 11. Handling data sent by an MQSeries Workflow server | 18 | 50. Sample COBOL program to create a process instance list (via PERFORM) | 517 |
| 12. Reading a vector in C (using First/NextElement() calls). | 25 | 51. Sample COBOL program to create a process instance list (via CALL) | 519 |
| 13. Reading a vector in C (using NextElement() call only) | 26 | 52. Sample C program to query worklists | 522 |
| 14. Reading a vector in COBOL (using First/NextElement calls) | 27 | 53. Sample C++ program to query worklists | 524 |
| 15. Reading a vector in COBOL (using NextElement calls only) | 29 | 54. Sample Java program to query worklists | 525 |
| 16. Process instance monitors and block instance monitors | 66 | 55. Sample COBOL program to query worklists (via PERFORM). | 528 |
| 17. C example using basic functions | 78 | 56. Sample COBOL program to query worklists (via CALL) | 531 |
| 18. C++ example using basic methods | 79 | 57. Sample C program to query process instances | 536 |
| 19. COBOL example using basic calls (via PERFORM) | 80 | 58. Sample C++ program to query process instances | 538 |
| 20. COBOL example using basic calls (via CALL) | 82 | 59. Sample Java program to query process instances | 539 |
| 21. Accessing values in C. | 115 | 60. Sample COBOL program to query process instances (via PERFORM) | 542 |
| 22. Accessing values in C++. | 116 | 61. Sample COBOL program to query process instances (via CALL) | 544 |
| 23. Accessing values in COBOL (via PERFORM) | 117 | 62. Sample C program to query work items from a worklist. | 546 |
| 24. Accessing values in COBOL (via CALL) | 119 | 63. Sample C++ program to query work items from a worklist | 548 |
| 25. MQSeries Workflow message | 149 | 64. Sample Java program to query work items from a worklist | 550 |
| 26. Sending requests to MQSeries Workflow | 152 | 65. Sample COBOL program to query work items from a worklist (via PERFORM) | 553 |
| 27. Starting an activity implementation via XML | 154 | 66. Sample COBOL program to query work items from a worklist (via CALL) | 555 |
| 28. Sample activity implementation using XML | 156 | 67. Sample activity implementation (C) | 559 |
| 29. Document type definition (DTD) for MQSeries Workflow XML messages | 158 | 68. Sample activity implementation (C++) | 560 |
| 30. Program mapping illustration. | 164 | 69. Sample activity implementation (COBOL, via PERFORM) | 561 |
| 31. Program mapping control flow | 164 | 70. Sample activity implementation (COBOL, via CALL) | 563 |
| 32. How to create a program mapping. | 165 | | |
| 33. Default forward/backward mapping | 167 | | |
| 34. Usertype example | 168 | | |
| 35. Default forward mapping illustration. | 169 | | |
| 36. Forward2: Non-default forward mapping illustration. | 169 | | |
| 37. Non-default backward mapping Backward1 illustration. | 170 | | |
| 38. Backward2: Explicit mapping illustration. | 170 | | |
| 39. Forward mapping with constants.. | 172 | | |

Tables

| | | | |
|---|-----|--|-----|
| 1. List of return codes | 9 | 11. Mapping with constant definition. | 171 |
| 2. Authorization for persons. | 68 | 12. Mapping combinations | 174 |
| 3. JCLs provided for C/C++ programs | 126 | 13. C/C++ data type mappings (legacy application (C/C++) to FDL types (structure)) | 175 |
| 4. Copybooks provided for COBOL programs | 136 | 14. COBOL data type mappings (legacy application (COBOL) to FDL types (structure)) | 176 |
| 5. JCLs provided for COBOL programs | 137 | 15. Interface element size. | 191 |
| 6. Mapping C to COBOL data types | 137 | 16. Context types | 212 |
| 7. Function name mapping | 138 | | |
| 8. Class prefix abbreviations | 144 | | |
| 9. Abbreviations for COBOL naming | 145 | | |
| 10. Rule mapping with no constant definition | 171 | | |

About this book

This book describes how to use the IBM MQSeries Workflow for OS/390 Runtime (Client) Application Programming Interface (MQSeries Workflow API) and also to invoke API requests by passing messages in Extensible Markup Language (XML) to an MQSeries queue from an application. The first part of the book describes the concepts underlying the API while the rest of the book provides a reference for the API action calls. The book also describes the MQSeries Workflow predefined data structures.

Note: The licensed books that were declassified in OS/390 Version 2 Release 4 appear on the OS/390 Online Library Collection, SK2T-6700. The remaining licensed books for OS/390 Version 2 appear on the OS/390 Licensed Product library, LK2T-2499, in unencrypted form.

Who should read this book

This book is intended for programmers who design and implement programs using an MQSeries Workflow API and who may participate in designing an MQSeries Workflow workflow model. It assumes that readers are experienced OS/390 programmers and that they understand the process modeling concepts.

How to get additional information

Visit the MQSeries Workflow home page at
<http://www.software.ibm.com/ts/mqseries/workflow>

For a list of additional publications, refer to “MQSeries Workflow publications” on page 571.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any other MQSeries Workflow documentation, choose one of the following methods:

- Send your comments by e-mail to: s390id@de.ibm.com
Be sure to include the name of the book, the part number of the book, the version of MQSeries Workflow, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).
- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

How this book is organized

“Notices” on page xiii describes some notices and trademarks.

“Chapter 1. MQSeries Workflow programming concepts” on page 1 provides an overview of how to design applications to work with the MQSeries Workflow workflow manager.

“Chapter 2. Language interfaces” on page 123 discusses the API from the perspective of the language used: C, C++, Java, or COBOL.

“Chapter 3. Interfacing with the Program Execution Server” on page 163 describes the interface with the Program Execution Server, including the use of program mappings to bring Workflow API containers into a format acceptable by legacy applications and how to use exits.

“Chapter 4. API classes and objects” on page 215 provides an overview of the classes supported by the API.

“Chapter 5. API action and activity implementation calls” on page 273 describes the API calls that enable applications to manipulate worklists and work items, to work with process instances and container data, and to log on to and log off from an MQSeries Workflow server.

“Chapter 6. Examples” on page 511 provides some examples that show how to use the API.

The back of the book includes a glossary that defines terms as they are used in this book, a bibliography, and an index.

How to read the syntax diagrams

Throughout this book, syntax is described the following way; all spaces and other characters are significant:

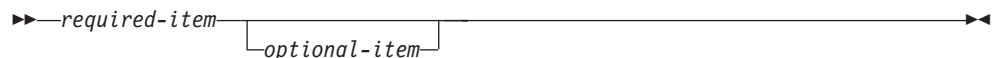
- Read the syntax diagrams from left to right, from top to bottom, following the main path of the line.
The **▶▶**— symbol indicates the beginning of a statement.
The **—▶** symbol indicates that the statement syntax is continued on the next line.
The **▶—** symbol indicates that a statement is continued from the previous line.
The **—▶▶** symbol indicates the end of a statement.
- Diagrams can be broken into fragments. A fragment is indicated by vertical bars with the name of the fragment between the bars. The fragment itself follows the same syntactical rules as the main diagram.



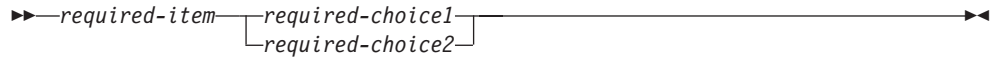
- Required items appear on the horizontal line, the main path.



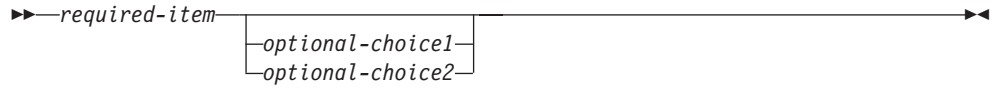
- Optional items appear below (or above) the main path.



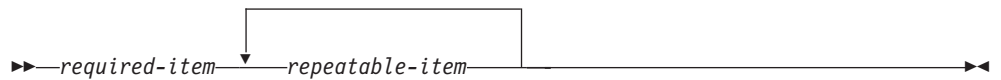
- If you can choose from one or more items, they appear vertically, in a stack.
If you must choose one of the items, one item of the stack appears on the main path.



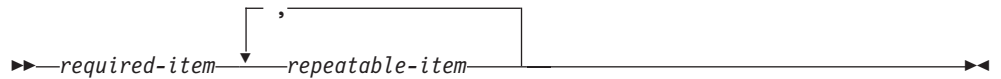
If choosing one of the items is optional, the entire stack appears below the main path.



- An arrow returning to the left, above the main path, indicates an item that can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.



- Keywords appear in uppercase, for example, NAME. They must be spelled exactly as shown. Variables appear in lowercase italic letters, for example, *string*. They represent user-supplied values.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
 IBM Corporation
 North Castle Drive
 Armonk, NY 10504-1785
 U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples can include the names of individuals, companies, brands, or products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information in an online form, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

- AIX
- CICS
- DB2
- FlowMark
- IBM
- IMS
- Language Environment
- MQSeries
- MVS
- OS/2
- OS/390
- RISC System/6000

Lotus Notes is a registered trademark, and Domino and Lotus Go Webserver are trademarks of Lotus Development Corporation.

Microsoft, Windows, Windows NT and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and/or other countries.

Other company, product, and service names may be trademarks or service marks of others.

Summary of changes

Changes to this document for IBM MQSeries Workflow for OS/390 Version 3 Release 2.1 are:

- Support for C++, Java, and XML has been added.
- The book was restructured to better accommodate the description of the C++ and Java interfaces.
- Lists of figures and tables have been added.
- The discussion of the audit trail (formerly an appendix in this manual) can now be found in *IBM MQSeries Workflow: Administration Guide*.
- The specification of a configuration identifier is supported.
- The ExecutionService class exposes a new action API call Refresh().
- The ProcessTemplate class exposes a new action API call ExecuteProcessInstance() in synchronous and asynchronous modes. Accordingly, an asynchronous communication protocol has been added. This class also exposes a new API call ProgramTemplate().
- A new class and new functions have been added to describe a program template. A program template supports an API call Execute() in synchronous and asynchronous modes.
- The ProcessInstance class exposes a new action Restart().
- The WorkItem and ActivityInstanceNotification classes expose the new action StartTool().
- The WorkItem class exposes the new action CancelCheckOut().
- Restrictions have been removed from the WorkItem Restart(), ForceFinish(), and ForceRestart() API calls; a work item implemented by a *process* can now also be restarted or finished.
- XML message interface support has been added to create and start or to execute a process instance. Additionally, an activity implementation can be started by the MQSeries Workflow execution server via a user-defined program execution server.
- The ProcessTemplate InContainer() API call has been renamed to InitialInContainer(). The usage of InContainer(), which is retained for compatibility, is discouraged.
- The ProgramData class exposes new accessor API calls ExecutionMode(), ExecutionUser(), and ProgramTrusted().
- The ExecutionService class provides for new allocation means which allow specifying the system group only. This is to exploit the IBM MQSeries clustering capabilities.
- The ExecutionService API calls to query process instances or to create a process instance list allow for the specification of a new filter attribute, START_TIME.

Chapter 1. MQSeries Workflow programming concepts

This chapter provides you with a general introduction to the programming concepts of MQSeries Workflow.

Understanding Workflow programming

This section introduces the concept of workflow modeling as it relates to the design of application programs for use with IBM MQSeries Workflow.

MQSeries Workflow provides a way to model a process and assign applications to activities in the resulting workflow model. This enables the workflow manager to automate the control of activities and the flow of data.

Work can be routed to the person who performs the activity instance. An application program required to perform an activity instance can be designed to start when a user starts an activity instance.

The role of the programmer in modeling a process

As workflow models are defined, the applications and data structures needed to support program activities are identified. Programmers can create new applications, integrate existing applications, or reengineer existing applications to support these program activities.

To reengineer existing applications with the workflow model, programmers must determine if the applications used by the enterprise can be functionally decomposed. The control and flow logic are separated from the application, the start and exit conditions are moved into the workflow model, and the program is divided into modules to be invoked by the workflow manager at the appropriate points. The resulting modules are applications that are assigned to perform the program activities defined in the workflow model.

Most applications include many diverse functions, and many can support several different activities in different stages of a process. Output produced by one function of a program can be used as input by another function of the same program. Therefore, the same application can be used to support many different program activities in a workflow model.

Your enterprise might also use Enterprise Resource Planning (ERP) or packaged applications like word-processing or spreadsheet applications.

Decomposition of such applications may not be possible. However, a programmer could write shell procedures that query the contents of containers, pass data from an input container to the program when the activity instance is started, and direct data into an output container when it finishes.

With MQSeries Workflow you will be able to use mappings so you can support any legacy application with this tool. There may be old applications whose interfaces you can't change because other applications or programs have been configured to work with these long time ago: if you changed one configuration of an interface, you would have to change them all. This mapper enables you to use all legacy applications with your Workflow applications via the mapping tool.

Programming concepts

Return codes, provided by the assigned program, can then be used to evaluate exit and transition conditions.

Programming interfaces

MQSeries Workflow provides application program interface (API) and Extensible Markup Language (XML) message interface support, as well as a set of predefined data structure members, to assist programmers who develop applications for use with workflow models. In addition, several programming samples are provided.

In a programming-language-based programming model, the client application makes an API call in order to execute a request. In a message-based programming model, the request and information needed to execute the request are contained in a message that is interchanged through a message queuing system between the client application and a server.

The MQSeries Workflow predefined data structure members provide information about the current process, activity, or block, and are associated with the operating characteristics of a process instance or activity instance.

API interfaces in the following languages are described in this book:

- C
- C++
- COBOL
- Java
- MQSeries Workflow XML message interface

| | | | | | | |
|----------------------|------------------------|-------------|----------------|----------------------------|----------------|--------------|
| X M L | ActiveX | Java | Lotus Notes | <i>Visual Basic V2</i> | <i>REXX V2</i> | COBOL |
| | C++ (V3/V2) | | | C (V2) | | |
| | C (V3) | | | | | |

= supported under OS/390

Figure 1. MQSeries Workflow Client API hierarchy

The basic interfaces for requesting Runtime services from MQSeries Workflow are a C API and an XML message interface. Access can be gained to the C functions from all languages that support C calls - see "Compiling and linking" on page 125 for more information. A C++ language API is provided on top of the C API. The C++ API is a small layer of inline methods, that is, delivered as source code. The Java API is implemented on top of the C++ layer, and the COBOL API on top of the C layer.

MQSeries Workflow uses the XML 1.0 standard (see *W3C Recommendation: Extensible Markup Language (XML) 1.0*) as the document description language.

The MQSeries Workflow API provides API calls:

- To execute process models, that is, to work with process instances and container data and to manipulate worklists and work items
- To monitor the progress of execution
- To issue process administrator functions

- To receive information sent by an MQSeries Workflow server
- To process container data associated with an activity implementation

Prerequisites for using a programming language API

MQSeries Workflow application development assumes that the appropriate environment is established. This means that:

- MQSeries Workflow for OS/390 must be installed on the machine where you are developing your applications.
- A compiler of one of the supported languages must be installed and configured.
- Buildtime must be installed on the machine where you are developing your applications.

Refer to “Chapter 2. Language interfaces” on page 123 for more information.

Overview of the Runtime API

There are various tasks which you typically want to address by writing an MQSeries Workflow application program:

- You can write a client application to:
 - Manage process instances
 - Handle worklists and/or work items
 - Administer process instances or work items
 - Monitor the progress of execution
- You can write a program that implements an activity in your workflow process.

These programs typically use only a subset of the MQSeries Workflow API. For example, an activity implementation typically only accesses its containers, that is, uses only the so-called Container API, which is a subset of the full API especially configured for container-only processing. The MQSeries Workflow API, that is, its header files and library structures or its import packages take this fact into account. IMS programs must use the Container API.

In order to ask for Runtime services, communication must be established between the client application and an MQSeries Workflow execution server.

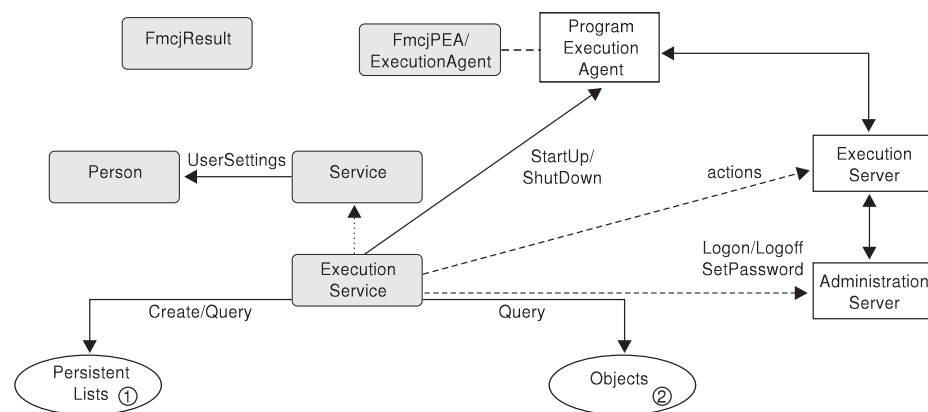


Figure 2. Setting up client/server communication. Legend: --> Inheritance (C++); —> provides for access; - -> sends messages to

As a first step, an ExecutionService object must be obtained (constructed/allocated/located). An ExecutionService object represents a session

Programming concepts

between a user and an MQSeries Workflow execution server. It essentially provides the basic API calls to set up a communication path to the specified MQSeries Workflow execution server and to establish the user session (Logon() or Passthrough()), and finish it (Logoff()). To log on, not only the execution server but also the administration server must be up and running so that authentication can be done. This is, however, transparent to you.

When the session to an execution server has been established, you can:

- Query objects for which you are authorized: process templates, process instances, items (work items, activity instance notifications, process instance notifications), or lists containing such objects.
- Create persistent lists, that is, persistent views on objects contained in the MQSeries Workflow database.
- Query information about the logged-on user or change that user's password.

In C, C++, and COBOL, all API calls update a so-called result object. Detailed information about an erroneous request can be obtained from there. See "Handling errors" on page 8 for more information.

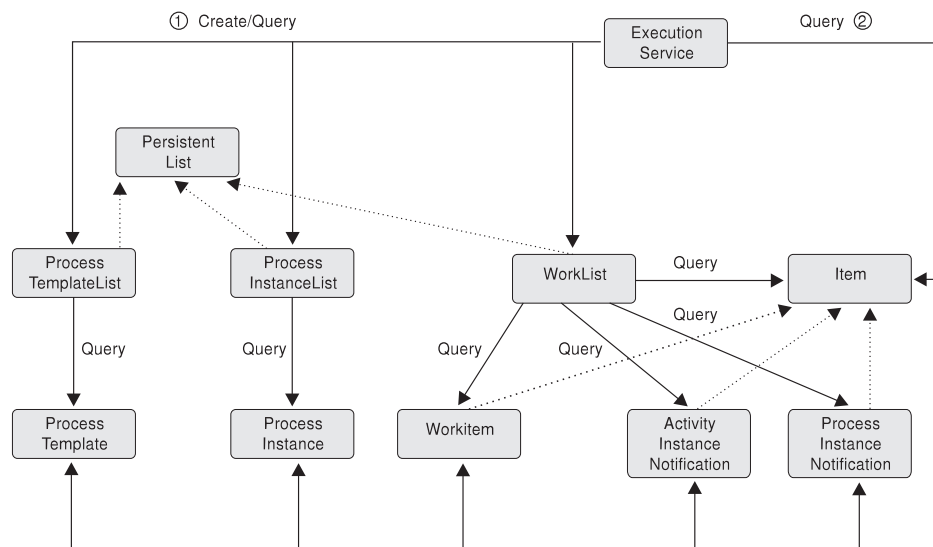


Figure 3. Querying objects. Legend: --> Inheritance (C++); —> provides for access

When the session to an execution server has been established, you can create or query persistent lists (process template lists, process instance lists, worklists) or query other objects for which you are authorized. At runtime, you can retrieve the currently valid version of a process template only; you cannot see any future or past versions.

A persistent list represents a set of objects the user is authorized for. It is a view of those objects. All objects which are accessible through the list have the same characteristics. These characteristics are specified by a filter. For example, depending on the filter specified, a worklist can contain a set of work items only. No activity instance notifications or process instance notifications are accessible through that list. The worklist content, the work items, can be queried and their attributes can be accessed. As soon as a work item has been read from the execution server, further actions can be called, for example, starting a work item.

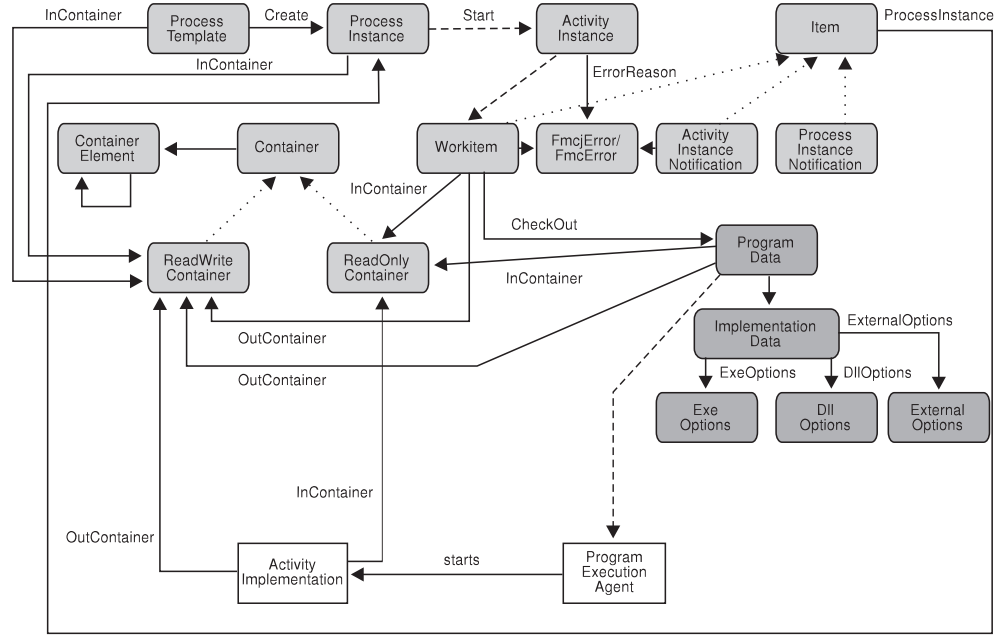


Figure 4. Dealing with process instances and (work) items. Legend: --> Inheritance (C++); -> provides for access; —> data is passed to or results in

When a valid version of a process template has been retrieved, a process instance can be created and started. Starting a process instance can require input data. You can use the Container API calls for reading and writing values. See “Handling containers” on page 30 for more information.

Starting a process instance triggers the scheduling of activity instances and, as a result of that, the creation of a set of work items and possibly activity instance notifications or process instance notifications when they are not worked on in time. A work item implemented by a program can then be executed either by MQSeries Workflow-specific means or by user-specific means.

When executed by user-specific means, the work item is to be checked out. Checking out provides for all information needed to execute the underlying program, the program data and its description of the implementing options and the input container data.

When executed by MQSeries Workflow-specific means, that program data is automatically sent to the program execution server which starts the appropriate activity implementation. The activity implementation can then access its input and output containers via an appropriate request to the program execution server. The same container accessor API calls are applicable whether called from a client application program or from an activity implementation program.

When a work item and thus the associated activity instance has not been executed successfully, the FmcjError or FmcError object provides for analyzing the cause of the state InError.

Programming concepts

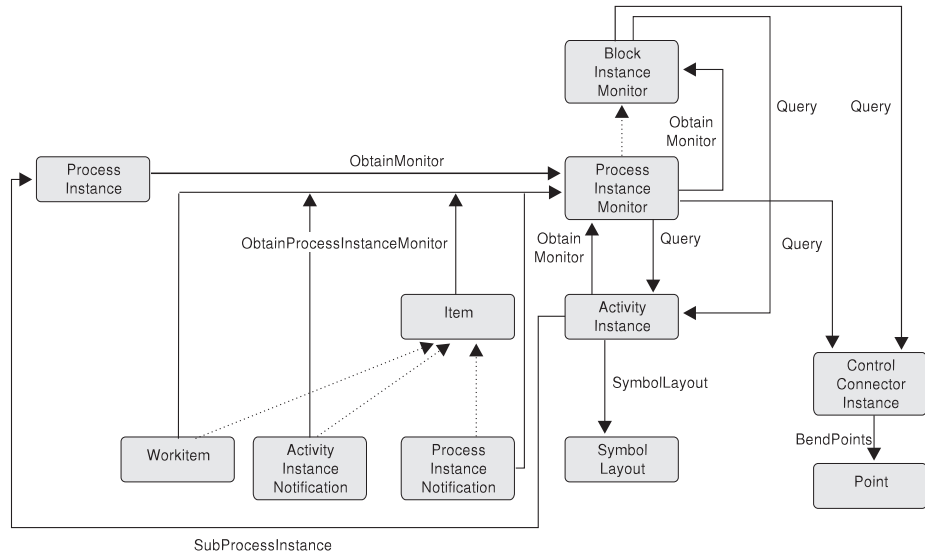


Figure 5. Monitoring a process instance. Legend: --> Inheritance (C++); —> provides for access

When a process instance or item, that is, a work item, an activity instance notification, or a process instance notification, has been retrieved, you can obtain the associated process instance monitor. The process instance monitor then allows for analyzing the states of activity instances and control connector instances. The path taken through the process instance can thus be determined. In case you want to present this information graphically, the activity instance symbol layout and the control connector instance positions and bend points offer support.

Once a process instance monitor has been obtained, you can iterate into the process model by obtaining block instance monitors for activities of type Block or process instance monitors for activities of type Process, that is, for subprocess instances. See “Monitoring a process instance” on page 65 for more information.

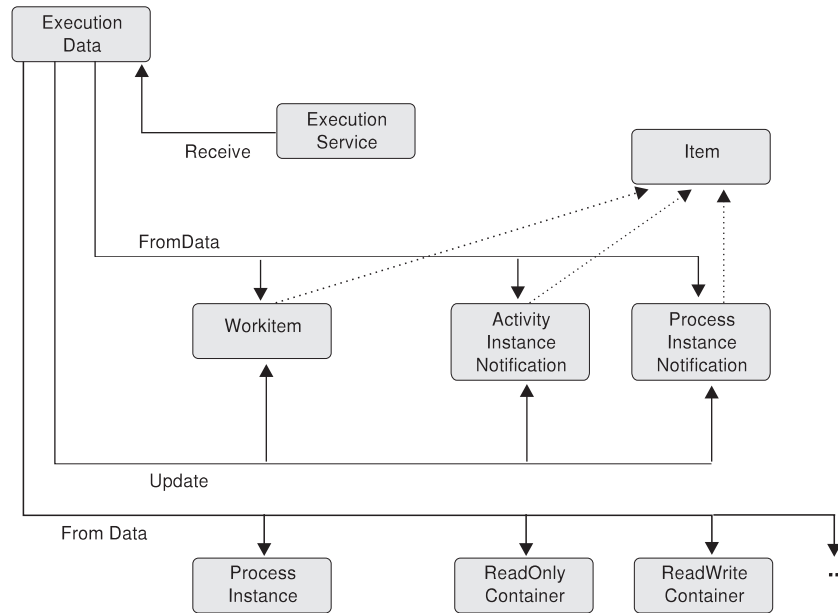


Figure 6. Handling data sent by an MQSeries Workflow server. Legend: --> Inheritance (C++); -> provides for access

When the process setting specifies a *push refresh policy*, then the MQSeries Workflow execution server pushes changes on work items or notifications to a present client. In this case, or when the application issues an asynchronous request, the client application should set up a means in order to receive data or responses sent by the server. Once received, the appropriate object can be updated, created, or deleted depending on the information sent. See “Client/server communication and data access models” on page 16 for more information.

Building an MQSeries Workflow application

Overview

There are essentially two different tasks which you can address by using the MQSeries Workflow API:

- You can write your own client application . For example, you may want to:
 - control the MQSeries Workflow functionality provided to your user.
 - present the MQSeries Workflow functionality in a way that your user is accustomed to.
 - run selected MQSeries Workflow tasks in batch mode.
- You can write a program that implements an activity in your workflow process model.

These two kinds of programs usually contain specific parts which are described in the sections “Coding an MQSeries Workflow client application” and “Coding an MQSeries Workflow activity implementation”. in the discussions of the respective language interfaces in “Chapter 2. Language interfaces” on page 123.

The concepts underlying the MQSeries Workflow API are common to all programs using the MQSeries Workflow API. They are summarized here and discussed in more detail in the following chapters.

Programming concepts

Concepts of the programming language API

All persistent objects such as work items and process instances are accessed through transient objects which represent their state at the time they were queried from a server. In C and COBOL, a so-called *handle* represents a pointer to such a transient object.

In order to request an action on an object, a session must have been established with an appropriate MQSeries Workflow server. The action itself can then be executed synchronously. Some actions can also be executed asynchronously.

Only objects for which you are authorized are returned from the server to the client.

Separate API calls (termed functions, methods, or subprograms, depending on the language) in the C, C++, COBOL, or Java languages are available for each action on an object or for accessing each property of an object. This approach allows API call parameters to be checked by the compiler and best represents the object-action paradigm supported by MQSeries Workflow.

In C, C++, and COBOL, detailed error information is provided by a so-called *result object*. This object is available in addition to the return code set by action API calls. See "The result object" on page 12 for detailed information on the result object.

Objects are managed by the application programmer but object memory is owned by the MQSeries Workflow API. The application programmer determines the lifetime of transient objects by using allocate, or query, and deallocate mechanisms. The MQSeries Workflow API hides the internal structure of transient objects.

Concepts of the XML message interface

All persistent objects are accessed by their unique name, that is, the actual name may need to be padded with the printable version of the object's identifier in order to achieve uniqueness.

In order to request an action, a session need not be established as in the programming language API. You must, however, be authorized for the action itself.

All actions are executed asynchronously. Correlation data is part of the message so that the application can correlate the request sent to MQSeries Workflow and the execution server response.

Handling errors

All action, activity implementation, or program execution management API calls messages show whether or not the call has been successfully executed by passing back a *return code*. Java throws an appropriate *FmcException* when the method has not been executed successfully. The XML message interface provides the return code in the response message. The return code is one of a set of predefined codes (see "List of return codes" on page 9). The exact return codes or exceptions for each of those API calls are listed with the description of each call. You should design your programs to handle all return codes or exceptions that can arise.

In addition to the return code, a so-called *result object* can be accessed in C, C++, and COBOL, which describes the result of the call in more detail - see "The result object" on page 12.

Basic and accessor API calls either do not return a value or return the value queried. Since they are querying transient objects and are able to return default

values, an error does normally not occur. It can, however, happen during application development that a wrong handle or a buffer too small to hold a character value is specified. To look for such erroneous situations, the *result object* can be queried (besides checking the trace).

List of return codes

The following list shows the numeric values of the return codes that are issued by the MQSeries Workflow API; it is strongly advised to use the symbolic names instead of the integer values. For COBOL, the return code identifiers have a maximum length of 30 characters. Additional words in the return codes are separated by hyphens and not by underscores (as is common for C). In order to avoid misunderstandings, the C version of the return codes is used in this book, especially in descriptions of the API calls (“Chapter 5. API action and activity implementation calls” on page 273).

Table 1. List of return codes

| Numeric value | Symbolic value (C/C++) | Symbolic value (COBOL) |
|---------------|--------------------------------|--------------------------------|
| 0 | FMC_OK | FMC-OK |
| 1 | FMC_ERROR | FMC-ERROR |
| 10 | FMC_ERROR_USERID_UNKNOWN | FMC-ERROR-USERID-UNKNOWN |
| 11 | FMC_ERROR_ALREADY_LOGGED_ON | FMC-ERROR-ALR-LOGGED-ON |
| 12 | FMC_ERROR_PASSWORD | FMC-ERROR-PASSWORD |
| 13 | FMC_ERROR_COMMUNICATION | FMC-ERROR-COMMUNICATION |
| 14 | FMC_ERROR_TIMEOUT | FMC-ERROR-TIMEOUT |
| 100 | FMC_ERROR_INTERNAL | FMC-ERROR-INTERNAL |
| 101 | FMC_ERROR_SERVER | FMC-ERROR-SERVER |
| 102 | FMC_ERROR_UNKNOWN | FMC-ERROR-UNKNOWN |
| 103 | FMC_ERROR_MESSAGE_FORMAT | FMC-ERROR-MESSAGE-FORMAT |
| 104 | FMC_ERROR_MESSAGE_DATA | FMC-ERROR-MESSAGE-DATA |
| 105 | FMC_ERROR_RESOURCE | FMC-ERROR-RESOURCE |
| 106 | FMC_ERROR_NOT_LOGGED_ON | FMC-ERROR-NOT-LOGGED-ON |
| 107 | FMC_ERROR_NEW_OWNER_NOT_FOUND | FMC-ERROR-NEW-OWNER-NOT-FOUND |
| 108 | FMC_ERROR_NO_OLD_OWNER | FMC-ERROR-NO-OLD-OWNER |
| 109 | FMC_ERROR_OLD_OWNER_ABSENT | FMC-ERROR-OLD-OWNER-ABSENT |
| 110 | FMC_ERROR_NEW_OWNER_ABSENT | FMC-ERROR-NEW-OWNER-ABSENT |
| 111 | FMC_ERROR_ALREADY_STARTED | FMC-ERROR-ALR-STRTD |
| 112 | FMC_ERROR_MEMBER_NOT_FOUND | FMC-ERROR-MEMBER-NOT-FOUND |
| 113 | FMC_ERROR_MEMBER_NOT_SET | FMC-ERROR-MEMBER-NOT-SET |
| 114 | FMC_ERROR_WRONG_TYPE | FMC-ERROR-WRONG-TYPE |
| 115 | FMC_ERROR_MEMBER_CANNOT_BE_SET | FMC-ERROR-MEMBER-CANNOT-BE-SET |
| 116 | FMC_ERROR_MEMBER_INVALID | FMC-ERROR-MEMBER-INVAL |
| 117 | FMC_ERROR_FORMAT | FMC-ERROR-FORMAT |
| 118 | FMC_ERROR_DOES_NOT_EXIST | FMC-ERROR-DOES-NOT-EXIST |
| 119 | FMC_ERROR_NOT_AUTHORIZED | FMC-ERROR-NOT-AUTH |
| 120 | FMC_ERROR_WRONG_STATE | FMC-ERROR-WRONG-STATE |
| 121 | FMC_ERROR_NOT_UNIQUE | FMC-ERROR-NOT-UNIQUE |
| 122 | FMC_ERROR_EMPTY | FMC-ERROR-EMPTY |
| 123 | FMC_ERROR_NO_MANUAL_EXIT | FMC-ERROR-NO-MANUAL-EXIT |
| 124 | FMC_ERROR_PROFILE | FMC-ERROR-PROFILE |
| 125 | FMC_ERROR_INVALID_FILTER | FMC-ERROR-INVAL-FILTER |
| 126 | FMC_ERROR_PROGRAM_EXECUTION | FMC-ERROR-PROGRAM-EXECUTION |
| 127 | FMC_ERROR_PROTOCOL | FMC-ERROR-PROTOCOL |
| 128 | FMC_ERROR_TOOL_FUNCTION | FMC-ERROR-TOOL-FUNCTION |
| 129 | FMC_ERROR_INVALID_TOOL | FMC-ERROR-INVAL-TOOL |
| 130 | FMC_ERROR_INVALID_HANDLE | FMC-ERROR-INVAL-HANDLE |

Programming concepts

Table 1. List of return codes (continued)

| Numeric value | Symbolic value (C/C++) | Symbolic value (COBOL) |
|---------------|--|----------------------------------|
| 131 | FMC_ERROR_NOT_EMPTY | FMC-ERROR-NOT-EMPTY |
| 132 | FMC_ERROR_INVALID_USER | FMC-ERROR-INVAL-USER |
| 133 | FMC_ERROR_OWNER_ALREADY_ASSIGNED | FMC-ERROR-OWNER-ALR-ASSIGNED |
| 134 | FMC_ERROR_INVALID_NAME | FMC-ERROR-INVAL-NAME |
| 135 | FMC_ERROR_INVALID_PROGRAMID | FMC-ERROR-INVAL-PROGRAMID |
| 136 | FMC_ERROR_SIZE_EXCEEDED | FMC-ERROR-SIZE-EXCEEDED |
| 406 | FMC_ERROR_WRONG_ACT_IMPL_KIND | FMC-ERROR-WRONG-ACT-IMPL-KIND |
| 500 | FMC_ERROR_NON_LOCAL_USER | FMC-ERROR-NON-LOCAL-USER |
| 501 | FMC_ERROR_WRONG_KIND | FMC-ERROR-WRONG-KIND |
| 502 | FMC_ERROR_INVALID_ACTIVITY | FMC-ERROR-INVAL-ACT |
| 503 | FMC_ERROR_CHECKOUT_NOT_POSSIBLE | FMC-ERROR-CHKOUT-NOT-POSSIBLE |
| 504 | FMC_BACK_LEVEL_VERSION | FMC-BACK-LEVEL-VERSION |
| 505 | FMC_ERROR_NEWER_VERSION | FMC-ERROR-NEWER-VERSION |
| 506 | FMC_ERROR_INVALID_CORRELATION_ID | FMC-ERROR-INVAL-CORRELATION-ID |
| 507 | FMC_ERROR_NOT_ALLOWED | FMC-ERROR-NOT-ALLOWED |
| 508 | FMC_ERROR_BACK_LEVEL_OBJECT | FMC-ERROR-BACK-LEVEL-OBJ |
| 509 | FMC_ERROR_INVALID_CONTAINER | FMC-ERROR-INVAL-CNTR |
| 510 | FMC_ERROR_UNEXPECTED_CONTAINER | FMC-ERROR-UNEXPECTED-CNTR |
| 511 | FMC_ERROR_NO_PROGRAM_FOR_PLATFORM | FMC-ERROR-NO-PROG-FOR-PLATF |
| 800 | FMC_ERROR_BUFFER | FMC-ERROR-BUFFER |
| 801 | FMC_ERROR_INVALID_SESSION | FMC-ERROR-INVAL-SESSION |
| 802 | FMC_ERROR_INVALID_TIME | FMC-ERROR-INVAL-TIME |
| 804 | FMC_ERROR_NO_MORE_DATA | FMC-ERROR-NO-MORE-DATA |
| 805 | FMC_ERROR_INVALID_OID | FMC-ERROR-INVAL-OID |
| 807 | FMC_ERROR_INVALID_THRESHOLD | FMC-ERROR-INVAL-THRESHOLD |
| 808 | FMC_ERROR_INVALID_SORT | FMC-ERROR-INVAL-SORT |
| 809 | FMC_ERROR_OBJECT_IN_USE | FMC-ERROR-OBJ-IN-USE |
| 810 | FMC_ERROR_INVALID_DESCRIPTION | FMC-ERROR-INVAL-DESCRIPTION |
| 811 | FMC_ERROR_INVALID_INVOCATION_TYPE | FMC-ERROR-INVAL-INV-TYPE |
| 812 | FMC_ERROR_OWNER_NOT_FOUND | FMC-ERROR-OWNER-NOT-FOUND |
| 813 | FMC_ERROR_INVALID_LIST_TYPE | FMC-ERROR-INVAL-LIST-TYPE |
| 814 | FMC_ERROR_INVALID_RESULT_HANDLE | FMC-ERROR-INVAL-RESULT-HANDLE |
| 815 | FMC_ERROR_MESSAGE_CATALOG | FMC-ERROR-MESSAGE-CATALOG |
| 816 | FMC_ERROR_INVALID_SPECIFICATION | FMC-ERROR-INVAL-SPECIFICATION |
| 817 | FMC_ERROR_QRY_RESULT_TOO_LARGE | FMC-ERROR-QRY-RESULT-TOO-LARGE |
| 818 | FMC_ERROR_NO_VERSION_2_FILTER | FMC-ERROR-NO-VERSION-2-FILTER |
| 819 | FMC_ERROR_INVALID_USER_CONTEXT | FMC-ERROR-INVAL-USER-CONTEXT |
| 900 | FMC_ERROR_NO_SYS_ADMIN | FMC-ERROR-NO-SYS-ADMIN |
| 901 | FMC_ERROR_INVALID_SESSION_MODE | FMC-ERROR-INVAL-SESSION-MODE |
| 902 | FMC_ERROR_PROGRAM_UNDEFINED | FMC-ERROR-PROGRAM-UNDEFINED |
| 903 | FMC_ERROR_PEA_NOT_RUNNING | FMC-ERROR-PEA-NOT-RUNNING |
| 904 | FMC_ERROR_PEA_NOT_LOCAL | FMC-ERROR-PEA-NOT-LOCAL |
| 905 | FMC_ERROR_INVALID_ABSENCE_SPEC | FMC-ERROR-INVAL-ABSENCE-SPEC |
| 1000 | FMC_ERROR_NOT_SUPPORTED | FMC-ERROR-NOT-SUPPORTED |
| 1012 | FMC_ERROR_PROGRAM_NOT_DEFINED | FMC-ERROR-PROGRAM-NOT-DEFINED |
| 1014 | FMC_ERROR_PEA_NOT_REACHABLE | FMC-ERROR-PEA-NOT-REACHABLE |
| 1015 | FMC_ERROR_INVALID_PEA_FROM_CTNR | FMC-ERROR-INVALID-PEA-FRM-CTNR |
| 1016 | FMC_ERROR_INVALID_PEA_FROM_MODEL | FMC-ERROR-INVAL-PEA-FRM-MODEL |
| 1017 | FMC_ERROR_INVALID_SYSTEM_FROM_CTNR | FMC-ERROR-INVAL-SYSTEM-FRM-CTNR |
| 1018 | FMC_ERROR_INVALID_SYSTEM_FROM_MODEL | FMC-ERROR-INVAL-SYSTEM-FRM-MODEL |
| 1019 | FMC_ERROR_SUB_PROC_TERMINATED_BY_ERROR | FMC-ERROR-SB-PRC-TERM-BY-ERROR |
| 1020 | FMC_ERROR_NO_PEA_FOUND_FOR_AUTO_START | FMC-ERROR-NO-PEA-FND-FR-AUT-ST |

Table 1. List of return codes (continued)

| Numeric value | Symbolic value (C/C++) | Symbolic value (COBOL) |
|---------------|--------------------------------------|--------------------------------|
| 1021 | FMC_ERROR_NO_CTNR_ACCESS | FMC-ERROR-NO-CTNR-ACCESS |
| 1022 | FMC_ERROR_INVALID_CONFIG_ID | FMC-ERROR-INVAL-CONFIG_ID |
| 1023 | FMC_ERROR_MIG_OF_RUNNING_PROG | FMC-ERROR-MIG-OF-RUNNING-PROG |
| 1024 | FMC_ERROR_MIG_OF_CHKDOUT_SUSP | FMC-ERROR-MIG-OF-CHKDOUT-SUSP |
| 1025 | FMC_ERROR_MIGRATION_NO_SUBPROC | FMC-ERROR-MIGRATION-NO-SUBPROC |
| 1100 | FMC_ERROR_XML_DOCUMENT_INVALID | FMC-ERROR-XML-DOCUMENT-INVAL |
| 1101 | FMC_ERROR_XML_NO_MQSWF_DOCUMENT | FMC-ERROR-XML-NO-MQSWF-DOC |
| 1102 | FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED | FMC-ERROR-XML-MSG-NOT-SUPP |
| 1103 | FMC_ERROR_XML_WRONG_DATA_STRUCTURE | FMC-ERROR-XML-WRONG-DATA-STR |
| 1104 | FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND | FMC-ERROR-XML-D-M-NOT-FOUND |
| 1105 | FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE | FMC-ERROR-XML-D-M-WRONG-TYPE |
| 2000 | FMC_ERROR_INVALID_QUEUE_SCOPE | FMC-ERROR-INVAL-QUEUE-SCOPE |

Object and memory management

Workflow process models, their instances, and resulting work items are all objects persistently stored in an MQSeries Workflow database. This means that they exist independently from an application program.

When persistent objects are queried by an application program, they are represented by *transient objects* which carry the states of the persistent objects at the time of the query. When multiple queries are issued, there can be multiple transient objects representing the same persistent object, even representing different states of that object.

The lifetime of transient objects and their memory is *fully managed* by you, because you know best when those objects are no longer needed, that is, when objects are to be deallocated (C, COBOL) or destructed (C++). Transient objects are, however, no longer available when your application program ends.

Some transient objects are *explicitly allocated* by you. These support objects which do not reflect persistent ones. Examples are `FmcjStringVector` when you specify a set of persons to stand in for (C or COBOL) or `ExecutionService` object, which allows services to be requested from an execution server.

Transient objects which reflect persistent objects are *implicitly allocated* by you when you create or retrieve persistent objects, for example, by querying.

Although the lifetime of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQSeries Workflow API. The MQSeries Workflow API provides a handle (C, COBOL) to you so that you can issue requests against the object. In the C++ API, that handle is the only data member of your class. Therefore, you are independent of internal changes. It further allows MQSeries Workflow to "lazy read" (read only on demand) a collection of objects passed from the server and thus increases performance.

The MQSeries Workflow API follows the *programming by contract* concept. This means that any handle passed to it which is not 0 (NULL) is assumed to be a valid handle which can be used to access an object. This is especially important to be considered for queries. Any nonzero vector handle is assumed to point to an already existing vector of objects and is used in order to add newly qualifying objects. In other words, **you should initialize any new handle to 0.**

Programming concepts

As all resource memory is finally owned by the application process itself, you can access all objects from different threads within that process. MQSeries Workflow does not hinder you from using threads; it is coded reentrantly. On the other hand, MQSeries Workflow does not explicitly support threads. If you want to access the same transient object from within different threads, you have to synchronize the access to that object. Objects are **not** thread-safe.

The result object

In general, a result object states the result of the last MQSeries Workflow API request (in the affected thread). It especially allows for analyzing an erroneous situation in more detail and contains the following information:

- The return code.
- The origin of the result, that is, the file that caused the result to be written, and the line and function where the error or the completion of the request occurred.
- Parameters (up to five) which describe the objects involved.

The result can be retrieved as a formatted message text with all parameters added to the text. The current locale is considered when building that message text so that the message is provided in your selected language.

All results of API calls are written into the result object associated with the thread the request executes in. It is sufficient to access the result object just once per thread using the `FmcjResultObjectOfCurrentThread()` function or the `FmcjResult::ObjectOfCurrentThread()` method. (As threads are not supported in MQSeries Workflow for OS/390, "OfCurrentThread" is mentioned here for compatibility reasons with versions that do support threads.) The result object is automatically updated with each request.

A result object is automatically allocated by MQSeries Workflow when the first MQSeries Workflow API call is issued in that thread. It can be accessed at any time and as often as needed.

C example:

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
    FmcjResultHandle      result      = 0;
    FmcjStringVectorHandle parms      = 0;
    char                  buffer[2000]= "";

    result= FmcjResultObjectOfCurrentThread();
    printf( "Accessed result object of current thread\n" );

    printf( "Return code: %i\n", FmcjResultRc(result) );
    printf( "Text          : %s\n", FmcjResultMessageText(result,buffer,2000) );
    printf( "Origin       : %s\n", FmcjResultOrigin(result,buffer,2000) );
    parms= FmcjResultParameters(result);
    while ( 0 != FmcjStringVectorNextResultParmElement( parms, buffer, 2000 ) )
        printf( "Parameter  : %s\n", buffer );

    return 0;
}
```

Figure 7. Accessing a result object in C

Note: The NextResultParmElement() function is used on the string vector so that the result object is not changed while reading the parameters.

C++ example:

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
    vector<string> parms;
    FmcjResult *pResult = FmcjResult::ObjectOfCurrentThread();

    cout << "Accessed result object of current thread" << endl;
    cout << "Return code: " << pResult->Rc() << endl;
    cout << "Text      : " << pResult->MessageText() << endl;
    cout << "Origin    : " << pResult->Origin() << endl;
    pResult->Parameters(parms);
    cout << "Parameter  : ";
        for (int i=0; i<parms.size(); i++)
        {
            cout << parms[i] << " ";
        }
    cout << endl;

    delete pResult;                // cleanup object from heap
    return 0;
}
```

Figure 8. Accessing a result object in C++

Note: The transient C++ representation of your result object is destructed like any other object. Each retrieval of the result object constructs a separate representation.

COBOL examples:

Programming concepts

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. "RESOBJ".  
  
DATA DIVISION.  
  
WORKING-STORAGE SECTION.  
  
COPY fmcvars.  
  
01 buffer PIC X(2000) VALUE SPACES.  
  
PROCEDURE DIVISION.  
  
    PERFORM FmcjResultObjOfCurrentThread.  
    DISPLAY "Accessed result object of current thread".  
  
    SET hd1Result TO FmcjResultHandleReturnValue.  
    PERFORM FmcjResultRc.  
    DISPLAY "Return code: " intReturnValue.  
    MOVE 2000 TO bufferLength.  
    CALL "SETADDR" USING buffer messageBuffer.  
    PERFORM FmcjResultMessageText.  
    DISPLAY "Text      : " buffer.  
    CALL "SETADDR" USING buffer originBuffer.  
    PERFORM FmcjResultOrigin.  
    DISPLAY "Origin   : " buffer.  
    PERFORM FmcjResultParms.  
    SET hd1Vector TO FmcjStrVHandleReturnValue.  
  
    CALL "SETADDR" USING buffer elementBuffer.  
    PERFORM FmcjStrVNextResultParmElement.  
  
    PERFORM UNTIL pointerReturnValue = NULL  
        DISPLAY "Parameter : " buffer  
        PERFORM FmcjStrVNextResultParmElement  
    END-PERFORM.  
  
    STOP RUN.  
  
    COPY fmcperf.
```

Figure 9. Accessing a result object in COBOL (via PERFORM)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "RESOBJ".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.

01 buffer PIC X(2000) VALUE SPACES.

PROCEDURE DIVISION.

CALL "FmcjResultObjectOfCurrentThread"
RETURNING FmcjResultHandleReturnValue.
DISPLAY "Accessed result object of current thread".

SET hd1Result TO FmcjResultHandleReturnValue.
CALL "FmcjResultRc"
USING BY VALUE hd1Result
RETURNING intReturnValue.
DISPLAY "Return code: " intReturnValue.
MOVE 2000 TO bufferLength.
CALL "SETADDR" USING buffer messageBuffer.
CALL "FmcjResultMessageText"
USING BY VALUE hd1Result
messageBuffer
bufferLength
RETURNING pointerReturnValue.
DISPLAY "Text      : " buffer.
CALL "SETADDR" USING buffer originBuffer.
CALL "FmcjResultOrigin"
USING BY VALUE hd1Result
originBuffer
bufferLength
RETURNING pointerReturnValue.
DISPLAY "Origin   : " buffer.
CALL "FmcjResultParameters"
USING BY VALUE hd1Result
RETURNING FmcjStrVHandleReturnValue.
SET hd1Vector TO FmcjStrVHandleReturnValue.

CALL "SETADDR" USING buffer elementBuffer.
CALL "FmcjStringVectorNextResultParmElement"
USING BY VALUE hd1Vector
elementBuffer
bufferLength
RETURNING pointerReturnValue.

PERFORM UNTIL pointerReturnValue = NULL
DISPLAY "Parameter : " buffer
CALL "FmcjStringVectorNextResultParmElement"
USING BY VALUE hd1Vector
elementBuffer
bufferLength
RETURNING pointerReturnValue
END-PERFORM.

STOP RUN.

```

Figure 10. Accessing a result object in COBOL (via CALL)

Note: The SETADDR routine is shown in “Example of the use of strings” on page 148 .

Client/server communication and data access models

When you request actions from an MQSeries Workflow server or when you want to observe the result of actions, you can:

- Use a synchronous protocol to ask for an action and to view changes of the object which you used to call the action.
- Use a synchronous protocol to pull for data created or changed.
- Receive unsolicited information on created or changed objects pushed by the server.

For example, when you ask a process instance object to be started:

- As an immediate result, the state of the process instance is updated.
- You can query work items in order to view (pull for) new objects created.
- You can automatically receive new work items sent (pushed) to you.

Synchronous client/server communication

Applying a synchronous protocol means that you issue a request to an MQSeries Workflow server and then wait until you receive a response. All action API calls operate this way; your application (thread) is blocked until the response arrives or until your timeout set on the execution service object exceeds.

Note: The synchronous mode of communication is not supported for the XML message interface.

Asynchronous client/server communication

Applying an asynchronous protocol means that you issue a request to an MQSeries Workflow server but you do not wait until you receive a response. The `ExecuteProcessInstanceAsync()` API call operates this way; your application (thread) is not blocked and you can receive the response at a later time.

When you issue an action asynchronously, you do, however, receive an acknowledgement telling whether MQSeries Workflow accepted the request. You can also receive a correlation identification which you can use in order to receive a specific response. You can specify a user context in order to correlate a response received.

For example, when you ask a process instance to be executed asynchronously:

- As an immediate result, you are informed whether the request is accepted.
- When you specify a buffer to hold a correlation ID, you get an ID which you can use in the `Receive()` call to wait for that specific response.
- When you specify a user context, that context is returned to you as part of the response. You can use it for user-specific correlation.

Note: The asynchronous mode of communication is only supported in C, C++, and COBOL. All message-based requests are executed asynchronously.

The push data access model

Receiving unsolicited information pushed by an MQSeries Workflow server means that you set up communication in a way that you are automatically informed about new or changed objects.

Note: The push data access model is not supported in Java or the XML message interface.

In order to obtain information pushed by an MQSeries Workflow server:

1. The server must be asked to send data. This means that:
 - The settings of the applicable process instance must specify *REFRESH_POLICY PUSH*. This setting is inherited from the domain level, through the system group to the system and down to the process template. Each specification can be overwritten on a lower level.
 - The users must be logged on with a *Present* or *PresentHere* session mode, that is, they are enabled to receive information.
2. The application must use API calls in order to receive data pushed.

Provided that these prerequisites are fulfilled, the MQSeries Workflow execution server pushes changes on work items or notifications to the owner of the item:

1. On creation of the item.
2. On deletion of the item.
3. Whenever a primary property of the item changes - see "Accessor API calls" on page 84 for a definition of primary properties.

The caller of the action will, however, not receive such information because, as a result of the action, the transient object has already been updated with relevant data.

Changes to disabled work items are not pushed. Only the deletion of such work items is pushed.

Examples:

When a process instance is suspended and when its refresh policy is push, the MQSeries Workflow execution server notifies all owners of non-disabled items which are currently logged on as present.

When the description of a process instance is changed and when the refresh policy is push, the MQSeries Workflow execution server notifies all owners of process instance notifications which are currently logged on as present.

When a work item is transferred to user N by the owner of the work item and when the refresh policy of the associated process instance is push, the MQSeries Workflow execution server notifies user N when he/she is currently logged on as present. The owner of the work item as the requester of the action gets no additional notification.

Note: Filtering and sorting is left to the application. No indication about affected worklists is pushed to the client.

Receiving information

In C, C++, and COBOL, the ExecutionService object provides for a means to receive information (execution data) pushed by an MQSeries Workflow execution server at any time desired. The Receive() call blocks the calling application until information is received or until the specified timeout value has been reached. That is why an application, if possible, typically starts a separate thread for receiving data, in order to prevent blocking the entire application.

A timeout value of -1 specifies an indefinite wait time interval. Note that in this case you must ensure that you stop receiving data before your application ends.

Programming concepts

There is a `TerminateReceive()` API call which can be used to send a terminate indication to the receiving part of the application in order to provide notification that receiving data may end.

Notes:

1. A `Receive()` call survives a `Logoff()` call, which ends your session with an execution server. The execution server, however, stops pushing information when logoff has been executed. If you did not send a `TerminateReceive()` to the receiving application thread, you have to end that thread because of other knowledge. `TerminateReceive()` can only be called as long as a session exists.
2. If information is not received and therefore stays in the client input queue, the MQSeries expiration mechanism applies in order to eliminate such "dead" messages. The expiration time of client messages can be configured for MQSeries Workflow.

When receiving data, a correlation identification can be specified to indicate which information is to be read. If it is not specified or points to `FMCJ_NO_CORRELID`, then any data arriving is received; the correlation identification is set as the result of a successful receive.

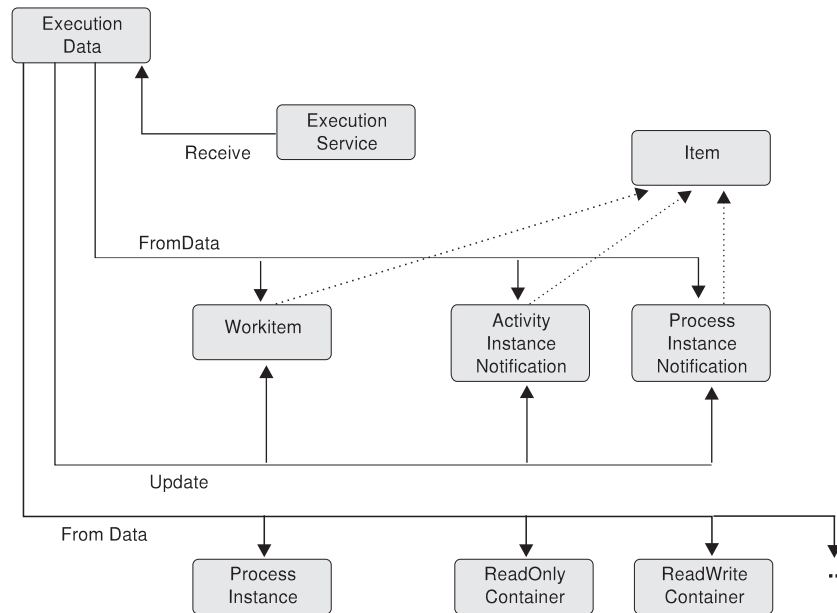


Figure 11. Handling data sent by an MQSeries Workflow server. Legend: \dashrightarrow Inheritance (C++); \rightarrow provides for access

Once execution data has been received, its type can be determined and the appropriate action can be called. For example, when a work item creation is indicated, a conversion from the execution data to a work item can be requested. When a work item change is indicated, the persistent object ID of the work item can be requested so that the appropriate work item can be updated.

When the response to an `ExecuteProcessInstanceAsync()` request is received, the process instance created and executed can be analyzed. For example, its state can be used to determine whether the process instance executed successfully. Its output container can then be read. If an error occurs, the error description can be examined.

An MQSeries Workflow session

In order to communicate with an MQSeries Workflow server, a session must have been established between the user and that server. The server is either identified explicitly (system at system group) or taken from the user's profile. If the information is not found in the user's profile, the configuration profile is read.

Note: Authentication is not required in order to use the XML message interface, that is, a session need not be established.

The session is established by logging on. From then on services can be requested from the server; the service object which represents the session between the user logging on and the server, is set up accordingly.

Logon requires that the administration server be up and running on the selected system, because the administration server manages sessions and checks the authentication of the user. It additionally ensures that any severe errors are written to the error log.

Any objects which are retrieved or created belong to the session where they have been queried or created. They carry the session identification so that further actions on those objects are executed in the same session with the authorization of the logged-on user.

A single application program or multiple application programs can allocate multiple service objects and log on with different users or the same user in parallel. Sessions are kept separate by the service objects. A single service object thus represents a single session. A second request to log on via a service object will be rejected if it comes from a different user. Otherwise, it is accepted but not repeated; the logon request has already been executed successfully.

A session can run in *default* mode or in *present* mode. When you are operating in the present session mode, activity instances which are started automatically can be scheduled on your behalf and you can receive information pushed by an MQSeries Workflow server. There can only be a single present session per user.

The service object provides for a timeout value to be set. This is the time the application waits for the answer from a server. The application is thus blocked during this time at a maximum. The timeout is specified in milliseconds. A value of -1 denotes an indefinite timeout value. The timeout value can be changed at any time.

Note: MQSeries Workflow uses the communication mechanisms of IBM MQSeries. If your application sets up its own signal handler, then you should refer to the *MQSeries Application Programming Guide*, especially the chapter *UNIX signal handling*, for restrictions imposed by MQSeries.

Querying data

There are essentially three means of querying data from an MQSeries Workflow server:

- A query via a service object, which returns all authorized objects. The number of objects returned to the client can be restricted by a filter and a threshold.
- A query using a persistent list definition, which returns all objects qualifying through the list definition.

Programming concepts

- A specific request, like the request for user settings or a refresh request for a specific object.

Note: Querying data is not supported by the XML message interface.

Persistent lists

A persistent list represents a set of objects of the same type. Moreover, all objects which are accessible through the list have the same characteristics. A list can be for public usage, that is, it is visible by all users, or for private usage, that is, it has an owner and is only visible by that owner.

The characteristics of the objects contained in the list are given by so-called *filter criteria*. The filter criteria specified and the authorization of the user issuing the query determine the contents of the list. This means that the contents itself is not stored persistently but determined when a query request is issued. This in particular means that a public list can deliver different results depending on the user who applies the query.

The number of objects transferred from the server to the client as the result of the query can be restricted by specifying a *threshold*. The threshold is used after *sort criteria* have been applied.

A list can be a process template list, a process instance list, or a worklist.

Using filters, sort criteria, and thresholds

A filter is a character string specifying criteria which must follow the rules stated by the filter syntax diagrams. Refer to the appropriate API calls for the exact syntax. Some sample criteria are shown here:

```
"NAME = 'MyProcessInstance'"
"NAME LIKE 'My*Ins?ance'"
"LAST_MODIFICATION_TIME > '1998-2-19 11:38:0'"
"STATE IN (READY,RUNNING)"
```

A sort criterion is a character string that must follow the rules stated by the sort criteria syntax diagrams. Refer to the appropriate API calls for the exact syntax. Some sample criteria are shown here:

```
"NAME ASC"
"NAME ASC, LAST_MODIFICATION_TIME DESC"
```

Objects are sorted on the server, that is, the code page of the server determines the sort sequence.

A threshold specifies the maximum number of objects to be returned to the client. That threshold is applied after the objects have been sorted.

Handling collections

The result of a query for a set of objects is a so-called vector of objects in C, C++, or COBOL, or an array of objects in Java.

A vector is provided by the caller and filled by the MQSeries Workflow API. The ownership of the vector elements, the objects, stays with the vector. They are automatically deleted when the vector is deleted.

Any objects returned are appended to the supplied vector. If you want to read the current objects only, you have to clear the vector before you call the query method. This means that you should erase all elements of the vector in the C++ API. This means that you should set the vector handle to 0 in C and COBOL.¹ If the vector handle is not initialized to 0, it *must* point to a vector of objects of the appropriate kind so that newly queried objects can be appended. In other words, any nonzero handle is used by C or COBOL in order to access a vector assumed to already exist.

In C or COBOL, the result of the query is the vector handle initialized to the set of objects, if a 0 handle was passed, otherwise the existing vector extended by the new objects. Special vector accessor functions are provided to access the objects (see below). When a vector element is read, it becomes an object of its own and thus has to be deleted when no longer used. Any operations on that object refer to the object only and do not have any impacts on the vector element from which the object was copied. For example, a Refresh() changes the object only but not its original copy within the vector. This means that a further iteration through the vector finds any elements unchanged.

In C++, the result of the query is an instance of vector<class T>. Access to the objects is gained via appropriate vector methods; refer to the STL documentation. When a vector element is read, a (const or non-const) reference to the object is returned. This means that a change of the object does actually change the vector element. A further iteration through the vector finds the elements changed.

An array is provided and filled by the MQSeries Workflow API. The ownership of the array elements, the objects, stays with the array.

C and COBOL vector accessor functions

Vector accessor functions are described below. This is because all these functions are similar in appearance and have similar requirements, even for different objects. They are all handled locally by the API, that is, they do not communicate with the server. Neither a connection to a server nor specific authorizations are required to execute.

Return codes

The C or COBOL functions or the result object can return the following codes. The number in parentheses shows their integer value:

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is expected, but 0 is passed.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is invalid; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_MORE_DATA(804)

The vector contains no or no more element.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

1. Declare a new vector handle or deallocate an existing vector object before reuse.

Programming concepts

Vector accessor functions allow for the operations listed below. 'Xxx' denotes a particular scope; for example, `FmcjXxxVectorFirstElement()` can stand for `FmcjProcessInstanceVectorFirstElement()`.

FmcjXxxVectorDeallocate()

Allows the application to deallocate the storage reserved for the specified transient vector object. All elements contained are also deallocated.

The handle is set to 0 so that it can no longer be used.

C

```
APIRET FMC_APIENTRY FmcjXxxVectorDeallocate(  
    FmcjXxxVectorHandle * hdlVector)
```

COBOL

```
FmcjXxxVectorDeallocate.  
  
    CALL    "FmcjXxxVectorDeallocate"  
          USING  
          BY REFERENCE  
          hdlVector  
          RETURNING  
          intReturnValue.
```

Parameters

hdlVector Input/Output. The address of the handle to the vector to be deallocated.

FmcjXxxVectorFirstElement()

Returns the first element of the vector. That element becomes an object on its own and has to be deallocated if no longer used. The vector is positioned to the next element.

If the vector is empty or if an error occurred, 0 (zero) is returned.

C

```
FmcjXxxHandle FMC_APIENTRY FmcjXxxVectorFirstElement(  
    FmcjXxxVectorHandle hdlVector )
```

COBOL

```
FmcjXxxVectorFirstElement.  
  
    CALL    "FmcjXxxVectorFirstElement"  
          USING  
          BY VALUE  
          hdlVector  
          RETURNING  
          FmcjXxxHandleReturnValue.
```

Parameters

hdlVector Input. The handle of the vector to be queried.

Return type

FmcjXxxHandle

The handle of the first element of the vector or 0.

FmcjXxxVectorNextElement()

Returns the vector element at the current vector position; the initial vector position is the first element. That element becomes an object on its own and has to be deallocated if no longer used. The vector is positioned to the next element.

If the vector is empty, if there are no more elements in the vector, or if an error occurred, 0 (zero) is returned.

C

```
FmcjXxxHandle FMC_APIENTRY FmcjXxxVectorNextElement(
    FmcjXxxVectorHandle hdVector )
```

COBOL

```
FmcjXxxVectorNextElement.
    CALL      "FmcjXxxVectorNextElement"
            USING
            BY VALUE
            hdVector
            RETURNING
            FmcjXxxHandleReturnValue.
```

Parameters

hdVector Input. The handle of the vector to be queried.

Return type

FmcjXxxHandle

The handle of the vector element at the current position or 0.

FmcjXxxVectorSize()

Returns the number of elements in the vector.

C

```
unsigned long FMC_APIENTRY FmcjXxxVectorSize(
    FmcjXxxVectorHandle hdVector )
```

COBOL

```
FmcjXxxVectorSize.
    CALL      "FmcjXxxVectorSize"
            USING
            BY VALUE
            hdVector
            RETURNING
            ulongReturnValue.
```

Parameters

Programming concepts

hdlVector Input. The handle of the vector to be queried.

Return type
unsigned long

The number of elements in the vector.

C examples

In the following, some C examples on how to read a vector are shown; note that you can start with a first element call or a next element call.

```

#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
    APIRET rc;
    FmcjExecutionServiceHandle service = 0;
    FmcjProcessInstanceVectorHandle hd1Vector = 0;
    FmcjProcessInstanceHandle hd1Instance = 0;
    unsigned long i = 0;
    unsigned long numElements = 0;
    char tInfo[FMC_PROCESS_INSTANCE_NAME_LENGTH] = "";

    FmcjGlobalConnect();

    FmcjExecutionServiceAllocate(&service);
    rc = FmcjExecutionServiceLogon( service,
                                   "ADMIN", "PASSWORD",
                                   Fmc_SM_Default, Fmc_SA_Reset
                                   );

    if ( rc != FMC_OK )
        return rc;
    printf("Logged on\n");
    rc= FmcjExecutionServiceQueryProcessInstances(
                                   service,
                                   FmcjNoFilter,
                                   FmcjNoSortCriteria,
                                   FmcjNoThreshold,
                                   &hd1Vector );

    if ( rc != FMC_OK )
        return rc;
    printf("Queried process instances\n");
    hd1Instance= FmcjProcessInstanceVectorFirstElement(hd1Vector);
    numElements= FmcjProcessInstanceVectorSize(hd1Vector);

    printf("Instances in the vector:\n");
    for( i=0; i< numElements; i++ )
    {
        printf("- name: %s\n",
               FmcjProcessInstanceName(hd1Instance,tInfo,
                                       FMC_PROCESS_INSTANCE_NAME_LENGTH));
        FmcjProcessInstanceDeallocate(&hd1Instance);
        hd1Instance= FmcjProcessInstanceVectorNextElement(hd1Vector) ;
    }

    FmcjProcessInstanceVectorDeallocate(&hd1Vecor);
    FmcjExecutionServiceLogoff(service);
    printf("Logged off\n");
    FmcjExecutionServiceDeallocate(&service);

    FmcjGlobalDisconnect();
    return FMC_OK;
}

```

Figure 12. Reading a vector in C (using First/NextElement() calls)

Programming concepts

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
    APIRET rc;
    FmcjExecutionServiceHandle service = 0;
    FmcjProcessInstanceVectorHandle hdlVector = 0;
    FmcjProcessInstanceHandle hdlInstance = 0;
    char tInfo[FMC_PROCESS_INSTANCE_NAME_LENGTH] = "";

    FmcjGlobalConnect();

    FmcjExecutionServiceAllocate(&service);
    rc = FmcjExecutionServiceLogon( service,
                                   "ADMIN", "PASSWORD",
                                   Fmc_SM_Default, Fmc_SA_Reset
                                   );

    if ( rc != FMC_OK )
        return rc;
    printf("Logged on\n");
    rc= FmcjExecutionServiceQueryProcessInstances(
        service,
        FmcjNoFilter,
        FmcjNoSortCriteria,
        FmcjNoThreshold,
        &hdlVector );

    if ( rc != FMC_OK )
        return rc;
    printf("Queried process instances\n");
    printf("Instances in the vector:\n");
    while (0 != (hdlInstance=FmcjProcessInstanceVectorNextElement(hdlVector)))
    {
        printf("- name: %s\n",
              FmcjProcessInstanceName(hdlInstance,tInfo,
                                       FMC_PROCESS_INSTANCE_NAME_LENGTH));
        FmcjProcessInstanceDeallocate(&hdlInstance) );
    }
    FmcjProcessInstanceVectorDeallocate(&hdlVector) );
    FmcjExecutionServiceLogoff(service);
    printf("Logged off\n");
    FmcjExecutionServiceDeallocate(&service);

    FmcjGlobalDisconnect();
    return FMC_OK;
}
```

Figure 13. Reading a vector in C (using NextElement() call only)

COBOL examples

In the following, some COBOL examples on how to read a vector are shown; note that you can start with a FirstElement or NextElement call.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VECTOR".

DATA DIVISION.

    WORKING-STORAGE SECTION.

        COPY fmcvars.
        COPY fmconst.
        COPY fmcrcs.

        01 localUserID    PIC X(30) VALUE z"ADMIN".
        01 localPassword  PIC X(30) VALUE z"PASSWORD".
        01 numElements    PIC 9(9) BINARY.
        01 i               PIC 9(9) BINARY.
        01 buffer         PIC X(64) VALUE SPACES.

    LINKAGE SECTION.

        01 retCode       PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.
    PERFORM FmcjESAllocate.

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK THEN GOBACK.
    DISPLAY "Logged on".

    CALL "SETADDR" USING FmcjNoFilter filter.
    CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
    MOVE FmcjNoThreshold TO threshold.
    PERFORM FmcjESQueryProcInsts.

    SET hd1Vector TO instances.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK THEN GOBACK.
    DISPLAY "Queried Process Instances".

    PERFORM FmcjPIVFirstElement.
    SET hd1Instance TO FmcjPIHandleReturnValue.
    PERFORM FmcjPIVSize.
    MOVE ulongReturnValue TO numElements.

    DISPLAY "Instances in the vector:".
    MOVE FMC-PROC-INST-NAME-LENGTH TO bufferLength.
    CALL "SETADDR" USING buffer instanceNameBuffer.
    PERFORM VARYING i FROM 0 BY 1 UNTIL i >= numElements
        PERFORM FmcjPIName
        DISPLAY "- name: " buffer
        PERFORM FmcjPIDeallocate
        PERFORM FmcjPIVNextElement
        SET hd1Instance TO FmcjPIHandleReturnValue
    END-PERFORM

```

Figure 14. Reading a vector in COBOL (using First/NextElement calls) (Part 1 of 2)

Programming concepts

```
PERFORM FmcjPIVDeallocate.  
PERFORM FmcjESLogoff.  
DISPLAY "Logged off".  
PERFORM FmcjESDeallocate.  
PERFORM FmcjGlobalDisconnect.  
MOVE FMC-OK TO retCode.  
GOBACK.  
  
COPY fmcperf.
```

Figure 14. Reading a vector in COBOL (using First/NextElement calls) (Part 2 of 2)


```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VECTOR".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID PIC X(30) VALUE z"ADMIN".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 buffer PIC X(64) VALUE SPACES.

LINKAGE SECTION.

01 retCode PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

PERFORM FmcjGlobalConnect.
PERFORM FmcjESAllocate.

CALL "SETADDR" USING localUserId userId.
CALL "SETADDR" USING localPassword passwordValue.
MOVE Fmc-SM-Default TO sessionMode.
MOVE Fmc-SA-Reset TO absenceIndicator.
PERFORM FmcjESLogon.

MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK THEN GOBACK.
DISPLAY "Logged on".

CALL "SETADDR" USING FmcjNoFilter filter.
CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
MOVE FmcjNoThreshold TO threshold.
PERFORM FmcjESQueryProcInsts.

SET hd1Vector TO instances.
MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK THEN GOBACK.
DISPLAY "Queried Process Instances".

DISPLAY "Instances in the vector:".
MOVE FMC-PROC-INST-NAME-LENGTH TO bufferLength.
CALL "SETADDR" USING buffer instanceNameBuffer.

PERFORM FmcjPIVNextElement.

PERFORM UNTIL FmcjPIHandleReturnValue = NULL
SET hd1Instance TO FmcjPIHandleReturnValue
PERFORM FmcjPIName
DISPLAY "- name: " buffer
PERFORM FmcjPIDeallocate
PERFORM FmcjPIVNextElement
END-PERFORM

```

Figure 15. Reading a vector in COBOL (using NextElement calls only) (Part 1 of 2)

Programming concepts

```
PERFORM FmcjPIVDeallocate.  
PERFORM FmcjESLogoff.  
DISPLAY "Logged off".  
PERFORM FmcjESDeallocate.  
PERFORM FmcjGlobalDisconnect.  
MOVE FMC-OK TO retCode.  
GOBACK.  
  
COPY fmcperf.
```

Figure 15. Reading a vector in COBOL (using NextElement calls only) (Part 2 of 2)

Java arrays

In Java, the result of a query for a set of objects is stored in arrays. The arrays are declared by you as a variable of the respective type, for example:

```
ProcessInstance[] processInstances;
```

With each new query, all existing objects in the array are deleted and the new objects are added.

The number of objects contained in an array is determined by accessing its length variable, for example:

```
processInstances.length
```

All array indexes start with 0 (zero). That is, valid index numbers are 0 to length-1. You access an object by providing its index number, for example, `processInstances[0]`. You should not save the index number of an object for later reuse, because the object can have a different index after each query, depending on the sort criteria and the number of objects returned.

Handling containers

A container represents input or output data of a process template, process instance, work item, or activity implementation at *runtime*. Each container is defined by a *data structure* which declares the container to be of the type of that data structure.

Data structure/container type

A data structure is uniquely identified by its name and contains an ordered list of *data members*. At runtime, it can become a stream of 32 KB passed between the client and the server.

The data structures and their usage as input containers or output containers are defined during modeling. A special data structure called `DEFAULT_DATA_STRUCTURE` is provided by MQSeries Workflow and contains no user-defined data members when installed. The `DEFAULT_DATA_STRUCTURE` cannot be deleted, but it can be extended during modeling.

Data member/container element

A data member of a data structure has a name and a *data type*. Data types are either basic and then `STRING`, `LONG`, `BINARY`, or `FLOAT`, or another data structure. Using a data structure as the data type of a data member (nesting) allows for recursive definitions of data members.

Programming concepts

A data member can represent a one-dimensional array. If a data member represents an array, the number of elements in that array is shown in parentheses ().

A data structure can have up to 512 user-defined data members. A data member that represents an array of data members counts with as many data members as it has elements.

Data members are specified using their fully qualified name within the container. The fully qualified name of a data member is a name in dot notation where the hierarchy of nested data members is presented from left to right, and their names are separated by a dot.

If a data member actually specifies an array of data members, the index number of a specific data member is specified in brackets ([n]) or parentheses ((n)).

When a data structure denotes the type of a container, then its data members (first level of any hierarchy) are also called *container elements*. They define the *structural members* of the container. When the data type of a container element (n-th level of any hierarchy) is a data structure (nesting), then that container element again has container elements or structural members.

Container elements of a basic data type are also called the *leaves* of the container. These are the members which can hold a value, that is, which can be asked for a value and which can be set to a new value.

For example, assume that the data structure PERSON describes an input container or output container and that PERSON has been defined as:

```
Name          STRING
Addr          ADDRESS
  Street      STRING
  POBOX       LONG(2)
```

PERSON has two structural data members named Name and Addr. Name is of basic data type STRING and Addr is of data type ADDRESS. That is the data structure ADDRESS is nested within the data structure PERSON.

The input or output container described by PERSON then has two container elements or structural members named Name and Addr, where Addr defines a structure by itself. The container elements or structural members of the container element Addr are Street and POBOX.

The leaves of the container, that is, the container elements which can carry a value, and their fully qualified names within the container are:

```
Name
Addr.Street
Addr.POBOX[0]
Addr.POBOX[1]
```

Note that since the size of the POBOX array is 2, the valid index numbers are 0 and 1. This is because all array indexes start with 0 (zero).

Also note that the fully qualified names are not prefixed with the name of the data structure PERSON. That data structure denotes the type of the container. There is

Programming concepts

only one exception to the rule, when the container itself is specified to be an array, for example, an array of PERSONs. Then, to set the name of a specific person, the fully qualified name is specified as

```
PERSON[i].Name
```

For detailed examples see “Chapter 6. Examples” on page 511.

In the XML message interface, arrays are depicted as a sequence of elements. Since the structure is given explicitly, names are not prefixed. For example:

```
<Name>
  <Addr>
    <Street></Street>
    <POBOX></POBOX>
    <POBOX></POBOX>
  </Addr>
</Name>
```

For more information refer to “XML message interface” on page 149.

Predefined data members

All containers automatically specify data members predefined by MQSeries Workflow. They can hold values associated with the operational characteristics of an activity or process. Predefined data members are data members that need not be defined by the modeler but are automatically available. They can be accessed by the container API. Their names start with the reserved character “_”.

Predefined data member values can be:

- Used to evaluate activity exit criteria.
- Accessed by activity implementations.
- Dynamically set to change the operational characteristics of subsequent activities.

Predefined data members provide for the flexibility of modelers. The decision on operational characteristics of a process or activity is taken at Runtime. They also provide activity implementations and support tools a means to access the operational characteristics through the use of API calls.

There are the following sets of predefined data members:

- Fixed data members
- Process information data members
- Activity information data members

Fixed data members provide information about the current activity instance. They *cannot be set* using an API call. An exception is the `_RC` data member, which should be set only if the program cannot otherwise define a return code (see the following).

Process information and activity information data members are associated with the operational characteristics of a process or activity. They operate the same way as any user-defined data members. This means that the values for specific operational characteristics of a process instance or activity instance can be accessed or changed just like the values for any other user-defined data member.

The following provides the fully qualified name and a brief description of each of the predefined data members.

There are no arrays of any predefined data member.

Fixed data members

Fixed data members `_ACTIVITY`, `_PROCESS`, and `_PROCESS_MODEL` *cannot be set* using API calls. Their values *can be read* using container API calls.

`_ACTIVITY`

This data member contains the fully-qualified name of the considered activity instance. The value of this data member is automatically set when the activity instance or an associated work item is started.

Data type: STRING

`_PROCESS`

This data member contains the name of the associated process instance. The value of this data member is automatically set when the activity instance or an associated work item is started.

Data type: STRING

`_PROCESS_MODEL`

This data member contains the name of the associated process model. The value of this data member is automatically set when the activity instance or an associated work item is started.

Data type: STRING

`_RC`

This data member contains the return code of the activity implementation. Typically it is used to evaluate exit and transition conditions. `_RC` is the only way for a CICS- or IMS-based application to set a return code. If the application does not set `_RC` explicitly (via the Container API), the field is automatically set to the exit code when program execution is completed.

Data type: LONG

Process information data members

Process information data members serve to dynamically specify properties of a process instance. In general, the process modeler can choose where values for process instance properties are to be obtained.

- Values can be inherited from a top-level process instance.
- Values can be obtained from the process information data members in the input container. They are then either set as default values or provided in the input container when the process instance is started.

If specified via the `DATA_FROM_INPUT_CONTAINER` indicator, the values of the process information data members are read by MQSeries Workflow when the process instance is started. If a value for a process information data member is not set, then a default value is used (see the detailed descriptions below).

`_PROCESS_INFO.Role`

A role that people assigned to an activity instance of the process instance must fulfill.

Any role set becomes an additional criterion to roles set for the activity instance. Only people who are members of all the specified roles are eligible.

If no role is set and no roles are specified for the activity instance, then no role criteria are applied.

Data type: STRING

Programming concepts

_PROCESS_INFO.Organization

The organization to which people must belong to receive work items of the process instance. This setting is only used if no organization is specified for the activity instance.

If no organization is set and no organization is specified for the activity instance, the default is the organization of the person who starts the process instance.

Data type: STRING

_PROCESS_INFO.ProcessAdministrator

The user ID of the person notified if:

- The process instance is expired.
- No person meets the criteria to perform an activity instance.
- No valid person has been specified for notification.
- The person notified that an activity instance is overdue has exceeded the time allowed for an action, that is, the second notification is sent.

If not set, the default process administrator is the person who starts the process instance.

Data type: STRING

_PROCESS_INFO.Duration

Specifies how long the process instance is allowed to take. The value is expressed in seconds.

If not set, the default is "Endless".

Data type: LONG

Activity information data members

Activity information data members serve to dynamically specify properties of an activity instance. In general, the process modeler can choose where values for activity instance properties are to be obtained.

- Values can be obtained from the activity information data members in the input container. They are then either set as default values or provided in the input container when an activity instance or associated work item is started.

If specified, the values of the activity information data members are read by MQSeries Workflow when the activity instance is scheduled. If a value is not set, then a default value is used (see the detailed descriptions below).

The following indicators specify that activity information data members are to be read:

- DONE_BY STAFF DEFINED_IN INPUT_CONTAINER
- NOTIFICATION DEFINED_IN INPUT_CONTAINER
- PRIORITY DEFINED_IN INPUT_CONTAINER

_ACTIVITY_INFO.Priority

The numeric value assigned as the priority of an activity instance.

MQSeries Workflow does not deduce any meaning from this value; it is just used for client purposes. Any integer value between 0 and 9 can be specified. If the value specified is invalid or the data member is not set, a default of 0 (zero) is used.

Data type: LONG

ACTIVITY_INFO.MembersOfRoles

The role or roles a person must fulfill to receive a work item for the activity instance. Multiple roles may be specified and are then to be separated by a semicolon (;).

Any role or roles set for this data member become an additional criterion to the role set for the process instance. Only people who are members of all the specified roles are eligible.

If not set, the role specified for the process instance is used. If no role is set for the process instance and no roles are specified for the activity instance, then no role criteria are applied.

Note: This specification is ignored if any specific people are set using the `ACTIVITY_INFO.People` data member.

Data type: STRING

ACTIVITY_INFO.CoordinatorOfRole

The role or roles a person must coordinate to receive a work item for the activity instance. Multiple roles to coordinate may be specified and are then to be separated by a semicolon (;).

To receive a work item, the eligible person must be assigned as coordinator of all the specified roles in addition to being a member of all roles specified for the process instance and for the activity instance.

If not set, the roles specified by the process instance and the activity instance are solely used. If no roles to be member of nor roles to coordinate have been specified, no role criteria are applied.

Note: This criterion is ignored if any specific people are set using the `ACTIVITY_INFO.People` data member.

Data type: STRING

ACTIVITY_INFO.Organization

The organization to which people must belong to receive work items of the activity instance.

If an organization is set using this data member, any organization set for the process instance is ignored.

If not set, the organization specified by the process instance is used. If no organization is set and no organization is specified for the process instance properties, the default is the organization of the person who starts the process instance.

Note: This criterion is ignored if any specific people are set using the `ACTIVITY_INFO.People` data member.

Data type: STRING

ACTIVITY_INFO.OrganizationType

This data member is used to indicate if a work item for the activity instance should be assigned to persons in a child organization.

To make all persons in the specified organization and all of its child organizations eligible, set the value of this data member to 0.

Programming concepts

To limit the persons who are eligible to the members of the specified organization and the managers of the first level of child organizations, set this data member to any nonzero value.

If not set, the default is 0. If no organization is set for the `_ACTIVITY_INFO.Organization` data member, any value set here is ignored.

Note: This criterion is ignored if any specific people are set using the `_ACTIVITY_INFO.People` data member.

Data type: long

`_ACTIVITY_INFO.LowerLevel`

The minimum level persons must have to receive work items of the activity instance. A value between 0 and 9 can be set. The default value is 0 (zero).

If the level specified here is greater than the value specified for the upper level, or if the level is not set, the default value of 0 (zero) is used.

Note: This criterion is ignored if any specific people are set using the `_ACTIVITY_INFO.People` data member.

Data type: LONG

`_ACTIVITY_INFO.UpperLevel`

The maximum level for persons to receive work items of the activity instance. A value between 0 and 9 can be set. The default value is 9.

If the level specified here is less than the value specified for the lower level, or the level is not set, the default value of 9 is used.

Note: This criterion is ignored if any specific people are set using the `_ACTIVITY_INFO.People` data member.

Data type: LONG

`_ACTIVITY_INFO.People`

This data member is used to specifically identify the people who should receive a work item of the activity instance. Multiple entries are possible and are then to be separated by a semicolon (;).

If any people are identified using this data member, any values set for data members `_ACTIVITY_INFO.MembersOfRoles`, `_ACTIVITY_INFO.CoordinatorOfRole`, `_ACTIVITY_INFO.Organization`, `_ACTIVITY_INFO.OrganizationType`, `_ACTIVITY_INFO.LowerLevel`, and `_ACTIVITY_INFO.UpperLevel` are ignored.

If no value is set, any values set for the above data members are used. If no values have been set for those, the values set for staff definition for the process instance are used.

If no values have been set for the process instance, the people in the organization and all child organizations of the process starter receive a work item for the activity instance.

Data type: STRING

`_ACTIVITY_INFO.PersonToNotify`

Used to identify the person to notify if the specified duration to complete the activity instance expires before the activity instance is complete.

If the user ID specified by the data member is invalid or the data member is not set, the process administrator is notified.

Data type: LONG

ACTIVITY_INFO.Duration

Used to specify the maximum number of seconds allowed to complete the activity.

If the activity is not completed before the specified duration, the defined person is notified.

If the value specified by the data member is invalid or the data member is not set, no notification occurs.

Data type: LONG

ACTIVITY_INFO.Duration2

Used to specify the maximum number of seconds allowed to act on an activity instance notification.

If the notification is not acted on before the specified number of seconds expires, the process administrator is notified.

If the value specified by the data member is invalid or the data member is not set, no notification occurs.

Data type: LONG

Determining the structure of an unknown container

There are various API calls in order to determine the structure of an unknown container and/or its leaves. Applied to a container, they return a collection of container elements. Once the collection of container elements is available, similar API calls can be recursively applied in order to step down through a nested structure.

Note: In the XML message interface, a container is always completely described in the message. An application can thus determine the structure of a container by analyzing the container in the message.

Determining the leaves

The following API calls allow to determine the number of leaves in a container or to retrieve the leaves themselves. When all leaves are requested, then not only the user-defined leaves or their leaf count are provided, but also the MQSeries Workflow predefined data members.

C

```
unsigned long FmcjContainerLeafCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerLeaves( FmcjContainerHandle handle )

unsigned long FmcjContainerAllLeafCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerAllLeaves( FmcjContainerHandle handle )
```

Programming concepts

C++

```
unsigned long LeafCount()

void Leaves( vector<FmcjContainerElement> const & leaves ) const

unsigned long AllLeafCount()

void AllLeaves( vector<FmcjContainerElement> const & leaves ) const
```

Java

```
public abstract int leafCount() throws FmcException

public abstract ContainerElement[] leaves() throws FmcException

public abstract int allLeafCount() throws FmcException

public abstract ContainerElement[] allLeaves() throws FmcException
```

COBOL

```
FmcjCLeafCount.

    CALL    "FmcjContainerLeafCount"
           USING
           BY VALUE
           hd1Container
           RETURNING
           ulongReturnValue.

FmcjCLeaves.

    CALL    "FmcjContainerLeaves"
           USING
           BY VALUE
           hd1Container
           RETURNING
           FmcjCEVHandleReturnValue.

FmcjCA11LeafCount.

    CALL    "FmcjContainerAllLeafCount"
           USING
           BY VALUE
           hd1Container
           RETURNING
           ulongReturnValue.

FmcjCA11Leaves.

    CALL    "FmcjContainerAllLeaves"
           USING
           BY VALUE
           hd1Container
           RETURNING
           FmcjCEVHandleReturnValue.
```

Parameters

handle Input. The handle of the container to be queried.
leaves Input/Output. The vector or array of container elements to be filled.

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

The container elements which are leaves.

unsigned long/int

The number of user-defined leaves or the number of all leaves, user-defined and predefined.

Determining the structural members

The following API calls allow to determine the number of structural members in a container or to retrieve the structural members themselves.

C

```
unsigned long FmcjContainerMemberCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerStructMembers( FmcjContainerHandle handle )
```

C++

```
unsigned long MemberCount()

void StructMembers( vector<FmcjContainerElement> const & members ) const
```

Java

```
public abstract int memberCount() throws FmcException

public abstract ContainerElement[] structMembers() throws FmcException
```

COBOL

```
FmcjCMemberCount.

    CALL    "FmcjContainerMemberCount"
           USING
           BY VALUE
           hd1Container
           RETURNING
           uLongReturnValue.

FmcjCStructMembers.

    CALL    "FmcjContainerStructMembers"
           USING
           BY VALUE
           hd1Container
           RETURNING
           FmcjCEVHandleReturnValue.
```

Parameters

Programming concepts

handle Input. The handle of the container to be queried.
members Input/Output. The vector or array of container elements to be filled.

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

The container elements which are part of the container.

unsigned long/int

The number of structural members in the container.

Determining the type

The following API calls provide the type of a container, that is, the name of the associated data structure.

C

```
char * FmcjContainerType( FmcjContainerHandle handle,  
                        char * containerTypeBuffer,  
                        unsigned long bufferLength )
```

C++

```
string Type()
```

Java

```
public abstract String type() throws FmcException
```

COBOL

```
FmcjCType.  
CALL "FmcjContainerType"  
  USING  
  BY VALUE  
  hdIContainer  
  containerTypeBuffer  
  bufferLength  
  RETURNING  
  pointerReturnValue.
```

Parameters

bufferLength Input. The length of the buffer to contain the container type; must be at least FMC_CONTAINER_TYPE_LENGTH bytes.

containerTypeBuffer

Input/Output. The buffer to contain the container type.

handle

Input. The handle of the container to be queried.

Return type

char*/string/String

The type of the container.

Analyzing a container element

Once a container element has been accessed, it can be asked for its properties, its name, whether it is a leaf and an array, or a structure itself. Calls to the container can then be applied recursively in order to step down through a nested structure.

Determining the name or type of a container element

The following API calls allow to determine the name of a container element or its type.

C

```
char* FmcjContainerElementName (FmcjContainerElementHandle handle,
                               char * buffer,
                               unsigned long bufferLength)

char* FmcjContainerElementFullName(FmcjContainerElementHandle handle,
                                   char * buffer,
                                   unsigned long bufferLength)

char* FmcjContainerElementType (FmcjContainerElementHandle handle,
                                 char * buffer,
                                 unsigned long bufferLength)
```

C++

```
string Name() const
string FullName() const
string Type() const
```

Java

```
public abstract String name() throws FmcException
public abstract String fullName() throws FmcException
public abstract String type() throws FmcException
```

Programming concepts

COBOL

```
FmcjCENAME.  
    CALL    "FmcjContainerElementName"  
          USING  
          BY VALUE  
            hd1Element  
            elementNameBuffer  
            bufferLength  
          RETURNING  
            pointerReturnValue.  
  
FmcjCEFULLNAME.  
    CALL    "FmcjContainerElementFullName"  
          USING  
          BY VALUE  
            hd1Element  
            elementNameBuffer  
            bufferLength  
          RETURNING  
            pointerReturnValue.  
  
FmcjCETYPE.  
    CALL    "FmcjContainerElementType"  
          USING  
          BY VALUE  
            hd1Element  
            containerTypeBuffer  
            bufferLength  
          RETURNING  
            pointerReturnValue.
```

Parameters

bufferLength Input. The length of the buffer to be filled.
buffer Input/Output. The buffer to contain the container element name or type.
handle Input. The handle of the container element to be queried.

Return type

char*/string/String

The name or type of the container.

Determining the structural properties of a container element

The following API calls allow to determine whether the considered container element is a leaf or a structure by itself and whether it is denoted to be an array.

C

```
bool FmcjContainerElementIsArray ( FmcjContainerElementHandle handle )  
  
bool FmcjContainerElementIsLeaf  ( FmcjContainerElementHandle handle )  
  
bool FmcjContainerElementIsStruct( FmcjContainerElementHandle handle )
```

C++

```
bool isArray () const
bool isLeaf () const
bool isStruct() const
```

Java

```
public abstract boolean isArray () throws FmcException
public abstract boolean isLeaf () throws FmcException
public abstract boolean isStruct() throws FmcException
```

COBOL

```
FmcjCEisArray.
    CALL    "FmcjContainerElementisArray"
           USING
           BY VALUE
           hdElement
           RETURNING
           boolReturnValue.

FmcjCEisLeaf.
    CALL    "FmcjContainerElementisLeaf"
           USING
           BY VALUE
           hdElement
           RETURNING
           boolReturnValue.

FmcjCEisStruct.
    CALL    "FmcjContainerElementisStruct"
           USING
           BY VALUE
           hdElement
           RETURNING
           boolReturnValue.
```

Parameters

handle Input. The handle of the container element to be queried.

Return type

boolean/bool An indicator whether the container element is an array, a leaf, or a structure.

Determining the leaves of a container element

The following API calls allow to determine the number of leaves of a container element or to retrieve the leaves themselves.

Programming concepts

C

```
unsigned long  
FmcjContainerElementLeafCount( FmcjContainerElementHandle handle )  
  
FmcjContainerElementVectorHandle  
FmcjContainerElementLeaves( FmcjContainerElementHandle handle )
```

C++

```
unsigned long LeafCount()  
  
void Leaves( vector<FmcjContainerElement> const & leaves ) const
```

Java

```
public abstract int leafCount() throws FmcException  
  
public abstract ContainerElement[] leaves() throws FmcException
```

COBOL

```
FmcjCELeafCount.  
    CALL    "FmcjContainerElementLeafCount"  
           USING  
           BY VALUE  
           hd1Element  
           RETURNING  
           u1ongReturnValue.  
  
FmcjCELeaves.  
    CALL    "FmcjContainerElementLeaves"  
           USING  
           BY VALUE  
           hd1Element  
           RETURNING  
           FmcjCEVHandleReturnValue.
```

Parameters

handle Input. The handle of the container to be queried.
leaves Input/Output. The vector or array of container elements to be filled.

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

The container elements which are leaves.

unsigned long/int

The number of user-defined leaves.

Determining the structural members of a container element

The following API calls allow to determine the number of structural members of a container element or to retrieve the structural members themselves.

C

```
unsigned long
FmcjContainerElementMemberCount( FmcjContainerElementHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerElementStructMembers( FmcjContainerElementHandle handle )
```

C++

```
unsigned long MemberCount()

void StructMembers( vector<FmcjContainerElement> const & members ) const
```

Java

```
public abstract int memberCount() throws FmcException

public abstract ContainerElement[] structMembers() throws FmcException
```

COBOL

```
FmcjCEMemberCount.

    CALL      "FmcjContainerElementMemberCount"
              USING
              BY VALUE
              hd1Element
              RETURNING
              uLongReturnValue.

FmcjCEStructMembers.

    CALL      "FmcjContainerElementStructMembers"
              USING
              BY VALUE
              hd1Element
              RETURNING
              FmcjCEVHandleReturnValue.
```

Parameters

handle Input. The handle of the container element to be queried.
members Input/Output. The vector or array of container elements to be filled.

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

The container elements which are structural members.

unsigned long/int

The number of structural members.

Programming concepts

Determining the elements of an array

The following API calls allow to determine the number of elements in an array or to retrieve the elements themselves.

C

```
unsigned long  
FmcjContainerElementCardinality( FmcjContainerElementHandle handle )  
  
FmcjContainerElementVectorHandle  
FmcjContainerElementArrayElements( FmcjContainerElementHandle handle )
```

C++

```
unsigned long Cardinality() const  
  
void ArrayMembers( vector<FmcjContainerElement> const & elements ) const
```

Java

```
public abstract int cardinality() throws FmcException  
  
public abstract ContainerElement[] arrayElements() throws FmcException
```

COBOL

```
FmcjCECardinality.  
CALL "FmcjContainerElementCardinality"  
    USING  
    BY VALUE  
    hd1Element  
    RETURNING  
    ulongReturnValue.  
  
FmcjCEArrayElements.  
CALL "FmcjContainerElementArrayElements"  
    USING  
    BY VALUE  
    hd1Element  
    RETURNING  
    FmcjCEVHandleReturnValue.
```

Parameters

handle

Input. The handle of the container element to be queried.

elements

Input/Output. The vector or array of container elements to be filled.

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

The container elements which are part of the queried array container element.

unsigned long

The cardinality of the array described by the container element.

Accessing a known container element

When you know the (dotted) name of a container element, that name can be used in order to directly access the container element without iterating and searching through the whole container structure.

C

```
APIRET FMC_APIENTRY FmcjContainerGetElement(
    FmcjContainerHandle    handle,
    char const *           qualifiedName,
    FmcjContainerElementHandle * element )
```

C++

```
APIRET GetElement( string const &    qualifiedName,
    FmcjContainerElement & element ) const
```

Java

```
public abstract
ContainerElement getElement( String qualifiedName ) throws FmcException
```

COBOL

```
FmcjCGetElement.
    CALL    "FmcjContainerGetElement"
           USING
           BY VALUE
           hd1Container
           qualifiedName
           BY REFERENCE
           element
           RETURNING
           intReturnValue.
```

Parameters

element Output. The container element.
handle Input. The handle of the container to be queried.
qualifiedName Input. The fully qualified name of the container element.

Return type

APIRET The return code from this API call.

Accessing a value of a container

The following API calls return the value of a container leaf.
FMC_ERROR_MEMBER_NOT_SET is returned if no information is available.

Programming concepts

When the leaf is an array of values, an index must be specified. Since an index is to be specified, the fully qualified name must be given without the index or its brackets.

```
C
unsigned long
    FMC_APIENTRY FmcjContainerArrayBinaryLength(
        FmcjContainerHandle handle,
        char const * qualified name,
        unsigned long index )

APIRET FMC_APIENTRY FmcjContainerArrayBinaryValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    FmcjBinary * value,
    unsigned long bufferLength )

unsigned long
    FMC_APIENTRY FmcjContainerBinaryLength(
        FmcjContainerHandle handle,
        char const * qualified name )

APIRET FMC_APIENTRY FmcjContainerBinaryValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    FmcjBinary * value,
    unsigned long bufferLength )
```

```
C
APIRET FMC_APIENTRY FmcjContainerArrayFloatValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    double * value )

APIRET FMC_APIENTRY FmcjContainerFloatValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    double * value )
    unsigned long bufferLength )
```

```
C
APIRET FMC_APIENTRY FmcjContainerArrayLongValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    long * value )

APIRET FMC_APIENTRY FmcjContainerLongValue(
    FmcjContainerHandle handle,
    long * value )
```

C

```

unsigned long
    FMC_APIENTRY FmcjContainerArrayStringLength(
        FmcjContainerHandle handle,
        char const * qualified name,
        unsigned long index )

APIRET FMC_APIENTRY FmcjContainerArrayStringValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    char * value,
    unsigned long bufferSize )

unsigned long
    FMC_APIENTRY FmcjContainerStringLength(
        FmcjContainerHandle handle,
        char const * qualified name )

APIRET FMC_APIENTRY FmcjContainerStringValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    char * value,
    unsigned long bufferSize )
    
```

C++

```

unsigned long BinaryLength( unsigned long index )

APIRET Value( string const & qualifiedName,
              unsigned long index,
              FmcjBinary * value,
              unsigned long bufferSize ) const

unsigned long BinaryLength()
    
```

C++

```

APIRET Value( string const & qualifiedName,
              unsigned long index,
              long & value ) const

APIRET Value( string const a qualifiedName,
              long & value ) const
    
```

C++

```

APIRET Value( string const & qualifiedName,
              unsigned long index,
              double & value ) const

APIRET Value( string const a qualifiedName,
              double & value ) const
    
```

Programming concepts

C++

```
APIRET Value( string const & qualifiedName,  
              unsigned long  index,  
              string &       value ) const  
  
APIRET Value( string const a qualifiedName,  
              string &       value ) const
```

Java

```
public abstract  
byte[] getBuffer2( String qualifiedName,  
                  int    index      ) throws FmcException  
  
public abstract  
byte[] getBuffer( String qualifiedName ) throws FmcException
```

Java

```
public abstract  
double getDouble2( String qualifiedName,  
                  int    index      ) throws FmcException  
  
public abstract  
double getDouble( String qualifiedName ) throws FmcException
```

Java

```
public abstract  
int getLong2( String qualifiedName,  
             int    index      ) throws FmcException  
  
public abstract  
int getLong( String qualifiedName ) throws FmcException
```

Java

```
public abstract  
String getString2( String qualifiedName,  
                  int    index      ) throws FmcException  
  
public abstract  
String getString( String qualifiedName ) throws FmcException
```

COBOL

```
FmcjArrayBinaryLength.
```

```
CALL    "FmcjContainerArrayBinaryLength"
        USING
        BY VALUE
        hdIContainer
        qualifiedName
        indexValue
        RETURNING
        ulongReturnValue.
```

```
FmcjArrayBinaryValue.
```

```
CALL    "FmcjContainerArrayBinaryValue"
        USING
        BY VALUE
        hdIContainer
        qualifiedName
        indexValue
        pointerValue
        dataLength
        RETURNING
        intReturnValue.
```

```
FmcjBinaryLength.
```

```
CALL    "FmcjContainerBinaryLength"
        USING
        BY VALUE
        hdIContainer
        qualifiedName
        RETURNING
        ulongReturnValue.
```

```
FmcjBinaryValue.
```

```
CALL    "FmcjContainerBinaryValue"
        USING
        BY VALUE
        hdIContainer
        qualifiedName
        pointerValue
        dataLength
        RETURNING
        intReturnValue.
```

Programming concepts

COBOL

```
FmcjCArrayFloatValue.  
  
    CALL    "FmcjContainerArrayFloatValue"  
          USING  
            BY VALUE  
              hd1Container  
              qualifiedName  
              indexValue  
            BY REFERENCE  
              doubleValue  
          RETURNING  
            intReturnValue.  
  
FmcjCFloatValue.  
  
    CALL    "FmcjContainerFloatValue"  
          USING  
            BY VALUE  
              hd1Container  
              qualifiedName  
            BY REFERENCE  
              doubleValue  
          RETURNING  
            intReturnValue.
```

COBOL

```
FmcjCArrayLongValue.  
  
    CALL    "FmcjContainerArrayLongValue"  
          USING  
            BY VALUE  
              hd1Container  
              qualifiedName  
              indexValue  
            BY REFERENCE  
              intValue  
          RETURNING  
            intReturnValue.  
  
FmcjCLongValue.  
  
    CALL    "FmcjContainerLongValue"  
          USING  
            BY VALUE  
              hd1Container  
              qualifiedName  
            BY REFERENCE  
              intValue  
          RETURNING  
            intReturnValue.
```


COBOL

```

FmcjCStringLength.

    CALL    "FmcjContainerArrayStringLength"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           indexValue
           RETURNING
           ulongReturnValue.

FmcjCStringValue.

    CALL    "FmcjContainerArrayStringValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           indexValue
           valueBuffer
           bufferLength
           RETURNING
           intReturnValue.

FmcjCStringLength.

    CALL    "FmcjContainerStringLength"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           RETURNING
           ulongReturnValue.

FmcjCStringValue.

    CALL    "FmcjContainerStringValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           valueBuffer
           bufferLength
           RETURNING
           intReturnValue.
    
```

Parameters

- bufferLength* Input. The length of the buffer available for passing the value; must be greater than or equal to the actual length. Use the appropriate Length() API calls to determine the actual length.
- handle* Input. The handle of the container to be queried.
- index* Input. When the leaf is an array, the index of the array element to be queried.
- isArray* Input. If set to *True*, an array is to be queried and the index is used.
- qualifiedName* Input. The fully qualified name of the leaf within the container.
- value* Output. The value of the leaf.

Programming concepts

Return type

byte[]/double/int/String

The leaf value.

unsigned long

The minimum required buffer length for reading the value.

APIRET

The return code from this API call.

Accessing a value of a container element

The following API calls return the value of a container element leaf. When the leaf is an array of values, an index must be specified.

FMC_ERROR_MEMBER_NOT_SET is returned if no information is available. Note that, in contrast to querying container leaves, the name of the leaf need not be specified because the container element itself is the leaf queried.

```
C
unsigned long
    FMC_APIENTRY FmcjContainerElementArrayBinaryLength(
        FmcjContainerElementHandle handle,
        unsigned long index )

APIRET FMC_APIENTRY FmcjContainerElementArrayBinaryValue(
    unsigned long index,
    FmcjBinary * value,
    unsigned long bufferLength )

unsigned long
    FMC_APIENTRY FmcjContainerElementBinaryLength(
        FmcjContainerElementHandle handle )

APIRET FMC_APIENTRY FmcjContainerElementBinaryValue(
    FmcjContainerElementHandle handle,
    FmcjBinary * value,
    unsigned long bufferLength )
```

```
C
APIRET FMC_APIENTRY FmcjContainerElementArrayFloatValue(
    FmcjContainerElementHandle handle,
    unsigned long index,
    double * value )

APIRET FMC_APIENTRY FmcjContainerElementFloatValue(
    FmcjContainerElementHandle handle,
    double * value )
```

```
C
APIRET FMC_APIENTRY FmcjContainerElementArrayLongValue(
    FmcjContainerElementHandle handle,
    unsigned long index,
    long * value )

APIRET FMC_APIENTRY FmcjContainerElementLongValue(
    FmcjContainerElementHandle handle,
    long * value )
```

C

```

unsigned long
    FMC_APIENTRY FmcjContainerElementArrayStringLength(
        FmcjContainerElementHandle handle,
        unsigned long index )

APIRET FMC_APIENTRY FmcjContainerElementArrayStringValue(
    FmcjContainerElementHandle handle,
    unsigned long index,
    char * value,
    unsigned long bufferLength )

unsigned long
    FMC_APIENTRY FmcjContainerElementStringLength(
        FmcjContainerElementHandle handle )

APIRET FMC_APIENTRY FmcjContainerElementStringValue(
    FmcjContainerElementHandle handle,
    char * value,
    unsigned long bufferLength )

```

C++

```

unsigned long BinaryLength( unsigned long index )

APIRET Value( unsigned long index,
              FmcjBinary * value,
              unsigned long bufferLength ) const

unsigned long BinaryLength()

APIRET Value( FmcjBinary * value,
              unsigned long bufferLength ) const

```

C++

```

APIRET Value( unsigned long index,
              long & value ) const

APIRET Value( long & value ) const

APIRET Value( unsigned long index,
              double & value ) const

APIRET Value( double & value ) const

APIRET Value( unsigned long index,
              string & value ) const

APIRET Value( string & value ) const

```

Programming concepts

Java

```
public abstract
byte[] getBuffer2( int index ) throws FmcException

public abstract
byte[] getBuffer() throws FmcException

public abstract
double getDouble2( int index ) throws FmcException

public abstract
double getDouble() throws FmcException

public abstract
int getLong2( int index ) throws FmcException

public abstract
int getLong() throws FmcException

public abstract
String getString2( int index ) throws FmcException

public abstract
String getString() throws FmcException
```

COBOL

FmcjCEArrayBinaryLength.

```
CALL    "FmcjContainerElementArrayBinaryLength"  
        USING  
        BY VALUE  
        hdElement  
        indexValue  
        RETURNING  
        ulongReturnValue.
```

FmcjCEArrayBinaryValue.

```
CALL    "FmcjContainerElementArrayBinaryValue"  
        USING  
        BY VALUE  
        hdElement  
        indexValue  
        pointerValue  
        dataLength  
        RETURNING  
        intReturnValue
```

FmcjCEBinaryLength.

```
CALL    "FmcjContainerElementBinaryLength"  
        USING  
        BY VALUE  
        hdElement  
        RETURNING  
        ulongReturnValue.
```

FmcjCEBinaryValue.

```
CALL    "FmcjContainerElementBinaryValue"  
        USING  
        BY VALUE  
        hdElement  
        pointerValue  
        dataLength  
        RETURNING  
        intReturnValue.
```

Programming concepts

COBOL

```
FmcjCEArrayFloatValue.  
  
    CALL    "FmcjContainerElementArrayFloatValue"  
          USING  
            BY VALUE  
              hd1Element  
              indexValue  
            BY REFERENCE  
              doubleValue  
            RETURNING  
              intReturnValue.  
  
FmcjCEFloatValue.  
  
    CALL    "FmcjContainerElementFloatValue"  
          USING  
            BY VALUE  
              hd1Element  
            BY REFERENCE  
              doubleValue  
            RETURNING  
              intReturnValue.
```

COBOL

```
FmcjCEArrayLongValue.  
  
    CALL    "FmcjContainerElementArrayLongValue"  
          USING  
            BY VALUE  
              hd1Element  
              indexValue  
            BY REFERENCE  
              intValue  
            RETURNING  
              intReturnValue.  
  
FmcjCELONGValue.  
  
    CALL    "FmcjContainerElementLongValue"  
          USING  
            BY VALUE  
              hd1Element  
            BY REFERENCE  
              intValue  
            RETURNING  
              intReturnValue.
```

COBOL

```

FmcjCEArrayStringLength.

    CALL    "FmcjContainerElementArrayStringLength"
           USING
           BY VALUE
           hdElement
           indexValue
           RETURNING
           ulongReturnValue.

FmcjCEArrayStringValue.

    CALL    "FmcjContainerElementArrayStringValue"
           USING
           BY VALUE
           hdElement
           indexValue
           valueBuffer
           bufferLength
           RETURNING
           intReturnValue.

FmcjCEStringLength.

    CALL    "FmcjContainerElementStringLength"
           USING
           BY VALUE
           hdElement
           RETURNING
           ulongReturnValue.

FmcjCEStringValue.

    CALL    "FmcjContainerElementStringValue"
           USING
           BY VALUE
           hdElement
           valueBuffer
           bufferLength
           RETURNING
           intReturnValue.
    
```

Parameters

- bufferLength* Input. The length of the buffer available for passing the value; must be greater than or equal to the actual length. Use the appropriate Length() API calls to determine the actual length.
- handle* Input. The handle of the container element to be queried.
- index* Input. When the leaf is an array, the index of the array element to be queried.
- value* Output. The value of the leaf.

Return type

byte[]/double/int/String

The leaf value.

unsigned long

The minimum required buffer length for reading the value.

APIRET

The return code from this API call.

Programming concepts

Setting a value of a container

The following API calls allow to set the value of a container leaf.

When the leaf is an array of values, an index must be specified. Since an index is to be specified, the fully qualified name must be given without the index and its parentheses.

```
C
APIRET FMC_APIENTRY FmcjContainerSetArrayBinaryValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    FmcjBinary const * value,
    unsigned long dataLength )

APIRET FMC_APIENTRY FmcjContainerSetBinaryValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    FmcjBinary const * value,
    unsigned long dataLength )
```

```
C
APIRET FMC_APIENTRY FmcjContainerSetArrayFloatValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    double value )

APIRET FMC_APIENTRY FmcjContainerSetFloatValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    double value )
```

```
C
APIRET FMC_APIENTRY FmcjContainerSetArrayLongValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    long value )

APIRET FMC_APIENTRY FmcjContainerSetLongValue(
    FmcjContainerHandle handle,
    long value )
```


C

```
APIRET FMC_APIENTRY FmcjContainerSetArrayStringValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    unsigned long index,
    char const * value )

APIRET FMC_APIENTRY FmcjContainerSetStringValue(
    FmcjContainerHandle handle,
    char const * qualifiedName,
    char const * value )
```

C++

```
APIRET Value( string const & qualifiedName,
    unsigned long index,
    FmcjBinary const * value,
    unsigned long dataLength ) const

APIRET Value( string const & qualifiedName,
    FmcjBinary const * value,
    unsigned long dataLength ) const
```

C++

```
APIRET Value( string const & qualifiedName,
    unsigned long index,
    long value ) const

APIRET Value( string const a qualifiedName,
    long value ) const
```

C++

```
APIRET Value( string const & qualifiedName,
    unsigned long index,
    double value ) const

APIRET Value( string const a qualifiedName,
    double value ) const
```

C++

```
APIRET Value( string const & qualifiedName,
    unsigned long index,
    string const & value ) const

APIRET Value( string const & qualifiedName,
    string const & value ) const
```

Programming concepts

Java

```
public abstract
void setBuffer2( String qualifiedName,
                int index,
                byte value []) throws FmcException
```

```
public abstract
void setBuffer( String qualifiedName,
               byte value[] ) throws FmcException
```

Java

```
public abstract
void setDouble2( String qualifiedName,
                int index,
                double value ) throws FmcException
```

```
public abstract
void setDouble( String qualifiedName,
               double value ) throws FmcException
```

Java

```
public abstract
void setLong2( String qualifiedName,
              int index,
              long value ) throws FmcException
```

```
public abstract
void setLong( String qualifiedName,
             long value ) throws FmcException
```

Java

```
public abstract
void setString2( String qualifiedName,
                int index,
                String value ) throws FmcException
```

```
public abstract
void setString( String qualifiedName,
               String value ) throws FmcException
```

COBOL

```

FmcjRWSetArrayBinaryValue.

    CALL    "FmcjReadWriteContainerSetArrayBinaryValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           indexValue
           pointerValue
           dataLength
           RETURNING
           intReturnValue.

FmcjRWSetBinaryValue.

    CALL    "FmcjReadWriteContainerSetBinaryValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           pointerValue
           dataLength
           RETURNING
           intReturnValue.
    
```

COBOL

```

FmcjRWSetArrayFloatValue.

    CALL    "FmcjReadWriteContainerSetArrayFloatValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           indexValue
           doubleValue
           RETURNING
           intReturnValue.

FmcjRWSetFloatValue.

    CALL    "FmcjReadWriteContainerSetFloatValue"
           USING
           BY VALUE
           hdIContainer
           qualifiedName
           doubleValue
           RETURNING
           intReturnValue.
    
```

Programming concepts

COBOL

```
FmcjRWCSetsArrayLongValue.  
  
    CALL    "FmcjReadWriteContainerSetArrayLongValue"  
          USING  
          BY VALUE  
          hdIContainer  
          qualifiedName  
          indexValue  
          intValue  
          RETURNING  
          intReturnValue.  
  
FmcjRWCSetsLongValue.  
  
    CALL    "FmcjReadWriteContainerSetLongValue"  
          USING  
          BY VALUE  
          hdIContainer  
          qualifiedName  
          intValue  
          RETURNING  
          intReturnValue.
```

COBOL

```
FmcjRWCSetsArrayStringValue.  
  
    CALL    "FmcjReadWriteContainerSetArrayStringValue"  
          USING  
          BY VALUE  
          hdIContainer  
          qualifiedName  
          indexValue  
          pointerValue  
          RETURNING  
          intReturnValue.  
  
FmcjRWCSetsStringValue.  
  
    CALL    "FmcjReadWriteContainerSetStringValue"  
          USING  
          BY VALUE  
          hdIContainer  
          qualifiedName  
          pointerValue  
          RETURNING  
          intReturnValue.
```

Parameters

| | |
|----------------------|---|
| <i>dataLength</i> | Input. The length of the binary value. |
| <i>handle</i> | Input. The handle of the container to be set. |
| <i>index</i> | Input. When the leaf is an array, the index of the array element to be set. |
| <i>isArray</i> | Input. If set to <i>True</i> , an array element is to be set and the index is used. |
| <i>qualifiedName</i> | Input. The fully qualified name of the leaf within the container. |
| <i>value</i> | Input. The value of the leaf. Note that values for leaves of type BINARY must be specified as a sequence of two-digit hexadecimal |

numbers. For example, the string 'abc<cr><lf>' would be represented as '6162630d0a' (where <cr> denotes the ASCII 'carriage return' character and <lf> denotes the ASCII line-feed character).

Return type

APIRET The return code from this API call.

Return codes/FmcException

The following return codes can be issued or described by the result object, or the following exceptions can be thrown. The number in parentheses indicates the integer value:

FMC_OK(0) The API call completed successfully.

FMC_ERROR_BUFFER(800)

The provided buffer is too small.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is expected, but 0 is passed.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the server.

FMC_ERROR_FORMAT(117)

The qualified name does not conform to the syntax rules.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is invalid; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_MEMBER_CANNOT_BE_SET(115)

The specified member is an MQSeries Workflow predefined fixed data member; it is for information only.

FMC_ERROR_MEMBER_NOT_FOUND(112)

The specified member is not part of the container or container element.

FMC_ERROR_MEMBER_NOT_SET(113)

The specified member has no value.

Monitoring a process instance

MQSeries Workflow allows for obtaining a monitor for a specified process instance. A process instance monitor typically allows for:

- Observing the progress of a process instance execution.
- Determining the state of execution, that is, to determine which activity instance is currently in progress, is waiting to be executed by whom, is InError and waiting for some action. It allows to determine whether notifications occurred because the maximum work time was exceeded.
- Viewing the history of execution, that is, what path has been taken through the process instance and why. It allows to determine where the bottlenecks of execution are or where the most time-consuming parts are.

Note: Monitoring a process instance is not supported in the XML message interface.

Programming concepts

Obtaining a process instance monitor

Once a process instance² has been accessed, a **process instance monitor** can be obtained. The transient process instance monitor object then represents all information about activity instances directly contained in the described process instance as well as all information on control connector instances connecting those activity instances.

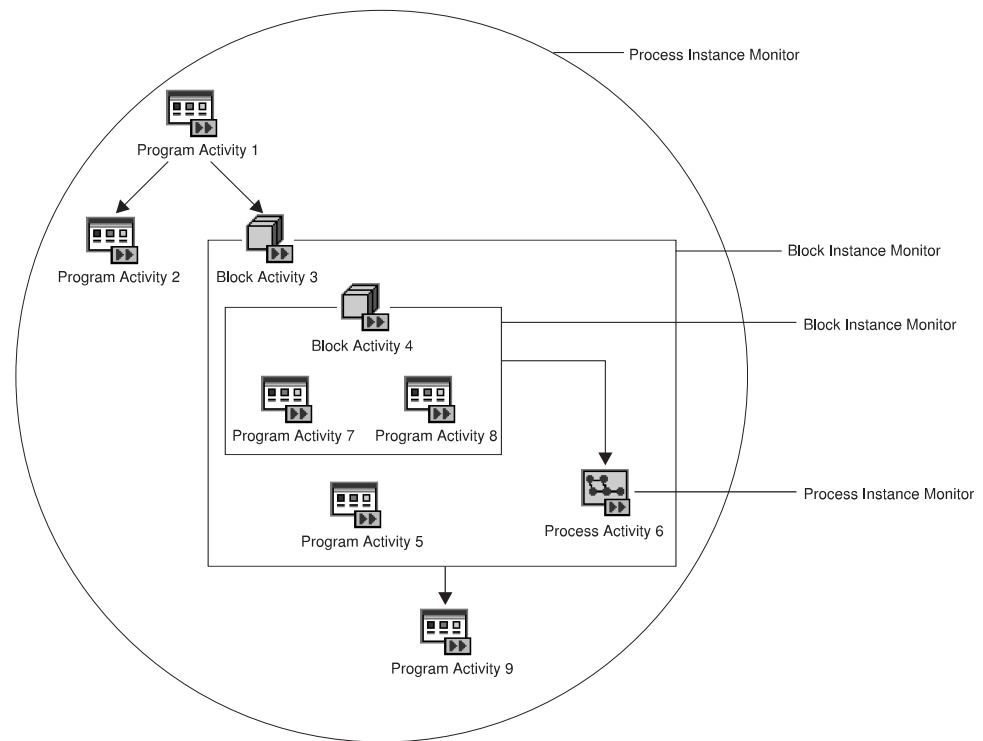


Figure 16. Process instance monitors and block instance monitors

For example, the illustrated process instance monitor describes three program activities, *Program Activity 1*, *Program Activity 2*, and *Program Activity 9*, and an activity of type Block, *Block Activity 3*. There are three control connectors between these activities.

The process instance monitor can then be asked for the activity instances and the control connector instances described and their properties can be determined, for example, the state of the activity and its graphical layout, or the result of control connector instance evaluation and activities to connect or bend points to be drawn.

When an activity of type *Block* is encountered, it is possible to obtain its **block instance monitor**. Similar to a process instance monitor, a block instance monitor object represents all information about activity instances directly contained in the described block activity instance as well as all information on control connector instances connecting those activity instances. For example, the block instance monitor of *Block Activity 3* describes *Block Activity 4*, *Program Activity 5*, and *Process Activity 6*. There is a control connector between *Block Activity 4* and *Process Activity 6*.

². or activity instance or a (work) item

When an activity of type *Process* is encountered, it is again possible to obtain its process instance monitor, either via the embracing monitor object or by retrieving the implementing (sub)process instance of the activity and then obtaining the associated process instance monitor. The process instance monitor obtained is a monitor which is completely separate from any other process instance monitor.

When obtaining a process instance monitor, it is possible to use the *deep* option in order to specify that *all* monitors for activities of kind *Block* are to be returned from the MQSeries Workflow execution server in the same step. The block instance monitors then all show the state of the process instance at this retrieval time. This means, when a block instance monitor is obtained via an API call, the API finds this monitor in its cache and provides it to the caller. When the *deep* option is not used, it can happen that a block instance monitor is not available. The API then automatically fetches the requested monitor from the execution server; it then represents a newer state than the ones previously retrieved.

Note: The *deep* option is not yet supported.

Ownership of monitors

As any other transient object, a process instance monitor is owned by the caller of the API. When a process instance monitor is no longer needed, you should delete/deallocate the object.

A block instance monitor, however, is considered to be part of a process instance monitor. It is cached by the API as part of the process instance monitor. It cannot be deallocated in C or COBOL. Deletion in C++ only deletes the C++ representation but not the block instance monitor itself in the API cache. Block instance monitors are **automatically deleted** when the owning process instance monitor is deleted/deallocated. This means that block instance monitor objects or handles can only be used as long as the containing process instance monitor exists. When the process instance monitor no longer exists, using a block instance monitor object or handle will return unexpected results; your program can even abend since the usage of a nonexisting object or handle violates the MQSeries Workflow *programming by contract* concept.

Authorization considerations

In general, authorization is granted to persons, either explicitly or implicitly. Implicitly means that the authority has been given as the result of performing some MQSeries Workflow action; performing that action can itself request some specific authority. See also *MQSeries Workflow for OS/390: Customization and Administration*.

Special authority is granted to a person playing the role of a *system administrator*. The system administrator has all privileges except on (work) items. Only the owner of a (work) item can issue any actions; the system administrator can, however, transfer the (work) item to himself. The system administrator role must be assigned to a single person at any time.

When a process instance is started, its *process administrator* is determined. The person determined to be the process administrator receives process administration rights for that process instance.

Programming concepts

The person who is to become the process administrator of a process instance is specified when the process model is defined. Identification of the process administrator can be done in the following ways:

- Specification of a user identification for the PROCESS_ADMINISTRATOR keyword. In this case, the process administrator is already known when the process model is defined.
- Specification of a member in the process input container via the PROCESS_ADMINISTRATOR TAKEN_FROM specification.
- Specification of DATA FROM INPUT_CONTAINER. The process administrator is then taken from the process information member _PROCESS_INFO.ProcessAdministrator field in the input container (see “Process information data members” on page 33 for details).

The following table shows the authorizations and the MQSeries Workflow functions which can be called when that authority has been granted. The E/I (Explicit/Implicit) column indicates how the authorization is granted to persons.

Note: For the programming language APIs, once a user has been authenticated to MQSeries Workflow (logged on), he can retrieve all objects he is authorized to see without any further special authorization. These are all objects he has created and all objects which are not specially secured or which are for public usage.

Table 2. Authorization for persons

| Name | E/I | Authorized Functions |
|--|-----|--|
| Authorization definition authorization | E | Create, update, and delete authorization information. Retrieve and update passwords. The appropriate FDL authorization keyword is AUTHORIZATION. |
| Operation administration authorization | E | Can perform all operation administration functions. The appropriate FDL authorization keyword is OPERATION. |
| Staff definition authorization | E | Create, retrieve, update, and delete staff information. As such, it includes authorization definition authorization. Create, retrieve, update, and delete public and private process instance lists, process template lists, and worklists. The appropriate FDL authorization keyword is STAFF. |
| Topology definition authorization | E | Create, retrieve, update, and delete topology information. The appropriate FDL authorization keyword is TOPOLOGY. |
| Process modeling authorization | E | Create, retrieve, update, and delete process models and process templates. The appropriate FDL authorization keyword is PROCESS_MODELING. |

Table 2. Authorization for persons (continued)

| Name | E/I | Authorized Functions |
|--------------------------------------|-----|---|
| Process authorization | E | <p>Can perform the following process instance functions if the process instance does not belong to any category. If the process instance does belong to a category, you must be authorized for all categories or for that specific category:</p> <ul style="list-style-type: none"> • Create • Start • Create and start • Set process instance name • Query • Refresh <p>Can perform the following process template functions if the process template does not belong to any category. If the process template does belong to a category, you must be authorized for all categories or for that specific category:</p> <ul style="list-style-type: none"> • Query • Refresh <p>The appropriate FDL authorization keyword is PROCESS_CATEGORY.</p> |
| Process administration authorization | E | <p>Has process authorization and can perform the following additional process instance functions if the process instance does not belong to any category. If the process instance does belong to a category, you must be authorized with administration rights for all categories or for that specific category:</p> <ul style="list-style-type: none"> • Delete • Restart • Resume • Suspend • Terminate <p>Can perform the following work item functions on the assigned work item for all process instances if the process instance does not belong to any category. If the process instance does belong to a category, you must be authorized for all categories or for that specific category:</p> <ul style="list-style-type: none"> • Force-finish • Force-restart <p>The appropriate FDL authorization keyword is PROCESS_CATEGORY AS ADMINISTRATOR.</p> |
| Process administrator | I | Has process administration authority for the appropriate process instance. |
| Process creator | I | <p>Can perform the following process instance functions:</p> <ul style="list-style-type: none"> • Set process instance name • Delete, if not yet started • Query • Refresh • Start |

Programming concepts

Table 2. Authorization for persons (continued)

| Name | E/I | Authorized Functions |
|---------------------|-----|--|
| Work item authority | E | Can perform the following functions on (work) items for all persons if you are authorized for all persons or for selected persons: <ul style="list-style-type: none">• Query• Refresh• Transfer The appropriate FDL authorization keyword is WORKITEMS_OF. |
| Workitem owner | I | Can perform all functions on the assigned (work) item except: <ul style="list-style-type: none">• Force Finish• Force Restart |

Types of API calls

MQSeries Workflow API calls can be divided into several categories which characterize the kind and behavior of the request to be executed.

| | |
|-------------------------|---|
| Basic | Manage transient objects |
| Accessor/mutator | Read and update properties of transient objects |
| Action | Read or manipulate persistent objects |
| Activity implementation | Deal with containers from within an activity implementation |

Basic and accessor API calls are described in some detail, but still generally, in the following paragraphs. This is because all these API calls are similar in appearance and have similar requirements, even for different objects. They are all handled locally by the API, that is, they do not communicate with the server. The API calls of the other categories are described separately in “Chapter 5. API action and activity implementation calls” on page 273. These API calls require client/server communication or communication with the program execution server.

Basic API calls

Basic API calls are provided primarily to allow transient objects to be allocated or constructed and deallocated or destructed. They allow for the construction of supporting objects like service objects. They allow for the destruction of such objects as well as for the destruction of transient representations of persistent objects allocated implicitly by the MQSeries Workflow API. Refer also to “Object and memory management” on page 11.

Basic API calls are only provided in the various APIs as far as needed. For example, the Java language supports only `IsComplete()`, `IsEmpty()`, and the Agent constructor.

Because of the nature of transient objects, neither a connection to a server nor some specific authorization is required to execute.

Return codes

The C and COBOL calls and the MQSeries Workflow result object can return the following codes, the number in parentheses shows their integer value:

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is expected, but 0 is passed.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is invalid; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_NAME(134)

The name provided is invalid; it is a null pointer or it does not conform to the syntax rules.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

Basic API calls allow for the basic operations listed below; **Xxx** and **Yyy** denote a particular class or scope, for example, `FmcjXxxEqual()` can stand for `FmcjProcessInstanceEqual()`.

Allocation

The following API calls allow the application to set up the respective object. This is needed for supporting objects like string vectors. Transient objects representing persistent objects are allocated implicitly by the MQSeries Workflow API when persistent objects are created or queried from an MQSeries Workflow server.

In the C++ API, constructors are made public for **all** classes so that their instances can be put into collections. When they are called by the application, empty objects of the appropriate class are created; they do not yet represent a persistent object.

All constructed objects are transient.

C

```
APIRET FMC_APIENTRY
FmcjExecutionServiceAllocate( FmcjExecutionServiceHandle * service )

APIRET FMC_APIENTRY FmcjExecutionServiceAllocateForGroup(
    char const *          systemGroup,
    FmcjExecutionServiceHandle * service )

APIRET FMC_APIENTRY FmcjExecutionServiceAllocateForSystem(
    char const *          system,
    char const *          systemGroup,
    FmcjExecutionServiceHandle * service )

APIRET FMC_APIENTRY
FmcjStringVectorAllocate( FmcjStringVectorHandle * hdlVector )
```

C++

```
FmcjXxx()

FmcjDateTime( bool initWithCurrentDateTime= false )

FmcjDateTime( unsigned short year,   unsigned short month,
              unsigned short day,    unsigned short hour,
              unsigned short minute, unsigned short second )
```

Programming concepts

Java

Agent()

COBOL

```
FmcjESAllocate.  
  
    CALL    "FmcjExecutionServiceAllocate"  
           USING  
           BY REFERENCE  
           serviceValue  
           RETURNING  
           intReturnValue.  
  
FmcjESAllocateForGroup.  
  
    CALL    "FmcjExecutionServiceAllocateForGroup"  
           USING  
           BY VALUE  
           systemGroup  
           BY REFERENCE  
           serviceValue  
           RETURNING  
           intReturnValue.  
  
FmcjESAllocateForSyst.  
  
    CALL    "FmcjExecutionServiceAllocateForSystem"  
           USING  
           BY VALUE  
           system  
           systemGroup  
           BY REFERENCE  
           serviceValue  
           RETURNING  
           intReturnValue.  
  
FmcjStrVAllocate.  
  
    CALL    "FmcjStringVectorAllocate"  
           USING  
           BY REFERENCE  
           hdlVector  
           RETURNING  
           intReturnValue.
```

Parameters

service Input/Output. The address of the handle to the object to be set when the object has been constructed. Ensure that the handle passed is not pointing to a still valid object, since that object is not automatically deallocated before the new object's handle is set.

initWithCurrentTime

Input. An indicator whether the date/time should be initialized with the current date/time.

system Input. The specific system where the execution server runs.

systemGroup

Input. The system group where the execution server resides. Specifying only the system group allows for exploiting the MQSeries clustering capabilities.

year/month/day

Input. The date part of the date/time.

hour/minute/second

Input. The time part of the date/time.

Return type

APIRET

The return code set by the allocation.

*Object**

The newly constructed object.

Assignment

In the C++ API, the assignment operator allows the application to assign the contents of the specified object to the target object, and returns the target object. The assignment is achieved by deleting the target object before the contents are assigned from the specified object.

C++

```
FmcjXxx & operator=( FmcjXxx const & anObject )
```

Parameters

anObject Input. The object from which the contents are to be assigned.

Comparison/equality

The following API calls allow an application to compare two transient objects in order to determine whether they represent the same persistent or API object.

Normally, comparison is done on the basis of the object identifiers. True is returned if both transient objects represent the same persistent object. The contents of the transient objects to be compared are not further checked, that is, it is not checked whether both transient objects carry the same states of the persistent object.

Exceptions:

- Service objects are equal when they represent the same session.
- Error objects are equal when they report the same error, that is, when they contain the same return code and the same parameters.
- Program data objects are equal when they belong to the same work item.
- Control connector instance objects are equal when they have the same source and target activity instances.
- Point and symbol layout objects are equal when their properties are equal.

In C and COBOL, the return code of the result object is set to *invalid handle* if one of the handles passed is invalid. True is returned if both are invalid, else false.

C

```
boo1 FMC_APIENTRY FmcjXxxEqual( FmcjXxxHandle handle1,
                                FmcjXxxHandle handle2 )
```

Programming concepts

C++

```
bool operator==( FmcjXxx const & anObject ) const
```

COBOL

```
FmcjXxxEqual.  
  CALL "FmcjXxxEqual"  
      USING  
      BY VALUE  
      handle1  
      handle2  
  RETURNING  
  boolReturnValue.
```

Parameters

anObject Input. The object to be compared with this one.

handle1 Input. The first object to be compared.

handle2 Input. The other object to be compared.

Copy

The following API calls allow the application to make a copy of a particular transient object. That copy becomes a separate object and thus carries its own state.

An exception is the execution service where a copy points to the same session established by the original object. This especially means, when you request to log off on either object, then the (common) session is closed.

C

```
APIRET FMC_APIENTRY FmcjXxxCopy( FmcjXxxHandle handle,  
                                 FmcjXxxHandle * newHandle )
```

C++

```
FmcjXxx( FmcjXxx const & anObject )
```

COBOL

```
FmcjXxxCopy.  
  CALL "FmcjXxxCopy"  
      USING  
      BY VALUE  
      handle  
      BY REFERENCE  
      newHandle  
  RETURNING  
  intReturnValue.
```

Parameters

anObject Input. The object to be copied.

handle Input. The handle of the object to be copied.

newHandle Input/Output. The address of a handle to be set when the object

has been constructed. Ensure that the handle passed is not pointing to a still valid object since that object is not automatically deallocated before the new object's handle is set.

Deallocation

The following API calls allow the application to delete the specified transient object. Deletion of a transient object has no impact on the represented persistent object, if any.

The C or COBOL handle is set to 0 so that it can no longer be used. The C++ destructor is automatically called when an instance of FmcjXxx is deleted.

C

```
APIRET FMC_APIENTRY FmcjXxxDeallocate( FmcjXxxHandle * handle )
```

C++

```
virtual FmcjXxx()
```

COBOL

```
FmcjXxxDeallocate.  
  
CALL "FmcjXxxDeallocate"  
    USING  
    BY REFERENCE  
    handle  
    RETURNING  
    intReturnValue.
```

Parameters

handle Input/Output. The address of the handle to the object to be deallocated.

IsComplete()

Returns true when the object has been completely read from an MQSeries Workflow server, that is, both primary and secondary properties are available (see also "Accessor API calls" on page 84).

C

```
bool FMC_APIENTRY FmcjXxxIsComplete( FmcjXxxHandle handle )
```

C++

```
bool IsComplete()
```

Java

```
public abstract boolean IsComplete() throws FmcException
```

Programming concepts

COBOL

```
FmcjXxxIsComplete.  
  
CALL    "FmcjXxxIsComplete"  
        USING  
        BY VALUE  
        handle  
        RETURNING  
        boolReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

bool/boolean

True if the object has been completely read from the server, otherwise false.

IsEmpty()

Returns whether the transient object contains no actual data values yet. The transient object has just been created and still contains default values. It does not yet reflect a persistent object.

C++

```
bool IsEmpty()
```

Java

```
public abstract boolean IsEmpty() throws FmcException
```

Return type

bool/boolean

True if the object has not yet been read from the server, otherwise false.

Kind()

Returns the kind of the queried object.

C

```
enum FmcjXxxEnum FMC_APIENTRY FmcjXxxKind( FmcjXxxHandle handle )
```

C++

```
FmcjXxx::Enum Kind() const
```

Java

```
public abstract Enum kind() throws FmcException
```


COBOL

```
FmcjXxxKind.  
  CALL      "FmcjXxxKind"  
           USING  
           BY VALUE  
           handle  
           RETURNING  
           intReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

FmcjXxxEnum/Enum

The kind of the object; some element of an enumeration - see also
"Accessing an enumerated value" on page 87.

Programming concepts

C example using basic functions

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
    APIRET rc;
    FmcjExecutionServiceHandle service = 0;
    FmcjWorkitemVectorHandle wList = 0;
    FmcjWorkitemHandle workitem1 = 0;
    FmcjWorkitemHandle workitem2 = 0;
    FmcjWorkitemHandle workitem3 = 0;

    FmcjGlobalConnect();

    /* logon */
    FmcjExecutionServiceAllocate(&service);
    rc = FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_Reset
                                   );

    /* Query Workitems */
    rc= FmcjExecutionServiceQueryWorkitems( service,
                                             FmcjNoFilter,
                                             FmcjNoSortCriteria,
                                             FmcjNoThreshold,
                                             &wList );

    printf( "\nQuery workitems returns rc : %u\n", rc );
    fflush(stdout);

    if ( rc == FMC_OK && FmcjWorkitemVectorSize(wList) >= 2 )
    {
        /* access first element */
        workitem1= FmcjWorkitemVectorFirstElement(wList);
        if ( FmcjWorkitemIsComplete(workitem1) )
            printf( "Surprise - more than primary data available\n" );
        else
            printf( "Primary data of first workitem available\n" );
        fflush(stdout);

        /* access next element */
        workitem2= FmcjWorkitemVectorNextElement(wList) ;
        if ( FmcjWorkitemEqual(workitem1,workitem2) )
            printf( "Surprise - workitems are equal\n" );
        else
            printf( "Workitems represent different objects\n" );
        fflush(stdout);

        /* copy workitem */
        FmcjWorkitemCopy(workitem1,&workitem3);
        if ( FmcjWorkitemEqual(workitem1,workitem3) )
            printf( "Workitems represent same persistent object\n" );
        else
            printf( "Surprise - workitems are not equal\n" );
        fflush(stdout);

        /* cleanup */
        FmcjWorkitemDeallocate(&workitem1);
        FmcjWorkitemDeallocate(&workitem2);
        FmcjWorkitemDeallocate(&workitem3);
    }
    FmcjWorkitemVectorDeallocate( &wList );
}
```

Figure 17. C example using basic functions (Part 1 of 2)

```

/* logoff */
FmcjExecutionServiceLogoff(service);
FmcjExecutionServiceDeallocate(&service);

FmcjGlobalDisconnect();
return FMC_OK;
}

```

Figure 17. C example using basic functions (Part 2 of 2)

C++ example using basic methods

```

#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
    FmcjGlobal::Connect();
    // logon
    FmcjExecutionService service;
    APIRET rc = service.Logon("USERID", "password");

    FmcjWorkitem workitem1;
    if ( workitem1.IsEmpty() )
        cout << "Transient workitem object has been created" << endl;
    else
        cout << "Surprise - workitem contains actual data" << endl;

    // Query Workitems
    vector<FmcjWorkitem> wList;
    rc= service.QueryWorkitems( FmcjNoFilter,
                                FmcjNoSortCriteria,
                                FmcjNoThreshold,
                                wList );
    cout << "Query workitems returns rc : " << rc << endl ;
    if ( rc == FMC_OK && wList.size() >= 2 )
    {
        workitem1= wList[0]; // assign first element
        if ( workitem1.IsComplete() )
            cout << "Surprise - more than primary data available" << endl;
        else
            cout << "Primary data of first workitem available" << endl;
        FmcjWorkitem workitem2= wList[1]; // access next element
        if ( workitem1 == workitem2 )
            cout << "Surprise - workitems are equal" << endl;
        else
            cout << "Workitems represent different objects" << endl;
        // copy workitem
        FmcjWorkitem workitem3(workitem1);
        if ( workitem1 == workitem3 )
            cout << "Workitems represent same persistent object" << endl;
        else
            cout << "Surprise - workitems are not equal" << endl;
    } // destructors called automatically

    // logoff
    rc = service.Logoff();

    FmcjGlobal::Disconnect();
    return FMC_OK;
} // destructors called automatically

```

Figure 18. C++ example using basic methods

Programming concepts

COBOL example using basic calls

Note: The SETADDR routine, which sets a pointer item to the address of a given string, is listed in "Example of the use of strings" on page 148.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "BASIC".

DATA DIVISION.

    WORKING-STORAGE SECTION.

        COPY fmcvars.
        COPY fmcconst.
        COPY fmcrcs.

        01 localUserID    PIC X(30) VALUE z"USERID".
        01 localPassword  PIC X(30) VALUE z"PASSWORD".
        01 workitem1     USAGE IS POINTER VALUE NULL.
        01 workitem2     USAGE IS POINTER VALUE NULL.
        01 workitem3     USAGE IS POINTER VALUE NULL.

LINKAGE SECTION.

        01 retCode       PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

        PERFORM FmcjGlobalConnect.
* logon
        PERFORM FmcjESAllocate.

        CALL "SETADDR" USING localUserId userId.
        CALL "SETADDR" USING localPassword passwordValue.
        MOVE Fmc-SM-Default TO sessionMode.
        MOVE Fmc-SA-Reset TO absenceIndicator.
        PERFORM FmcjESLogon.
* Query Workitems
        CALL "SETADDR" USING FmcjNoFilter filter.
        CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
        MOVE FmcjNoThreshold TO threshold.
        PERFORM FmcjESQueryWorkitems.
        SET hdlVector TO workitems.
        MOVE intReturnValue TO retCode.
        DISPLAY "Query Workitems returns rc : " retCode.

        IF retCode = FMC-OK
            PERFORM FmcjWIVSize
            IF ulongReturnValue >= 2
* access first element
                PERFORM FmcjWIVFirstElement
                SET workitem1 TO FmcjWIHandleReturnValue
                SET hdlItem TO workitem1
                PERFORM FmcjWIIsComplete
                IF boolReturnValue = 1
                    DISPLAY "Surprise - more than primary data"
                    DISPLAY "available"
                ELSE
                    DISPLAY "Primary data of first workitem"
                    DISPLAY "available"
            END-IF
```

Figure 19. COBOL example using basic calls (via PERFORM) (Part 1 of 2)

```

* access next element
    PERFORM FmcjWIVNextElement
    SET workitem2 TO FmcjWIHandleReturnValue
    SET hdlItem2 TO workitem2
    PERFORM FmcjWIEqual
    IF boolReturnValue = 1
        DISPLAY "Surprise - workitems are equal"
    ELSE
        DISPLAY "Workitems represent different objects"
    END-IF
* copy workitem
    SET hdlWorkitem TO workitem1
    PERFORM FmcjWICopy
    SET workitem3 TO newWorkItem
    SET hdlItem2 TO workitem3
    PERFORM FmcjWIEqual
    IF boolReturnValue = 0
        DISPLAY "Surprise - workitems are not equal"
    ELSE
        DISPLAY "Workitems represent same persistent"
        DISPLAY "objects"
    END-IF

* cleanup
    SET hdlWorkitem TO workitem1
    PERFORM FmcjWIDeallocate
    SET hdlWorkitem TO workitem2
    PERFORM FmcjWIDeallocate
    SET hdlWorkitem TO workitem3
    PERFORM FmcjWIDeallocate
    END-IF
END-IF

PERFORM FmcjWIVDeallocate.

* logoff
    PERFORM FmcjESLogoff.
    DISPLAY "Logged off".
    PERFORM FmcjESDeallocate.
    PERFORM FmcjGlobalDisconnect.
    MOVE FMC-OK TO retCode.
    GOBACK.

COPY fmcperf.

```

Figure 19. COBOL example using basic calls (via PERFORM) (Part 2 of 2)

Programming concepts

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "BASIC".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 workitem1    USAGE IS POINTER VALUE NULL.
01 workitem2    USAGE IS POINTER VALUE NULL.
01 workitem3    USAGE IS POINTER VALUE NULL.

LINKAGE SECTION.

01 retCode      PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

CALL "FmcjGlobalConnect".
CALL "FmcjExecutionServiceAllocate"
  USING BY REFERENCE serviceValue
  RETURNING intReturnValue.
* logon
CALL "SETADDR" USING localUserId userId.
CALL "SETADDR" USING localPassword passwordValue.
CALL "FmcjExecutionServiceLogon"
  USING BY VALUE serviceValue
  userID
  passwordValue
  Fmc-SM-Default
  Fmc-SA-Reset
  RETURNING intReturnValue.
* Query Workitems
CALL "SETADDR" USING FmcjNoFilter filter.
CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
CALL "FmcjExecutionServiceQueryWorkitems"
  USING BY VALUE serviceValue
  filter
  sortCriteria
  FmcjNoThreshold
  BY REFERENCE workitems
  RETURNING intReturnValue.
MOVE intReturnValue TO retCode.
DISPLAY "Query Workitems returns rc : " retCode.

IF retCode = FMC-OK
CALL "FmcjWorkitemVectorSize"
  USING BY VALUE workitems
  RETURNING ulongReturnValue

IF ulongReturnValue >= 2

* access first element
CALL "FmcjWorkitemVectorFirstElement"
  USING BY VALUE workitems
```

Figure 20. COBOL example using basic calls (via CALL) (Part 1 of 3)

```

        RETURNING FmcjWIHandleReturnValue
SET workitem1 TO FmcjWIHandleReturnValue
CALL "FmcjItemIsComplete"
    USING BY VALUE workitem1
    RETURNING boolReturnValue
IF boolReturnValue = 1
    DISPLAY "Surprise - more than primary data"
    DISPLAY "available"
ELSE
    DISPLAY "Primary data of first workitem"
    DISPLAY "available"
END-IF

* access next element
CALL "FmcjWorkitemVectorNextElement"
    USING BY VALUE workitems
    RETURNING FmcjWIHandleReturnValue
SET workitem2 TO FmcjWIHandleReturnValue
CALL "FmcjItemEqual"
    USING BY VALUE workitem1
        workitem2
    RETURNING boolReturnValue
IF boolReturnValue = 1
    DISPLAY "Surprise - workitems are equal"
ELSE
    DISPLAY "Workitems represent different objects"
END-IF

* copy workitem
CALL "FmcjWorkitemCopy"
    USING BY VALUE workitem1
        BY REFERENCE workitem3
    RETURNING intReturnValue
CALL "FmcjItemEqual"
    USING BY VALUE workitem1
        workitem3
    RETURNING boolReturnValue
IF boolReturnValue = 0
    DISPLAY "Surprise - workitems are not equal"
ELSE
    DISPLAY "Workitems represent same persistent"
    DISPLAY "objects"
END-IF

* cleanup
CALL "FmcjWorkitemDeallocate"
    USING BY REFERENCE workitem1
    RETURNING intReturnValue
CALL "FmcjWorkitemDeallocate"
    USING BY REFERENCE workitem2
    RETURNING intReturnValue
CALL "FmcjWorkitemDeallocate"
    USING BY REFERENCE workitem2
    RETURNING intReturnValue
END-IF
END-IF
CALL "FmcjWorkitemVectorDeallocate"
    USING BY REFERENCE workitems
    RETURNING intReturnValue.

```

Figure 20. COBOL example using basic calls (via CALL) (Part 2 of 3)

Programming concepts

```
* logoff
  CALL "FmcjExecutionServiceLogoff"
    USING BY VALUE serviceValue
    RETURNING intReturnValue.
  DISPLAY "Logged off".
  CALL "FmcjExecutionServiceDeallocate"
    USING BY REFERENCE serviceValue
    RETURNING intReturnValue.
  CALL "FmcjGlobalDisconnect".
  MOVE FMC-OK TO retCode.
  GOBACK.
```

Figure 20. COBOL example using basic calls (via CALL) (Part 3 of 3)

Accessor API calls

Accessor API calls are provided so that properties of transient objects can be read or changed. If the transient object represents a persistent one, then the values that are returned reflect the state of the persistent object when it was retrieved and used to set the transient object or when it was created or updated. Retrieval is done from an MQSeries Workflow server using the appropriate create, query, or refresh API calls. Creation or update can be done on the client when the MQSeries Workflow server sends new information (pushes information).

Default values are provided to you as long as the transient object is *empty* or *not complete*, or when the accessed property is *optional* and not set.

Default values are: an empty string or buffer for character-valued properties, 0 (zero) for integer-valued properties, false for boolean-valued properties, a timestamp with all members set to 0 (zero) for time-valued properties, "NotSet" for enumeration-valued properties, and an empty vector for multi-valued properties.

A transient object just constructed in C++ or Java is called *empty* because it does not yet reflect any persistent object. You can use the *IsEmpty()* method to determine whether the transient object still contains the default values only. Note that no action API call can be executed on an empty object.

By default, the MQSeries Workflow API provides for two views of persistent objects. They divide the persistent object into so-called *primary* properties and so-called *secondary* properties. Primary properties are considered "more important" from an access point of view. They are immediately returned when objects are queried. Secondary properties, and a refresh of the primary properties, are only returned on an explicit Refresh() request; on a per-object basis. You can use the *IsComplete()* API call to determine whether both primary and secondary object values have been read from the server.

Besides being primary or secondary, properties of a persistent object can be optional. This means that they can carry a value or not. When a default value is returned to you, you can use the corresponding *IsNull()* API call to determine whether that value is a value explicitly set or whether that value actually denotes that no value has been set. For example, when Threshold() returns 0 (zero), the threshold can have been set to zero, that is, no object is returned to you, or the threshold can have been set to have no value, that is, all qualifying objects are returned to you. Java is able to return null objects so that an *IsNull()* method is not needed.

Note that "null" is a concept orthogonal to being completely read. As long as the object is not complete, *IsNull()* will return true for a secondary, optional property

because nothing is known yet about the actual value and whether it has been set or not. For example, the documentation is a secondary and optional property of an object. When the object has been queried, then only the primary properties have been retrieved from the server. The Documentation() API call returns an empty string or buffer. To determine whether a documentation has been set at all, you can use the DocumentationIsNull() API call. The result will be “true” independent from the actual documentation setting as long as IsComplete() returns false. The documentation is assumed to be not set as long as the secondary data has not been retrieved.

Data values are accessible as long as the transient objects exist, regardless of the state of the persistent objects or of the current logon or logoff state. In general, you decide about the lifetime of your transient objects.

Because of the nature of transient objects, neither a connection to a server nor some specific authorization is required to access object properties or to update object properties of the transient object.

Return codes

Accessor API calls provide the value asked for as their return value. Default values are returned when an error occurred during the execution of the accessor API call. In C, C++, or COBOL, you can query the MQSeries Workflow result object for any errors encountered. Java throws an FmcException. The following codes can occur (the number in parentheses shows the integer value):

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is expected, but 0 is passed.

FMC_ERROR_BUFFER(800)

The buffer provided is too small to hold the largest possible value. See the appropriate header file or copybook for required lengths.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, default values are returned.

FMC_ERROR_DOES_NOT_EXIST(118)

The object does not exist. For example, the message was not found in the message catalog.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CONFIGURATION_ID(1022)

The configuration provided is invalid; it is 0 or does not conform to its syntax rules.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is invalid; it is 0 or is not pointing to an object of the requested type.

FMC_ERROR_INVALID_RESULT_HANDLE(814)

The handle of the result object provided is invalid; it is 0 or is not pointing to a result object.

FMC_ERROR_INVALID_TIME(802)

The time passed is invalid.

FMC_ERROR_MESSAGE_CATALOG(815)

The message catalog cannot be found..

FMC_ERROR_PROFILE(124)

The profile cannot be found or opened.

Programming concepts

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call is not called from within an activity implementation, for example, ProgramID(), or it is not valid from within an activity implementation, for example, SetConfiguration().

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call. For example, the activity implementation is not trusted and thus cannot receive its program ID.

FMC_ERROR_WRONG_STATE(120)

The API call cannot be executed because the object is in the wrong state. For example, the configuration cannot be changed after logon.

Accessor API calls allow for the operations listed below; **Xxx** denotes a particular class or scope, and "Property" denotes the property queried. For example, FmcjXxxProperty() can stand for FmcjItemDescription().

Accessing a value of type bool

Returns the value of a property of type *bool*. A default of *false* is returned if no information is available.

C

```
bool FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
bool Property() const
```

Java

```
public abstract boolean property() throws FmcException
```

COBOL

```
FmcjXxxProperty.  
  CALL "FmcjXxxProperty"  
      USING  
      BY VALUE  
      handle  
      RETURNING  
      boolReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

bool/ boolean The property value.

Declaration examples

```
C          bool FMC_APIENTRY FmcjWorkitemManualStartMode(  
          FmcjWorkitemHandle handle );
```

C++ `bool ManualStartMode() const;`

Java `public abstract boolean manualStartMode() throws FmcException;`

Accessing a value of type date/time

Returns the value of a date/time property. A zero timestamp is returned if no information is available.

C

```
FmcjCDateTime FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
FmcjDateTime Property() const
```

Java

```
public abstract Calendar property() throws FmcException
```

COBOL

```
FmcjXxxProperty.
  CALL      "FmcjXxxProperty"
           USING
           BY VALUE
           handle
           RETURNING
           dateTimeReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

time Input/Output. The date/time object to be set.

Return type

FmcjCDateTime/ FmcjDateTime/Calendar

The property value.

Declaration examples

C `FmcjCDateTime FMC_APIENTRY FmcjWorkitemEndTime(FmcjWorkitemHandle handle);`

C++ `FmcjDateTime EndTime() const;`

Java `public abstract Calendar endTime() throws FmcException;`

Accessing an enumerated value

Returns an enumerating value of a property. It is strongly advised to use the symbolic names in order to determine the actual value instead of the corresponding integer values. It is not guaranteed that integer values always stay the same. For COBOL programs, the symbolic values are defined in file `fmconst.cpy`.

"NotSet" or a similar indicator is returned if no information is available.

Programming concepts

```
C
enum FmcjXxxEnum FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

```
C++
FmcjXxx::Enum Property() const
```

```
Java
public abstract Enum property() throws FmcException
```

```
COBOL
FmcjXxxProperty.
  CALL "FmcXxxProperty"
      USING
      BY VALUE
      handle
      RETURNING
      intReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

FmcjXxxEnum/Enum

The property value, some element of an enumeration.

Declaration examples

```
C      FmcjItemAssignReason FMC_APIENTRY
      FmcjWorkitemReceivedAs( FmcjWorkitemHandle handle );
```

```
C++   FmcjItem::AssignReason ReceivedAs() const;
```

```
Java  public abstract AssignReason receivedAs() throws FmcException;
```

The following enumeration types and constants are defined. Numbers in parentheses are the corresponding integer values. You are strongly advised to use the symbolic names only.

- FmcjItemAssignReason/ FmcjItem::AssignReason/
com.ibm.workflow.api.ItemPackage.AssignReason

NotSet(0) Indicates that nothing is known about the assign reason.

C Fmc_IR_NotSet

C++ FmcjItem::NotSpecified

Java AssignReason.NOT_SPECIFIED

COBOL Fmc-IR-NotSet

Normal(1) Indicates that the work item or notification has been assigned to the user because the user qualified to receive the item.

| | | |
|--------------------------------|---|------------------------------------|
| | C | Fmc_IR_Normal |
| | C++ | FmcjItem::Normal |
| | Java | AssignReason.NORMAL |
| | COBOL | Fmc-IR-Normal |
| Substitute(2) | Indicates that the work item or notification has been assigned because the user is the substitute for the person who should have received the item. | |
| | C | Fmc_IR_Substitute |
| | C++ | FmcjItem::Substitute |
| | Java | AssignReason.Substitute |
| | COBOL | Fmc-IR-Substitute |
| ProcessAdministrator(3) | Indicates that the work item or notification has been assigned because the user is the process administrator. | |
| | C | Fmc_IR_ProcessAdministrator |
| | C++ | FmcjItem::ProcessAdministrator |
| | Java | AssignReason.PROCESS_ADMINISTRATOR |
| | COBOL | Fmc-IR-ProcAdministrator |
| SystemAdministrator(4) | Indicates that the work item or notification has been assigned because the user is the system administrator. | |
| | C | Fmc_IR_SystemAdministrator |
| | C++ | FmcjItem::SystemAdministrator |
| | Java | AssignReason.SYSTEM_ADMINISTRATOR |
| | COBOL | Fmc-IR-SystAdministrator |
| ByTransfer(5) | Indicates that the work item or notification has been transferred to the user. | |
| | C | Fmc_IR_ByTransfer |
| | C++ | FmcjItem::ByTransfer |
| | Java | AssignReason.BY_TRANSFER |
| | COBOL | Fmc-IR-ByTransfer |
| | • FmcjProcessTemplateAuditSetting/ FmcjProcessTemplate::AuditSetting/ com.ibm.workflow.api.ProcessTemplatePackage.AuditSetting | |
| NotSet(0) | Indicates that nothing is known about the audit setting. | |
| | C | Fmc_TA_NotSet |
| | C++ | FmcjProcessTemplate::NotSet |
| | Java | AuditSetting.NOT_SET |
| | COBOL | Fmc-TA-NotSet |
| NoAudit(1) | Indicates that auditing is not to be performed. | |
| | C | Fmc_TA_NoAudit |

Programming concepts

| | | |
|---------------------------|---|--|
| | C++ | FmcjProcessTemplate::NoAudit |
| | Java | AuditSetting.NO_AUDIT |
| | COBOL | Fmc-TA-NoAudit |
| Condensed(2) | Indicates that condensed auditing is to be performed. | |
| | C | Fmc_TA_Condensed |
| | C++ | FmcjProcessTemplate::Condensed |
| | Java | AuditSetting.CONDENSED |
| | COBOL | Fmc-TA-Condensed |
| Full(3) | Indicates that full auditing is to be performed. | |
| | C | Fmc_TA_Full |
| | C++ | FmcjProcessTemplate::Full |
| | Java | AuditSetting.FULL |
| | COBOL | Fmc-TA-Full |
| | <ul style="list-style-type: none"> • FmcjActivityInstanceEscalation/ FmcjActivityInstance::Escalation/ com.ibm.workflow.api.ActivityInstancePackage.Escalation | |
| NotSet(0) | Indicates that it is not known whether there is a notification on the activity instance. | |
| | C | Fmc_AE_NotSet |
| | C++ | FmcjActivityInstance::NotSpecified |
| | Java | Escalation.NOT_SPECIFIED |
| | COBOL | Fmc-AE-NotSet |
| NoNotification(1) | Indicates that no notification occurred so far on the activity instance. | |
| | C | Fmc_AE_NoNotification |
| | C++ | FmcjActivityInstance::NoNotification |
| | Java | Escalation.NO_NOTIFICATION |
| | COBOL | Fmc-AE-NoNotif |
| FirstNotification | Indicates that the first notification occurred. | |
| | C(4) | Fmc_AE_FirstNotification |
| | C++(4) | FmcjActivityInstance::FirstNotification |
| | Java(2) | Escalation.FIRST_NOTIFICATION |
| | COBOL(4) | Fmc-AE-FirstNotif |
| SecondNotification | Indicates that the second notification occurred. | |
| | C(5) | Fmc_AE_SecondNotification |
| | C++(5) | FmcjActivityInstance::SecondNotification |
| | Java(3) | Escalation.SECOND_NOTIFICATION |
| | COBOL(5) | Fmc-AE-SecNotif |

- FmcjActivityInstanceStateValue/ FmcjActivityInstance::state/
com.ibm.workflow.api.ActivityInstancePackage.ExecutionState

| | |
|-------------------|---|
| NotSet(0) | Indicates that nothing is known about the state of the activity instance. |
| C | Fmc_AS_NotSet |
| C++ | FmcjActivityInstance::undefined |
| Java | ExecutionState.UNDEFINED |
| COBOL | Fmc-AS-NotSet |
| Ready(1) | Indicates that the activity instance is in the ready state. |
| C | Fmc_AS_Ready |
| C++ | FmcjActivityInstance::ready |
| Java | ExecutionState.READY |
| COBOL | Fmc-AS-Ready |
| Running(2) | Indicates that the activity instance is in the running state. |
| C | Fmc_AS_Running |
| C++ | FmcjActivityInstance::running |
| Java | ExecutionState.RUNNING |
| COBOL | Fmc-AS-Running |
| Finished | Indicates that the activity instance is in the finished state. |
| C(4) | Fmc_AS_Finished |
| C++(4) | FmcjActivityInstance::finished |
| Java(3) | ExecutionState.FINISHED |
| COBOL(4) | Fmc-AS-Finished |
| Terminated | Indicates that the activity instance is in the terminated state. |
| C(8) | Fmc_AS_Terminated |
| C++(8) | FmcjActivityInstance::terminated |
| Java(4) | ExecutionState.TERMINATED |
| COBOL(8) | Fmc-AS-Term |
| Suspended | Indicates that the activity instance is in the suspended state. |
| C(16) | Fmc_AS_Suspended |
| C++(16) | FmcjActivityInstance::suspended |
| Java(5) | ExecutionState.SUSPENDED |
| COBOL(16) | Fmc-AS-Suspended |
| Inactive | Indicates that the activity instance is still inactive. |
| C(32) | Fmc_AS_Inactive |
| C++(32) | FmcjActivityInstance::inactive |
| Java(6) | ExecutionState.INACTIVE |
| COBOL(32) | Fmc-AS-Inactive |

Programming concepts

| | |
|----------------------|--|
| CheckedOut | Indicates that the activity instance has been checked out. |
| C(64) | Fmc_AS_CheckedOut |
| C++(64) | FmcjActivityInstance::checkedOut |
| Java(7) | ExecutionState.CHECKED_OUT |
| COBOL(64) | Fmc-AS-CheckedOut |
| InError | Indicates that the activity instance has not been executed successfully. |
| C(128) | Fmc_AS_InError |
| C++(128) | FmcjActivityInstance::inError |
| Java(8) | ExecutionState.IN_ERROR |
| COBOL(128) | Fmc-AS-InError |
| Executed | Indicates that the activity instance has been executed. |
| C(256) | Fmc_AS_Executed |
| C++(256) | FmcjActivityInstance::executed |
| Java(9) | ExecutionState.EXECUTED |
| COBOL(256) | Fmc-AS-Executed |
| Planning | Indicates that the activity instance is in the planning state. |
| C(512) | Fmc_AS_Planning |
| C++(512) | FmcjActivityInstance::planning |
| Java(10) | ExecutionState.PLANNING |
| COBOL(512) | Fmc-AS-Planning |
| ForceFinished | Indicates that the activity instance is in the force-finished state. |
| C(1024) | Fmc_AS_ForceFinished |
| C++(1024) | FmcjActivityInstance::forceFinished |
| Java(11) | ExecutionState.FORCE_FINISHED |
| COBOL(1024) | Fmc-AS-Force-Finished |
| Skipped | Indicates that the activity instance has not been executed but skipped. |
| C(2048) | Fmc_AS_Skipped |
| C++(2048) | FmcjActivityInstance::skipped |
| Java(12) | ExecutionState.SKIPPED |
| COBOL(2048) | Fmc-AS-Skipped |
| Deleted | Indicates that the activity instance has been deleted. |
| C(4096) | Fmc_AS_Deleted |
| C++(4096) | FmcjActivityInstance::deleted |
| Java(13) | ExecutionState.DELETED |
| COBOL(4096) | Fmc-AS-Deleted |
| Terminating | Indicates that the activity instance is in the terminating state. |

| | | |
|-------------------|--|--|
| | C(8192) | Fmc_AS_Terminating |
| | C++(8192) | FmcjActivityInstance::terminating |
| | Java(14) | ExecutionState.TERMINATING |
| | COBOL(8192) | Fmc-AS-Terminating |
| Suspending | | Indicates that the activity instance is in the suspending state. |
| | C(16384) | Fmc_AS_Suspending |
| | C++(16384) | FmcjActivityInstance::suspending |
| | Java(15) | ExecutionState.SUSPENDING |
| | COBOL(16384) | Fmc-AS-Suspending |
| | • FmcjActivityInstanceType/ FmcjActivityInstance::Type/ com.ibm.workflow.api.ActivityInstancePackage.Type | |
| NotSet(0) | | Indicates that nothing is known about the type of the activity instance. |
| | C | Fmc_AT_NotSet |
| | C++ | FmcjActivityInstance::NotSet |
| | Java | Type.NOT_SET |
| | COBOL | Fmc-AT-NotSet |
| Process(1) | | Indicates that the activity instance is implemented by a process. |
| | C | Fmc_AT_Process |
| | C++ | FmcjActivityInstance::Process |
| | Java | Type.PROCESS |
| | COBOL | Fmc-AT-Proc |
| Program(2) | | Indicates that the activity instance is implemented by a program. |
| | C | Fmc_AT_Program |
| | C++ | FmcjActivityInstance::Program |
| | Java | Type.PROGRAM |
| | COBOL | Fmc-AT-Program |
| Block | | Indicates that the activity instance is implemented by a block. |
| | C(16) | Fmc_AT_Block |
| | C++(16) | FmcjActivityInstance::Block |
| | Java(3) | Type.BLOCK |
| | COBOL(16) | Fmc-AT-Block |
| | • FmcjControlConnectorInstanceStateValue/ FmcjControlConnectorInstance::state/ com.ibm.workflow.api.ControlConnectorInstancePackage.EvaluationState | |
| False(0) | | Indicates that evaluation of the control connector resulted in False. |
| | C | Fmc_CS_False |
| | C++ | FmcjControlConnectorInstance::False |

Programming concepts

| | | |
|------------------------|---|--|
| | Java | EvaluationState.IS_FALSE |
| | COBOL | Fmc-CS-False |
| True(1) | Indicates that evaluation of the control connector resulted in True. | |
| | C | Fmc_CS_True |
| | C++ | FmcjControlConnectorInstance::True |
| | Java | EvaluationState.IS_TRUE |
| | COBOL | Fmc-CS-True |
| NotEvaluated(2) | Indicates that the control connector has not yet been evaluated. | |
| | C | Fmc_CS_NotEvaluated |
| | C++ | FmcjControlConnectorInstance::NotEvaluated |
| | Java | EvaluationState.NOT_EVALUATED |
| | COBOL | Fmc-CS-NotEvaluated |
| NotSet(3) | Indicates that nothing is known about the evaluation of the control connector. | |
| | C | Fmc_CS_NotSet |
| | C++ | FmcjControlConnectorInstance::NotSet |
| | Java | EvaluationState.NOT_SET |
| | COBOL | Fmc-CS-NotSet |
| | <ul style="list-style-type: none"> • FmcjControlConnectorInstanceType/ FmcjControlConnectorInstance::Type/ com.ibm.workflow.api.ControlConnectorInstancePackage.Type | |
| NotSet(0) | Indicates that nothing is known about the type of the control connector instance. | |
| | C | Fmc_CT_NotSet |
| | C++ | FmcjControlConnectorInstance::Undefined |
| | Java | Type.UNDEFINED |
| | COBOL | Fmc-CT-NotSet |
| Condition(1) | Indicates that the control connector instance is a connector which can have a transition condition. | |
| | C | Fmc_CT_Condition |
| | C++ | FmcjControlConnectorInstance::Condition |
| | Java | Type.CONDITION |
| | COBOL | Fmc-CT-Condition |
| Otherwise(2) | Indicates that the control connector instance is the “otherwise” connector. | |
| | C | Fmc_CT_Otherwise |
| | C++ | FmcjControlConnectorInstance::Otherwise |
| | Java | Type.OTHERWISE |
| | COBOL | Fmc-CT-Otherwise |

- FmcjProgramTemplateExeMode/ FmcjProgramTemplate::ExeMode

| | |
|------------------|---|
| NotSet(0) | Indicates that nothing is known about the execution mode. |
| C | Fmc_GM_NotSet |
| C++ | FmcjProgramTemplate::NotSet |
| COBOL | Fmc-GM-NotSet |
| Normal(1) | Indicates that the program does not participate in global transactions. |
| C | Fmc_GM_Normal |
| C++ | FmcjProgramTemplate::Normal |
| COBOL | Fmc-GM-Normal |
| Safe(2) | Indicates that the program participates in global transactions. |
| C | Fmc_GM_Safe |
| C++ | FmcjProgramTemplate::Safe |
| COBOL | Fmc-GM-Safe |
- FmcjExeOptionsStyle/ FmcjExeOptions::Style/
com.ibm.workflow.api.ProgramDataPackage.Style

| | |
|---------------------|---|
| NotSet(0) | Indicates that nothing is known about the style of the EXE. |
| C | Fmc_EO_NotSet |
| C++ | FmcjExeOptions::NotSet |
| Java | Style.NOT_SET |
| COBOL | Fmc-EO-NotSet |
| Visible(1) | Indicates that the EXE should start visibly. |
| C | Fmc_EO_Visible |
| C++ | FmcjExeOptions::Visible |
| Java | Style.VISIBLE |
| COBOL | Fmc-EO-Visible |
| Invisible(2) | Indicates that the EXE should start invisibly. |
| C | Fmc_EO_Invisible |
| C++ | FmcjExeOptions::Invisible |
| Java | Style.INVISIBLE |
| COBOL | Fmc-EO-Invisible |
| Minimized(3) | Indicates that the EXE should start minimized. |
| C | Fmc_EO_Minimized |
| C++ | FmcjExeOptions::Minimized |
| Java | Style.MINIMIZED |
| COBOL | Fmc-EO-Minimized |
| Maximized(4) | Indicates that the EXE should start maximized. |
| C | Fmc_EO_Maximized |

Programming concepts

- | | | |
|--|--------------|---------------------------|
| | C++ | FmcjExeOptions::Maximized |
| | Java | Style.MAXIMIZED |
| | COBOL | Fmc-EO-Maximized |
- FmcjProgramTemplateExeUser/ FmcjProgramTemplate::ExeUser
- | | | |
|------------------|---|-----------------------------------|
| NotSet(0) | Indicates that nothing is known about the execution user. | |
| | C | Fmc_GU_NotSet |
| | C++ | FmcjProgramTemplate::NotSpecified |
| | COBOL | Fmc-GU-NotSet |
- | | | |
|-----------------|---|----------------------------|
| Agent(1) | Indicates that the program executes under the identifier of the program execution server. | |
| | C | Fmc_GU_Agent |
| | C++ | FmcjProgramTemplate::Agent |
| | COBOL | Fmc-GU-Agent |
- | | | |
|-------------------|--|------------------------------|
| Starter(2) | Indicates that the program executes under the user ID of the starter of the program. | |
| | C | Fmc_GU_Starter |
| | C++ | FmcjProgramTemplate::Starter |
| | COBOL | Fmc-GU-Starter |
- FmcjExternalOptionsTimePeriod/ FmcjExternalOptions::TimePeriod/
com.ibm.workflow.api.ProgramDataPackage.TimePeriod
- | | | |
|------------------|--|-----------------------------|
| NotSet(0) | Indicates that nothing is known about an external service timeout. | |
| | C | Fmc_EX_NotSet |
| | C++ | FmcjExternalOptions::NotSet |
| | Java | TimePeriod.NOT_SET |
| | COBOL | Fmc-EX-NotSet |
- | | | |
|------------------------|---|-----------------------------------|
| TimeInterval(1) | Indicates that the program execution server should wait a specified time interval for the answer of the started external service. | |
| | C | Fmc_EX_TimeInterval |
| | C++ | FmcjExternalOptions::TimeInterval |
| | Java | TimePeriod.TIME_INTERVAL |
| | COBOL | Fmc-EX-TimeInterval |
- | | | |
|-------------------|--|------------------------------|
| Forever(2) | Indicates that the program execution server should wait forever for the answer of the started external service, that is, whatever time it takes. | |
| | C | Fmc_EX_Forever |
| | C++ | FmcjExternalOptions::Forever |
| | Java | TimePeriod.FOREVER |
| | COBOL | Fmc-EX-Forever |

| | |
|--|--|
| Never(3) | Indicates that the program execution server should not wait for an answer of the started external service. |
| C | Fmc_EX_Never |
| C++ | FmcjExternalOptions::Never |
| Java | TimePeriod.NEVER |
| COBOL | Fmc-EX-Never |
| • FmcjExecutionDataKindEnum/ FmcjExecutionData::KindEnum | |
| NotSet(0) | Indicates that nothing is known about the type of the execution data. |
| C | Fmc_DART_NotSet |
| C++ | FmcjExecutionData::NotSet |
| Java | not supported |
| COBOL | Fmc-DART-NotSet |
| Error(1) | Indicates that execution of an asynchronous call returns an error. |
| C | Fmc_DART_Error |
| C++ | FmcjExecutionData::Error |
| Java | not supported |
| COBOL | Fmc-DART-Error |
| Terminate(2) | Indicates that receiving execution data can end. |
| C | Fmc_DART_Terminate |
| C++ | FmcjExecutionData::Terminate |
| Java | not supported |
| COBOL | Fmc-DART-Terminate |
| ItemDeleted(1000) | Indicates that the execution data describes the deletion of a work item or notification. |
| C | Fmc_DART_ItemDeleted |
| C++ | FmcjExecutionData::ItemDeleted |
| Java | not supported |
| COBOL | Fmc-DART-ItemDeleted |
| Workitem(1002) | Indicates that the execution data describes the creation or update of a work item. |
| C | Fmc_DART_Workitem |
| C++ | FmcjExecutionData::Workitem |
| Java | not supported |
| COBOL | Fmc-DART-Workitem |
| ActivityInstanceNotification(1003) | Indicates that the execution data describes the creation or update of an activity instance notification. |

Programming concepts

| | |
|--------------|---|
| C | Fmc_DART_ActivityInstanceNotification |
| C++ | FmcjExecutionData::ActivityInstanceNotification |
| Java | not supported |
| COBOL | Fmc-DART-ActInstNotif |

ProcessInstanceNotification(1004)

Indicates that the execution data describes the creation or update of a process instance notification.

| | |
|--------------|--|
| C | Fmc_DART_ProcessInstanceNotification |
| C++ | FmcjExecutionData::ProcessInstanceNotification |
| Java | not supported |
| COBOL | Fmc-DART-ProcInstNotif |

ExecuteInstanceResponse(1100)

Indicates that the execution data describes the answer to an asynchronous request which asked for the creation and execution of a process instance.

| | |
|--------------|--|
| C | Fmc_DART_ProcessInstanceNotification |
| C++ | FmcjExecutionData::ProcessInstanceNotification |
| Java | not supported |
| COBOL | Fmc-DART-ExecuteInstResponse |

ExecuteProgramResponse(1101)

Indicates that the execution data describes the answer to an asynchronous request which asked for the execution of a program.

| | |
|--------------|---|
| C | Fmc_DART_ExecuteProgramResponse |
| C++ | FmcjExecutionData::ExecuteProgramResponse |
| Java | not supported |
| COBOL | Fmc-DART-ExecuteProgResponse |

- FmcjImplementationDataBasis/ FmcjImplementationData::Basis/
com.ibm.workflow.api.ProgramDataPackage.Basis

NotSet(0) Indicates that nothing is known about the operating system platform of the implementing program.

| | |
|--------------|--------------------------------------|
| C | Fmc_DP_NotSet |
| C++ | FmcjImplementationData::NotSpecified |
| Java | Basis.NOT_SET |
| COBOL | Fmc-DP-NotSet |

OS2(1) Indicates that the program is an OS/2 program.

| | |
|--------------|-----------------------------|
| C | Fmc_DP_OS2 |
| C++ | FmcjImplementationData::OS2 |
| Java | Basis.OS2 |
| COBOL | Fmc-DP-OS2 |

AIX(2) Indicates that the program is an AIX program.

| | | |
|---------------------|---|-----------------------------------|
| | C | Fmc_DP_AIX |
| | C++ | FmcjImplementationData::AIX |
| | Java | Basis.AIX |
| | COBOL | Fmc-DP-AIX |
| HPUX(3) | Indicates that the program is an HP-UX program. | |
| | C | Fmc_DP_HPUX |
| | C++ | FmcjImplementationData::HPUX |
| | Java | Basis.HPUX |
| | COBOL | Fmc-DP-HPUX |
| Windows95(4) | Indicates that the program is a Windows 95 program. | |
| | C | Fmc_DP_Windows95 |
| | C++ | FmcjImplementationData::Windows95 |
| | Java | Basis.WINDOWS_95 |
| | COBOL | Fmc-DP-Windows95 |
| WindowsNT(5) | Indicates that the program is a Windows NT program. | |
| | C | Fmc_DP_WindowsNT |
| | C++ | FmcjImplementationData::WindowsNT |
| | Java | Basis.WINDOWS_NT |
| | COBOL | Fmc-DP-WindowsNT |
| OS/390(6) | Indicates that the program is an OS/390 program. | |
| | C | Fmc_DP_OS390 |
| | C++ | FmcjImplementationData::OS390 |
| | Java | Basis.OS390 |
| | COBOL | Fmc-DP-OS390 |
| Solaris(7) | Indicates that the program is a Solaris program. | |
| | C | Fmc_DP_Solaris |
| | C++ | FmcjImplementationData::Solaris |
| | Java | Basis.SOLARIS |
| | COBOL | Fmc-DP-SOLARIS |
| • | FmcjImplementationDataType/ FmcjImplementationData::Type/ com.ibm.workflow.api.ProgramDataPackage.Type | |
| NotSet(0) | Indicates that nothing is known about the implementation. | |
| | C | Fmc_DT_NotSet |
| | C++ | FmcjImplementationData::NotSet |
| | Java | ImplementationData.NOT_SET |
| | COBOL | Fmc-DT-NotSet |
| EXE(1) | Indicates that the program is an executable. | |

Programming concepts

| | | |
|--|--|---|
| | C | Fmc_DT_EXE |
| | C++ | FmcjImplementationData::EXE |
| | Java | ImplementationData.EXE |
| | COBOL | Fmc-DT-EXE |
| DLL(2) | Indicates that the program is implemented by a dynamic link library. | |
| | C | Fmc_DT_DLL |
| | C++ | FmcjImplementationData::DLL |
| | Java | ImplementationData.DLL |
| | COBOL | Fmc-DT-DLL |
| External | Indicates that the program is some external service. | |
| | C(4) | Fmc_DT_External |
| | C++(4) | FmcjImplementationData::External |
| | Java(3) | ImplementationData.EXTERNAL |
| | COBOL(4) | Fmc-DT-External |
| | • FmcjItemType/ FmcjItem::ItemType/ com.ibm.workflow.api.ItemPackage.ItemType | |
| NotSet(0) | Indicates that nothing is known about the item type. | |
| | C | Fmc_IT_NotSet |
| | C++ | FmcjItem::unknown |
| | Java | ItemType.UNKNOWN |
| | COBOL | Fmc-IT-NotSet |
| Workitem(1) | Indicates that the item is a work item. | |
| | C | Fmc_IT_Workitem |
| | C++ | FmcjItem::Workitem |
| | Java | ItemType.WORK_ITEM |
| | COBOL | Fmc-IT-Workitem |
| ProcessInstanceNotification | Indicates that the item is a process instance notification. | |
| | C(3) | Fmc_IT_ProcessInstanceNotification |
| | C++(3) | FmcjItem::ProcessInstanceNotification |
| | Java(2) | ItemType.PROCESS_INSTANCE_NOTIFICATION |
| | COBOL(3) | Fmc-IT-ProInstNotif |
| FirstActivityInstanceNotification | Indicates that the item is the first activity instance notification. | |
| | C(4) | Fmc_IT_FirstActivityInstanceNotification |
| | C++(4) | FmcjItem::FirstActivityInstanceNotification |
| | Java(3) | ItemType.FIRST_ACTIVITY_INSTANCE_NOTIFICATION |

COBOL(4) Fmc-IT-FirstActInstNotif

SecondActivityInstanceNotification

Indicates that the item is the second activity instance notification.

C(5) Fmc_IT_SecondActivityInstanceNotification

C++(5) FmcjItem::SecondActivityInstanceNotification

Java(4) ItemType.SECOND_ACTIVITY_INSTANCE_NOTIFICATION

COBOL(5) Fmc-IT-SecActInstNotif

- FmcjProcessInstanceEscalation/ FmcjProcessInstance::Escalation/
com.ibm.workflow.api.ProcessInstancePackage.Escalation

NotSet(0) Indicates that it is not known whether there is a notification on the process instance.

C Fmc_PE_NotSet

C++ FmcjProcessInstance::NotSet

Java Escalation.NOT_SET

COBOL Fmc-PE-NotSet

NoNotification(1)

Indicates that no notification occurred so far on the process instance.

C Fmc_PE_NoNotification

C++ FmcjProcessInstance::NoNotification

Java Escalation.NO_NOTIFICATION

COBOL Fmc-PE-NoNotif

ProcessInstanceNotification

Indicates that a process instance notification occurred.

C(3) Fmc_PE_ProcessNotification

C++(3) FmcjProcessInstance::ProcessNotification

Java(2) Escalation.PROCESS_NOTIFICATION

COBOL(3) Fmc-PE-ProcNotif

- FmcjProcessInstanceStateValue/ FmcjProcessInstance::state/
com.ibm.workflow.api.ProcessInstancePackage.ExecutionState

NotSet(0) Indicates that nothing is known about the state of the process instance.

C Fmc_PS_NotSet

C++ FmcjProcessInstance::undefined

Java ExecutionState.UNDEFINED

COBOL Fmc-PS-NotSet

Ready(1) Indicates that the process instance is in the ready state.

C Fmc_PS_Ready

C++ FmcjProcessInstance::ready

Java ExecutionState.READY

Programming concepts

| | | |
|--------------------|-------------------|--|
| | COBOL | Fmc-PS-Ready |
| Running(2) | | Indicates that the process instance is in the running state. |
| | C | Fmc_PS_Running |
| | C++ | FmcjProcessInstance::running |
| | Java | ExecutionState.RUNNING |
| | COBOL | Fmc-PS-Running |
| Finished | | Indicates that the process instance is in the finished state. |
| | C(4) | Fmc_PS_Finished |
| | C++(4) | FmcjProcessInstance::finished |
| | Java(3) | ExecutionState.FINISHED |
| | COBOL(4) | Fmc-PS-Finished |
| Terminated | | Indicates that the process instance is in the terminated state. |
| | C(8) | Fmc_PS_Terminated |
| | C++(8) | FmcjProcessInstance::terminated |
| | Java(4) | ExecutionState.TERMINATED |
| | COBOL(8) | Fmc-PS-Term |
| Suspended | | Indicates that the process instance is in the suspended state. |
| | C(16) | Fmc_PS_Suspended |
| | C++(16) | FmcjProcessInstance::suspended |
| | Java(5) | ExecutionState.SUSPENDED |
| | COBOL(16) | Fmc-PS-Suspended |
| Terminating | | Indicates that the process instance is in the terminating state. |
| | C(32) | Fmc_PS_Terminating |
| | C++(32) | FmcjProcessInstance::terminating |
| | Java(6) | ExecutionState.TERMINATING |
| | COBOL(32) | Fmc-PS-Terminating |
| Suspending | | Indicates that the process instance is in the suspending state. |
| | C(64) | Fmc_PS_Suspending |
| | C++(64) | FmcjProcessInstance::suspending |
| | Java(7) | ExecutionState.SUSPENDING |
| | COBOL(64) | Fmc-PS-Suspending |
| Deleted | | Indicates that the process instance is in the deleted state. |
| | C(128) | Fmc_PS_Deleted |
| | C++(128) | FmcjProcessInstance::deleted |
| | Java(8) | ExecutionState.DELETED |
| | COBOL(128) | Fmc-PS-Deleted |

- State/ FmcjItemStateValue/ FmcjItem::state/
com.ibm.workflow.api.ItemPackage.ExecutionState

| | |
|-------------------|--|
| NotSet(0) | Indicates that nothing is known about the state of the item. |
| C | Fmc_IS_NotSet |
| C++ | FmcjItem::undefined |
| Java | ExecutionState.UNDEFINED |
| COBOL | Fmc-IS-NotSet |
| Ready(1) | Indicates that the item is in the ready state. |
| C | Fmc_IS_Ready |
| C++ | FmcjItem::ready |
| Java | ExecutionState.READY |
| COBOL | Fmc-IS-Ready |
| Running(2) | Indicates that the item is in the running state. |
| C | Fmc_IS_Running |
| C++ | FmcjItem::running |
| Java | ExecutionState.RUNNING |
| COBOL | Fmc-IS-Running |
| Finished | Indicates that the item is in the finished state. |
| C(4) | Fmc_IS_Finished |
| C++(4) | FmcjItem::finished |
| Java(3) | ExecutionState.FINISHED |
| COBOL(4) | Fmc-IS-Finished |
| Terminated | Indicates that the item is in the terminated state. |
| C(8) | Fmc_IS_Terminated |
| C++(8) | FmcjItem::terminated |
| Java(4) | ExecutionState.TERMINATED |
| COBOL(8) | Fmc-IS-Term |
| Suspended | Indicates that the item is in the suspended state. |
| C(16) | Fmc_IS_Suspended |
| C++(16) | FmcjItem::suspended |
| Java(5) | ExecutionState.SUSPENDED |
| COBOL(16) | Fmc-IS-Suspended |
| Disabled | Indicates that the item is disabled. |
| C(32) | Fmc_IS_Disabled |
| C++(32) | FmcjItem::disabled |
| Java(6) | ExecutionState.DISABLED |
| COBOL(32) | Fmc-IS-Disabled |
| CheckedOut | Indicates that the item is checked out. |
| C(64) | Fmc_IS_CheckedOut |

Programming concepts

| | | |
|----------------------|--------------------|--|
| | C++(64) | FmcjItem::checkedOut |
| | Java(7) | ExecutionState.CHECKED_OUT |
| | COBOL(64) | Fmc-IS-CheckedOut |
| InError | | Indicates that the item is in the InError state. |
| | C(128) | Fmc_IS_InError |
| | C++(128) | FmcjItem::inError |
| | Java(8) | ExecutionState.IN_ERROR |
| | COBOL(128) | Fmc-IS-InError |
| Executed | | Indicates that the item has been executed. |
| | C(256) | Fmc_IS_Executed |
| | C++(256) | FmcjItem::Executed |
| | Java(9) | ExecutionState.EXECUTED |
| | COBOL(256) | Fmc-IS-Executed |
| Planning | | Indicates that the item is in the planning state. |
| | C(512) | Fmc_IS_Planning |
| | C++(512) | FmcjItem::Planning |
| | Java(10) | ExecutionState.PLANNING |
| | COBOL(512) | Fmc-IS-Planning |
| ForceFinished | | Indicates that the item has been force-finished. |
| | C(1024) | Fmc_IS_ForceFinished |
| | C++(1024) | FmcjItem::ForceFinished |
| | Java(11) | ExecutionState.FORCE_FINISHED |
| | COBOL(1024) | Fmc-IS-ForceFinished |
| Deleted | | Indicates that the item has been deleted. |
| | C(4096) | Fmc_IS_Deleted |
| | C++(4096) | FmcjItem::Deleted |
| | Java(12) | ExecutionState.DELETED |
| | COBOL(4096) | Fmc-IS-Deleted |
| Terminating | | Indicates that the item is in the terminating state. |
| | C(8192) | Fmc_IS_Terminating |
| | C++(8192) | FmcjItem::Terminating |
| | Java(13) | ExecutionState.TERMINATING |
| | COBOL(8192) | Fmc-IS-Terminating |
| Suspending | | Indicates that the item is in the suspending state. |
| | C(16384) | Fmc_IS_Suspending |
| | C++(16384) | FmcjItem::Suspending |
| | Java(14) | ExecutionState.SUSPENDING |

COBOL(16384)

Fmc-IS-Suspending

- FmcjWorkitemProgramRetrieval/ FmcjWorkitem::ProgramRetrieval/
com.ibm.workflow.api.WorkItemPackage.ProgramRetrieval

NotSet(0) Indicates that nothing is said about which program definitions to retrieve.

| | |
|--------------|--------------------------|
| C | Fmc_WS_NotSet |
| C++ | FmcjWorkitem::NotSet |
| Java | ProgramRetrieval.NOT_SET |
| COBOL | Fmc-WS-NotSet |

CommonDataOnly(1)

Indicates that the common parts of program definitions are to be retrieved.

| | |
|--------------|-----------------------------------|
| C | Fmc_WS_CommonDataOnly |
| C++ | FmcjWorkitem::CommonDataOnly |
| Java | ProgramRetrieval.COMMON_DATA_ONLY |
| COBOL | Fmc-WS-CommonDataOnly |

SpecifiedDefinitions(2)

Indicates that the specified program definitions are to be retrieved.

| | |
|--------------|--|
| C | Fmc_WS_SpecifiedDefinitions |
| C++ | FmcjWorkitem::SpecifiedDefinitions |
| Java | ProgramRetrieval.SPECIFIED_DEFINITIONS |
| COBOL | Fmc-WS-SpecifiedDefs |

AllDefinitions

Indicates that all program definitions are to be retrieved.

| | |
|-----------------|----------------------------------|
| C(4) | Fmc_WS_AllDefinitions |
| C++(4) | FmcjWorkitem::AllDefinitions |
| Java(3) | ProgramRetrieval.ALL_DEFINITIONS |
| COBOL(4) | Fmc-WS-AllDefs |

- FmcjPersistentListOfList/ FmcjPersistentList::TypeOfList/
com.ibm.workflow.api.PersistentListPackage.TypeOfList

NotSet(0) Indicates that nothing is known about the list type.

| | |
|--------------|----------------------------|
| C | Fmc_LT_NotSet |
| C++ | FmcjPersistentList::NotSet |
| Java | TypeOfList.NOT_SET |
| COBOL | Fmc-LT-NotSet |

Public(1) Indicates that the list definition is for public usage.

| | |
|-------------|----------------------------|
| C | Fmc_LT_Public |
| C++ | FmcjPersistentList::Public |
| Java | TypeOfList.PUBLIC |

Programming concepts

| | | |
|----------------|-----------------|--|
| | COBOL | Fmc-LT-Public |
| Private | | Indicates that the list definition is for private usage. |
| | C(3) | Fmc_LT_Private |
| | C++(3) | FmcjPersistentList::Private |
| | Java(2) | TypeOfList.PRIVATE |
| | COBOL(3) | Fmc-LT-Private |

Accessing a value of type integer

Returns the value of a property of type *long*, *unsigned long*, or *int*. Zero (0) is returned if no information is available. The following examples illustrate return of a *long* value.

C

```
long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
unsigned long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
long Property() const
unsigned long Property() const
```

Java

```
public abstract int property() throws FmcException
```

COBOL

```
FmcjXxxProperty.
  CALL "FmcjXxxProperty"
      USING
      BY VALUE
      handle
  RETURNING
      longReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

long/unsigned long/int

The property value.

Declaration examples

```
C unsigned long FMC_APIENTRY FmcjWorkitemPriority(
    FmcjWorkitemHandle handle );
C++ unsigned long Priority() const;
Java public abstract int priority() throws FmcException;
```

Accessing a value of type string

Returns the value of a property of type *string*. An empty string or buffer is returned if no information is available.

```
C
char * FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle,
                                     char *         buffer,
                                     unsigned long  bufferLength )
```

```
C++
string Property() const
```

```
Java
public abstract String property() throws FmcException
```

```
COBOL
FmcjXxxProperty.
  CALL "FmcjXxxProperty"
    USING
      BY VALUE
        handle
        buffer
        bufferLength
    RETURNING
      pointerReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.
buffer Input/Output. A pointer to a buffer to contain the property value.
bufferLength Input. The length of the buffer; must be big enough to hold the largest possible value (see file `fmcmxcon.h` for the minimum required lengths). You can use a single buffer for retrieving all your character values.

Return type

char*/string/String
 The property value.

Declaration examples

```
C      char* FMC_APIENTRY FmcjWorkitemDescription(
      FmcjWorkitemHandle handle );

C++   string Description() const;

Java  public abstract String Description() throws FmcException;
```

Accessing a multi-valued property

Returns the value of a multi-valued property by providing a collection of values. The collection is represented as a vector in C, C++, and COBOL, and as an array in

Programming concepts

Java. In C++, the collection object to be filled has to be provided by the caller. Use the appropriate accessor API calls to read a single value (refer to “C and COBOL vector accessor functions” on page 21).

An unchanged vector or an empty array is returned if no information is available.

Any existing array elements are overwritten. Vector elements in C++ are, however, appended to the supplied vector. If you want to read the current values only, you have to erase all elements of the vector.

C

```
FmcjYyyVectorHandle  
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
void Property( vector<ValueType> & value ) const
```

Java

```
public abstract ValueType[] property() throws FmcException
```

COBOL

```
FmcjXxxProperty.  
CALL "FmcjXxxProperty"  
USING  
BY VALUE  
handle  
RETURNING  
FmcjYyyVectorHandleReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

value Input/Output. The vector or array to contain the values of the property.

Return type

FmcjValueTypeVectorHandle/ValueType[]

The vector or array of values of the property.

Declaration examples

```
C FmcjStringVectorHandle FMC_APIENTRY FmcjWorkitemStaff(  
FmcjWorkitemHandle handle );
```

```
C++ void Staff( vector<string> & staff ) const;
```

```
Java public abstract String[] staff() throws FmcException;
```

Accessing an object valued property

Returns the value of a property which is itself described by an object.

C

```
FmcjObjectHandle
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
FmcjObject Property() const
```

Java

```
public abstract Object property() throws FmcException

public abstract ExecutionService
locate( String systemGroup, String system) throws FmcException

public abstract
ExecutionAgent getExecutionAgent() throws FmcException
```

COBOL

```
FmcjXxxProperty.
    CALL      "FmcjXxxProperty"
            USING
            BY VALUE
            handle
    RETURNING
            FmcjObjectHandleReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

system Input. The system where the execution server runs.

systemGroup

Input. The system group where the execution server runs.

Return type

ExecutionAgent

The program execution agent which provides for the context of an activity implementation.

ExecutionService

The execution service which provides for the interface to the execution server.

Object/Handle/FmcjObject

The property value.

Declaration examples

C FmcjErrorHandle FMC_APIENTRY FmcjWorkitemErrorReason(
FmcjWorkitemHandle handle);

C++ FmcjError ErrorReason() const;

Java public abstract FmcError errorReason() throws FmcException;

Programming concepts

Accessing a pointer valued property

Returns the value of a property which is a pointer to some object.

C

```
FmcjObjectHandle  
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++

```
FmcjObject * Property() const
```

Java

```
public abstract Object property() throws FmcException
```

COBOL

```
FmcjXxxProperty.  
CALL "FmcjXxxProperty"  
USING  
BY VALUE  
handle  
RETURNING  
FmcjObjectHandleReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

Object*/Handle/FmcjObject*

A pointer or handle to the object or the object itself.

Declaration examples

```
C          FmcjReadOnlyContainerHandle FMC_APIENTRY  
          FmcjProgramDataInContainer( FmcjProgramDataHandle handle );
```

```
C++       FmcjReadOnlyContainer* InContainer() const;
```

```
Java      public abstract ReadOnlyContainer inContainer() throws  
          FmcException;
```

Determining whether an optional property is set

This API call states whether an optional property is set.

When the property is a secondary property and the object queried is not yet completely read, it is unknown whether the property is set or not so that a default value of true is returned.

Note: Java does not expose IsNull() methods since it is able to return null objects.

C

```
bool FMC_APIENTRY FmcjXxxPropertyIsNull( FmcjXxxHandle handle )
```

C++

```
bool PropertyIsNull() const
```

COBOL

```
FmcjXxxPropertyIsNull.  
  CALL "FmcjXxxPropertyIsNull"  
      USING  
      BY VALUE  
      handle  
      RETURNING  
      boolReturnValue.
```

Parameters

handle Input. The handle of the object to be queried.

Return type

bool/boolean True if the property is not set, otherwise false.

Declaration examples

C `bool FMC_APIENTRY FmcjWorkitemDescriptionIsNull(FmcjWorkitemHandle handle);`

C++ `bool DescriptionIsNull() const;`

Setting a value of type integer

This API call sets the specified property to the specified value.

C

```
void FMC_APIENTRY FmcjXxxSetProperty( FmcjXxxHandle handle,  
                                       long          newValue );
```

C++

```
void SetProperty( long newValue );
```

Java

```
public abstract void setProperty( int newValue ) throws FmcException
```

Programming concepts

COBOL

```
FmcjXxxSetProperty.  
  CALL    "FmcjXxxSetProperty"  
        USING  
        BY VALUE  
        handle  
        newValue.
```

Parameters

handle Input. The handle of the object to be queried.
newValue Input. The new value of the property.

Declaration examples

C void FMC_APIENTRY FmcjExecutionServiceSetTimeout(
 FmcjExecutionServiceHandle handle, long newValue);

C++ void SetTimeout(long newValue) const;

Java public abstract void SetTimeout(int newValue) throws
 FmcException;

An example is the `FmcjService::SetTimeout` API call which sets the timeout value for requests issued by the client to an MQSeries Workflow server via this `FmcjService` object. In other words, it sets the time the client is willing to wait for an answer.

When set, the new timeout value is used for all API calls requiring communication between the client and the server. It can be set (changed) as often as desired. It is to be provided as milliseconds. A negative value is interpreted as -1, that is, an indefinite timeout.

The default timeout value is taken from the user's profile, from the `APITimeOut` value; if not found, from the configuration profile. If it is also not found there, the default is 180000 ms (3 minutes).

Note: It is possible that, even though `FMC_ERROR_TIMEOUT` is returned when you issue a client-server call, the MQSeries Workflow server has successfully processed the request. However, the server could not send back `FMC_OK` because communication reported a timeout in the meantime. If the request has not been processed, increase the value set for the timeout and retry the call.

Setting an object valued property

This API call sets the specified property for the specified object.

Java

```
public abstract void addProperty( Object value )  
  
public abstract  
void setContext( String args[], Properties properties )  
  
public abstract  
void setContext( Applet applet, Properties properties )
```

Parameters

applet Input. The applet which instantiated the agent. If IIOP is used as the communication protocol, this information is required.

args Input. The command line arguments passed to the application which instantiated the agent bean.

properties Input. The environmental properties passed to the application or applet when it was instantiated.

value Input. The value of the property.

Declaration examples

Java public abstract void addPropertyChangeListener(
PropertyChangeListener value);

Updating an object

This API call updates the specified object with information sent from an MQSeries Workflow server. The update information must have been provided for the specified object.

The server pushes update information for work items (as long as they are not disabled), activity instance notifications, and process instance notifications. The process setting of the associated process instance must specify REFRESH_POLICY PUSH for that process instance itself or as a process default. Logon must have been performed with a present session mode.

```

C
APIRET FMC_APIENTRY FmcjXxxUpdate( FmcjXxxHandle            handle,
                                         FmcjExecutionDataHandle data );
    
```

```

C++
APIRET Update( FmcjExecutionData const & data );
    
```

```

COBOL
FmcjXxxUpdate.
CALL "FmcjXxxUpdate"
     USING
     BY VALUE
         handle
         dataValue
     RETURNING
         intReturnValue.
    
```

Parameters

handle Input. The handle of the object to be updated.

data/dataValue Input. The data which is to be used for the update.

Return codes

The C and COBOL calls and the MQSeries Workflow result object can return the following codes; the number in parentheses shows their integer value:
FMC_OK(0) The API call completed successfully.

Programming concepts

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is expected, but 0 is passed.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, it does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is invalid; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_OID(805)

The execution data is no data to update the specified object; it does not belong to the specified object.

FMC_ERROR_WRONG_KIND(501)

The execution data is no data to update the specified object; it is no update data.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

C example: accessing values

```

#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
    APIRET rc;
    FmcjExecutionServiceHandle service = 0;
    FmcjWorkitemHandle workitem = 0;
    FmcjStringVectorHandle sList = 0;
    char category[FMC_CATEGORY_NAME_LENGTH+1];
    char generalBuffer[200];
    unsigned long priority = 0;
    int enumValue = 0;
    FmcjCDateTime startTime;
    unsigned long i = 0;

    FmcjGlobalConnect();

    /* logon */
    FmcjExecutionServiceAllocate(&service);
    rc = FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_Reset
                                   );

                                   /* set the timeout for requests */
    FmcjExecutionServiceSetTimeout( service, 60000 );
    /* assumption: workitem has been queried from the server */
                                   /* access a value of type bool */
    if ( FmcjWorkitemCategoryIsNull( workitem ) )
        printf( "Category is not set\n" );
    else
                                   /* access a value of type char */
                                   /* use a buffer which fits */
        FmcjWorkitemCategory( workitem, category, FMC_CATEGORY_NAME_LENGTH+1 );
        printf( "Category : %s\n", category );
    }

                                   /* access a date/time value */
    startTime= FmcjWorkitemStartTime( workitem );
    printf( "Start time : %s\n",
           FmcjDateTimeAsString(&startTime, generalBuffer, 200) );

                                   /* access a value of type long */
    priority = FmcjWorkitemPriority( workitem );
    printf( "Priority : %u\n", priority );

                                   /* access an enumerated value */
    enumValue= FmcjWorkitemReceivedAs( workitem );
    if ( enumValue == Fmc_IR_Normal )
        printf( "Received as: %s\n", "qualified user" );
    ...

                                   /* access a multi-valued field */
    sList= FmcjWorkitemSupportTools( workitem );
    printf( "Support tools: " );
    for( i=0; i< FmcjStringVectorSize(sList); i++ )
    {
                                   /* use a large buffer */
        printf("%s ", FmcjStringVectorNextElement(sList, generalBuffer, 200) );
    }

    /* logoff */
    FmcjExecutionServiceLogoff(service);
    FmcjExecutionServiceDeallocate(&service);
    FmcjGlobalDisconnect();
    return FMC_OK;
}

```

Figure 21. Accessing values in C

Programming concepts

C++ example: accessing values

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
    FmcjGlobal::Connect();
    // logon
    FmcjExecutionService service;  APIRET rc = service.Logon("USERID", "password");
                                   // set the timeout for requests
    service.SetTimeout( 60000 );
    // assumption: workitem has been queried from the server
                                   // access a value of type bool
    if ( workitem.CategoryIsNull() )
        cout << "Category is not set" << endl;
    else
        // access a value of type char
        {
            // use a buffer which fits
            cout << "Category   : " << workitem.Category() << endl;
        }
                                   // access a value of type date/time
    cout << "Start time : " << workitem.StartTime() << endl;
                                   // access a value of type long
    cout << "Priority   : " << workitem.Priority() << endl;
                                   // access an enumerated value
    FmcjItem::AssignReason reason= workitem.ReceivedAs();
    cout << "Received as: " <<
        ((reason == FmcjItem::Normal) ? "normal user" : "...")
        << endl;
    vector<string> tools; int j;      // access a multi-valued field
    workitem.SupportTools( tools );
    cout << "Support tools: " ;
    while ( j < tools.size() )
        cout << tools[j++] << " ";
    // logoff
    rc = service.Logoff();
    FmcjGlobal::Disconnect();
    return FMC_OK;
}                                     // destructors called automatically
```

Figure 22. Accessing values in C++

COBOL example: accessing values

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VALUES".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 categBuffer   PIC X(34).
01 generalBuffer PIC X(200).
01 i             PIC 9(9) BINARY VALUE 0.

LINKAGE SECTION.

01 retCode          PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.
* logon
    PERFORM FmcjESAllocate.

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.
* set the timeout for requests
    MOVE 60000 TO newTimeOutValue.
    PERFORM FmcjESSetTimeout.

* assumption: workitem has been queried from the server
* and hdlItem points to this workitem

* access a value of type bool (PIC 9 BINARY)
    PERFORM FmcjWICategIsNull.
    IF boolReturnValue = 1
        DISPLAY "Category is not set"
    ELSE
* access a value of type char (POINTER to a PIC X(n))
* use a buffer which fits
    CALL "SETADDR" USING categBuffer categoryNameBuffer
    MOVE FMC-CATEG-NAME-LENGTH TO bufferLength
    PERFORM FmcjWICateg
    DISPLAY "Category   : " categBuffer
    END-IF
* access a date/time value
    PERFORM FmcjWIStartTime.
    MOVE dateTimeReturnValue TO timeValue.
    CALL "SETADDR" USING generalBuffer dateTimeBuffer.
    MOVE 200 TO bufferLength.
    PERFORM FmcjDateTimeAsString.
    DISPLAY "Start time : " generalBuffer.
* access a value of type unsigned long (PIC 9(9) BINARY)
    SET hdlWorkitem TO hdlItem.
    PERFORM FmcjWIPriority.
    DISPLAY "Priority   : " ulongReturnValue.

```

Figure 23. Accessing values in COBOL (via PERFORM) (Part 1 of 2)

Programming concepts

```
* access an enumerated value (PIC S9(9) BINARY)
  PERFORM FmcjWIReceivedAs.
  IF intReturnValue = Fmc-IR-Normal
    DISPLAY "Received as: qualified user"
  END-IF
* access a multi-valued field
  PERFORM FmcjWISupportTools.
  SET hdlVector TO FmcjStrVHandleReturnValue.
  PERFORM FmcjStrVSize.
  DISPLAY "Support tools: ".
* use a large buffer
  CALL "SETADDR" USING generalBuffer elementBuffer
  PERFORM VARYING i FROM 0 BY 1 UNTIL i >= ulongReturnValue
    PERFORM FmcjStrVNextElement
    DISPLAY generalBuffer
  END-PERFORM
* logoff
  PERFORM FmcjESLogoff.
  DISPLAY "Logged off".
  PERFORM FmcjESDeallocate.
  PERFORM FmcjGlobalDisconnect.
  MOVE FMC-OK TO retCode.
  GOBACK.

COPY fmcperf.
```

Figure 23. Accessing values in COBOL (via PERFORM) (Part 2 of 2)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VALUES".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 categBuffer   PIC X(34).
01 generalBuffer PIC X(200).
01 i             PIC 9(9) BINARY VALUE 0.

LINKAGE SECTION.

01 retCode          PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    CALL "FmcjGlobalConnect".
* logon
    CALL "FmcjExecutionServiceAllocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    CALL "FmcjExecutionServiceLogon"
        USING BY VALUE serviceValue
                    userID
                    passwordValue
                    Fmc-SM-Default
                    Fmc-SA-Reset
        RETURNING intReturnValue.

* set the timeout for requests
    MOVE 60000 TO newTimeOutValue.
    CALL "FmcjServiceSetTimeout"
        USING BY VALUE serviceValue
                    newTimeOutValue.

* assumption: workitem has been queried from the server
* and hdlItem points to this workitem

* access a value of type bool (PIC 9 BINARY)
    CALL "FmcjItemCategoryIsNull"
        USING BY VALUE hdlItem
        RETURNING boolReturnValue.
    IF boolReturnValue = 1
        DISPLAY "Category is not set"
    ELSE

* access a value of type char (POINTER to a PIC X(n))
* use a buffer which fits
    CALL "SETADDR" USING categBuffer categoryNameBuffer
    MOVE FMC-CATEG-NAME-LENGTH TO bufferLength
    CALL "FmcjItemCategory"

```

Figure 24. Accessing values in COBOL (via CALL) (Part 1 of 3)

Programming concepts

```
        USING BY VALUE hdItem
                categoryNameBuffer
                bufferLength
        RETURNING pointerReturnValue
        DISPLAY "Category : " categBuffer
    END-IF

* access a date/time value
    CALL "FmcjItemStartTime"
        USING BY VALUE hdItem
        RETURNING dateTimeReturnValue.
    CALL "SETADDR" USING generalBuffer dateTimeBuffer.
    MOVE 200 TO bufferLength.
    CALL "FmcjDateTimeAsString"
        USING BY REFERENCE dateTimeReturnValue
        BY VALUE          dateTimeBuffer
                bufferLength
        RETURNING pointerReturnValue.
    DISPLAY "Start time : " generalBuffer.
* access a value of type unsigned long (PIC 9(9) BINARY)
    SET hdWorkitem TO hdItem.
    CALL "FmcjWorkitemPriority"
        USING BY VALUE hdItem
        RETURNING ulongReturnValue.
    DISPLAY "Priority : " ulongReturnValue.
* access an enumerated value (PIC S9(9) BINARY)
    CALL "FmcjItemReceivedAs"
        USING BY VALUE hdItem
        RETURNING intReturnValue.
    IF intReturnValue = Fmc-IR-Normal
        DISPLAY "Received as: qualified user"
    END-IF
* access a multi-valued field
    CALL "FmcjWorkitemSupportTools"
        USING BY VALUE hdItem
        RETURNING FmcjStrVHandleReturnValue.
    SET hdVector TO FmcjStrVHandleReturnValue.
    CALL "FmcjStringVectorSize"
        USING BY VALUE hdVector
        RETURNING ulongReturnValue.
    DISPLAY "Support tools: ".
* use a large buffer
    CALL "SETADDR" USING generalBuffer elementBuffer
    PERFORM VARYING i FROM 0 BY 1 UNTIL i >= ulongReturnValue
    CALL "FmcjStringVectorNextElement"
        USING BY VALUE hdVector
                elementBuffer
                bufferLength
        RETURNING pointerReturnValue
    DISPLAY generalBuffer
    END-PERFORM
* logoff
    CALL "FmcjExecutionServiceLogoff"
        USING BY VALUE serviceValue
        RETURNING intReturnValue.
    DISPLAY "Logged off".
    CALL "FmcjExecutionServiceDeallocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.
    CALL "FmcjGlobalDisconnect".
```

Figure 24. Accessing values in COBOL (via CALL) (Part 2 of 3)

```
MOVE FMC-OK TO retCode.
GOBACK.
```

Figure 24. Accessing values in COBOL (via CALL) (Part 3 of 3)

Action API calls

Action API calls are client-server calls involving communication with an MQSeries Workflow server. As such, they require the client to be logged on.

Action API calls can be issued on service objects and on transient objects representing persistent ones. These objects remember the context of a user session so that a communication path to an MQSeries Workflow server can be established. As a consequence, empty objects cannot be used in order to issue action calls.

Action API calls are either synchronous requests waiting for the server's reply, asynchronous requests expecting the server's reply at some other point in time, or API calls receiving information from an MQSeries Workflow server.

All action API calls are described separately in "Chapter 5. API action and activity implementation calls" on page 273. You can find examples in "Chapter 6. Examples" on page 511.

Activity implementation API calls

An activity can be implemented by a program which uses the MQSeries Workflow API. In this case, the activity implementation API calls provide access to the input and output containers of the activity instance or work item. They also allow the program implementing an activity to return the updated output container to MQSeries Workflow so that navigation can continue on the basis of those values.

A program implementing an activity is usually executed under the control of an MQSeries Workflow program execution server on request from an MQSeries Workflow execution server. When an MQSeries Workflow execution server receives a request to start a work item, it determines the implementing program to be started and sends an appropriate request together with the input and output containers, if needed, to the program execution server or the logged-on user's program execution agent. Since containers are sent to the program execution server, input and output containers are requested from and passed to an MQSeries Workflow program execution server by the implementing program. You do *not* have to create an execution service object and log on to an MQSeries Workflow execution server to handle containers from within an activity implementation.

However, if you want not only to access containers but also to query information about the process instance the work item is a part of, you have to log on to the MQSeries Workflow execution server that requested that your program be started. You can use the Passthrough() API call of the execution service to begin a session with the execution server from within the activity implementation program or support tool. In this way, you can use the environment of the work item, that is, you do not need any other user ID, password, system group, or system information.

If the activity implementation does not handle all work by itself but distributes work by starting subprograms that run as separate operating system processes, and when those subprograms request containers, then the program execution server cannot know which is the calling program. For that purpose, the program calling the program execution server must provide the program identification of the actual

Programming concepts

activity implementation, that is, it must use the *remote* container or passthrough calls. This requires that the activity implementation has retrieved its program identification and passed it to the started program. Note that the program execution server only provides the program identification to *trusted* programs.

Chapter 2. Language interfaces

This chapter discusses language-specific considerations for using the API. It also describes the XML message interface in detail.

C and C++ interface

This section provides an overview of the concepts which are specific to the MQSeries Workflow C and C++ APIs.

Coding an MQSeries Workflow client application in C or C++

An MQSeries Workflow C or C++ client application typically contains the following parts (which may not be delimited this clearly, however):

| | |
|---------|---|
| Setup | <pre>#include <MQ Workflow API prerequisites (C++)> #include <MQ Workflow API> int main() { Declare objects : Connect Allocate service object Logon() }</pre> |
| Actions | <pre>MQSeries Workflow API calls</pre> |
| Cleanup | <pre>Logoff() Deallocate service object Disconnect return 0; }</pre> |

To set up your program, you typically declare the program variables or objects you are going to use and you include the MQSeries Workflow API header files you need for your actions. When using the C++ API, definitions of *bool*, *string*, and *vector* are needed. Include the respective files before the MQSeries Workflow API headers.

You should then initialize the MQSeries Workflow API by calling `Connect()` so that resources held by the API are allocated correctly. `Connect()` and `Disconnect()` are to be called at the beginning and end of each thread, respectively.

You then need to allocate a service object which represents the server you are going to ask services from. Once the service object is allocated, you can log on. `Logon` establishes a session between the user logging on and the server represented by your service object. All subsequent calls requiring client/server communication run through this session.

C and C++ interface

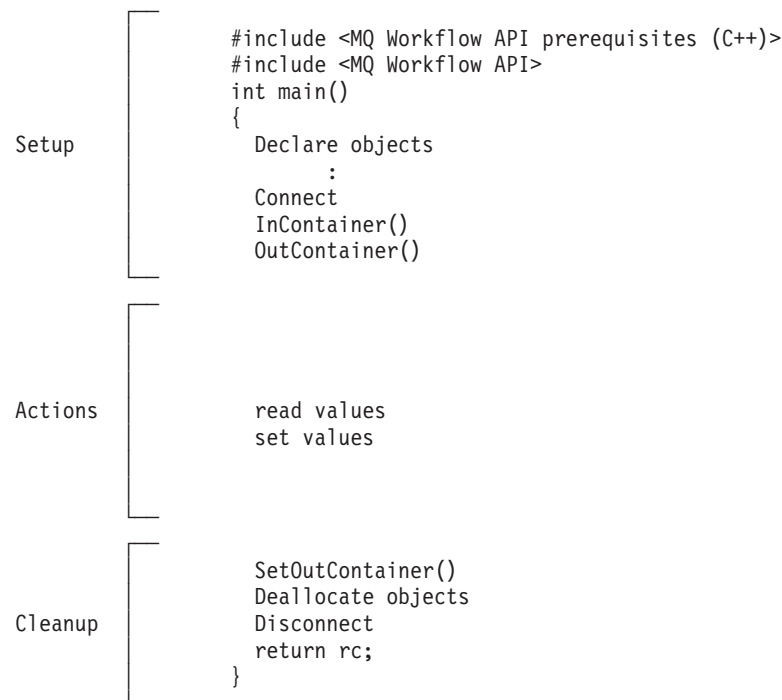
After a successful logon, you can issue action or program execution management API calls in order to query or manage MQSeries Workflow objects you are authorized for.

At the end of your program, you log off in order to close the session to the server and you deallocate any resources held by your program, especially the service object.

As a last step, you disconnect from the MQSeries Workflow API so that resources held by the API are deallocated correctly.

Coding an MQSeries Workflow activity implementation in C or C++

An MQSeries Workflow C or C++ activity implementation typically contains the following parts:



To set up your program, you typically declare the program variables or objects you are going to use and you include the MQSeries Workflow API header files you need for your actions. When using the C++ API, definitions of *bool*, *string*, and *vector* are needed. Include the respective files before the MQSeries Workflow API headers.

You should then initialize the MQSeries Workflow API by calling `Connect()` so that resources held by the API are allocated correctly. `Connect()` and `Disconnect()` are to be called at the beginning and end of each thread, respectively.

On platforms other than OS/390, an activity implementation can then retrieve the activity's input and output containers from the MQSeries Workflow program execution agent that started this program. On OS/390, the activity implementation in CICS or IMS gets input and output containers when it is started by the PES.

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQSeries Workflow program execution server. Any resources held by your program are deallocated. The return value of your program tells the program execution server about the overall outcome of your program.

The output container as well as the return code of your program are passed back to the MQSeries Workflow server which requested the execution of the activity implementation. The return code (`_RC`) can be used in exit or transition conditions in order to guide MQSeries Workflow navigation.³

As a last step, you disconnect from the MQSeries Workflow API so that resources held by the API are deallocated correctly.

Your activity implementation can also behave like a client application and request services from an MQSeries Workflow server, normally the server from where its execution had been triggered. The `Passthrough()` API call is then used instead of the `Logon()` API call in order to log on to the server which caused the program execution with the user identification and authority known to the server from the work item start request.

Compiling and linking

C and C++ programs for MQSeries Workflow for OS/390 must be compiled with IBM C/C++ for OS/390 Version 2 Release 4 or later.

All C and C++ programs developed for use with MQSeries Workflow must include header files provided by MQSeries Workflow and link the corresponding library files.

When using the MQ Workflow C++ API, definitions for `bool`, `string`, and `vector` must be provided. MQ Workflow delivers some files to be included: `bool.h`, which provides for the `bool` definition and must be included first, `fmcjstr.hxx`, which provides for the `string` definition, and `vector.h`, which provides for the `vector` definition.

Note that `bool.h` and `vector.h` are part of the Standard Template Library delivered with MQ Workflow and copyrighted by the Hewlett-Packard Company. Documentation of this library is provided on the MQ Workflow CD-ROM (non-390 version) in a file named `STLDOC.PS`. It is installed in the `stl` subdirectory of the API.

The MQSeries Workflow features you use determine which header files to include and the compilers you use which library files to link with. Depending on the feature used, the following header files must be included:

| Feature | C header | C++ header |
|---|-------------------------|----------------------------|
| Runtime client | <code>fmcjcrun.h</code> | <code>fmcjprun.hxx</code> |
| Runtime activity implementation: - container access only (Container API) | <code>fmcjcon.h</code> | <code>fmcjpccon.hxx</code> |

³ For compilers which do not support an exit code of an application, it is possible to set the `_RC` data member of the output container.

C and C++ interface

| Feature | C header | C++ header |
|--|------------|--------------|
| - container and server access (Full API) | fmcjcrun.h | fmcjprun.hxx |

Table 3. JCLs provided for C/C++ programs

| Job | Sample |
|--|----------|
| Native OS/390 C/C++ Full API compile job | FMCHJ1CF |
| Native OS/390 C/C++ API run job | FMCHJ1CR |
| CICS C/C++ Full API compile job | FMCHJ2CF |
| CICS C/C++ Container API compile job | FMCHJ2CC |
| IMS C/C++ Container API compile job | FMCHJ3CC |

For more information about CICS/IMS specifics like stubs or precompiler, refer to the documentation for these components.

The compilers given as prerequisites or newer versions can be used to compile and link your applications accessing the MQSeries Workflow API. Your compile and link options must ensure that the MQSeries Workflow API is called with the calling convention that is defined in the FMC_APIENTRY macro (see file fmcjcglo.h). FMC_APIENTRY has been defined to the standard C calling convention and will be automatically be applied when you use the header files provided by MQSeries Workflow.

Access can be gained to C functions using calls from all languages that support C calls.

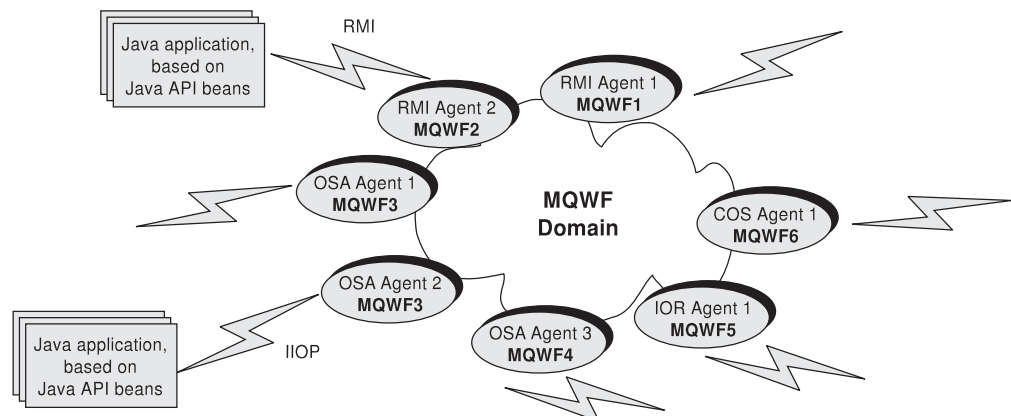
Java interface

Note: The following section describes the general Java environment supported by the LAN version of MQSeries Workflow. Only selected components of this environment, namely the local bindings and the Java agent, are currently implemented in the OS/390 version.

The MQSeries Workflow Java API consists of:

- An agent that connects an MQSeries Workflow domain to the Java world.
- A set of API Beans that provide MQSeries Workflow API functionality to Java based applications.

The Java CORBA Agent



In order to support *thin clients*, a Java agent approach has been chosen. The Java CORBA Agent serves as a proxy for the MQSeries Workflow domain.

The Java CORBA Agent is implemented in Java and wraps the MQSeries Workflow C++ API into a form that is accessible from the Java environment. On the one hand, the Java CORBA Agent thus wraps the native product API and on the other hand it publishes a Java form of the API on the network.

The communication layer

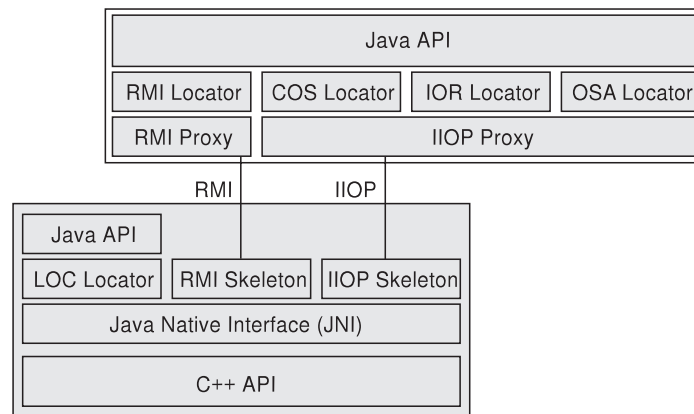
The Java CORBA Agent runs on an MQSeries Workflow machine and Java clients run somewhere on the network. MQSeries Workflow supports CORBA, RMI, and Local environments so that clients can access the agent.

- CORBA is the Object Management Group (OMG) standard for distributed computing. It is very easy to publish existing objects on a network using ORBs. The currently supported ORB is Inprise's VisiBroker Java 3.3.
- Java Remote Method Invocation (RMI) is an approach that is completely Java based and does not require additional software. RMI is included in most Java environments.

Note: Java RMI agents should only be used for prototyping. They are currently not suited for production purposes.

- Local bindings offer a special mechanism which imbeds the Java CORBA Agent. They bypass the communication layer and use procedure calls. If client applications use local bindings, then they have to consider the trade-off that they become MQSeries clients. It follows that local bindings are best suited for agent side applications, for example, *servlets* and Java-based non-GUI activity implementations.

The locator methods



In theory, there are a number of methods available to clients for finding their agent. These different methods are listed below.

Note: The only method currently supported by OS/390 is LOC naming.

- OSA naming: a VisiBroker specific naming facility (Smart Agent) that is very easy to use. It only requires one OSAgent running on the subnetwork that keeps track of all the objects and their name in the network. As smart agents synchronize their information via UDP, the only thing that has to be known is the name of the object the client program is looking for.

Java interface

- IOR naming: Via InterOrbReferences a vendor-independent naming service for CORBA applications exists. The identity of a specific object (its IOR) is published in a file on a Web server. This file can be accessed from clients via a published URL to obtain the actual reference of an object.
- COS naming: CORBA Naming Service is the native CORBA directory service. Objects can use COS to publish their identity to the CORBA system.
- RMI registry: The RMI registry comes with every Java development kit. It can be run as a stand-alone program where object implementations register or it can be embedded into the application. Embedding has the big advantage that no separate program is necessary to provide naming functions. To locate objects via the RMI registry, the host which runs the RMI registry has to be known.

Note: Java RMI agents should only be used for prototyping. They are currently not suited for production purposes.

- LOC naming: This approach can be used to connect the Java API to an MQSeries Workflow C++ API that is located on the same physical machine. This approach can be useful if a client is to be written on a platform that offers the API but does not offer a native client, for example, on AIX. It can also be used to access the MQSeries Workflow API from a Web server through servlet technology without the additional communication overhead, because local bindings use procedure calls.

Note: This method is the only method currently supported on OS/390.

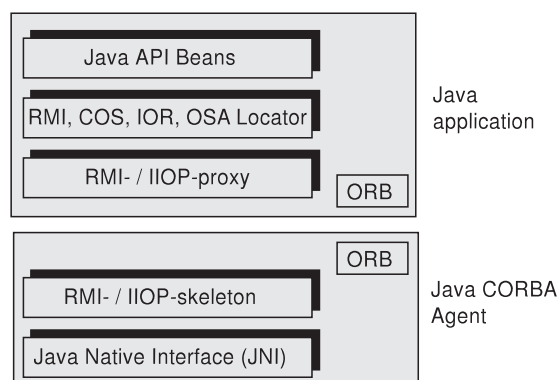
The Java API Beans

Both the client-side communication layer and the API Beans layer are implemented in Java. This makes it possible to run applications developed with the MQSeries Workflow Java API on any machine that provides a Java Virtual Machine (JVM).

The API Beans provide functionality equivalent to the other MQSeries Workflow APIs. Due to the introduction of an agent, its name, context, and locator policy must be specified, however.

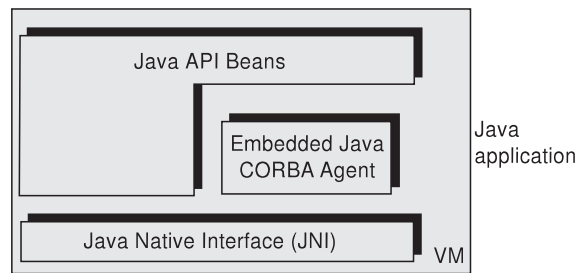
Following are some usage scenarios of the Java API.

Java in the intranet



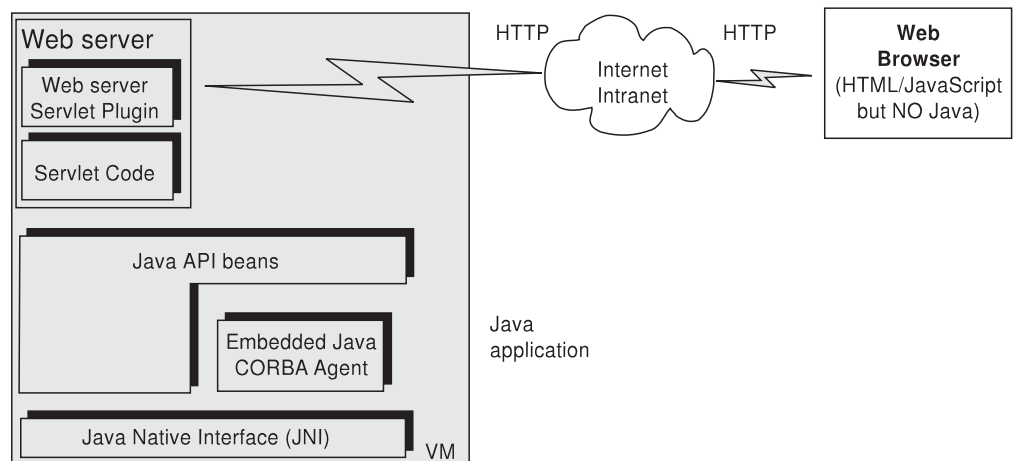
In this case, a non-LOC locator policy must be used and an external agent must be specified. The API Beans and, for a non-RMI protocol, the VisiBroker ORB (COS, IOR, OSA) must have been installed.

Java as a programming language



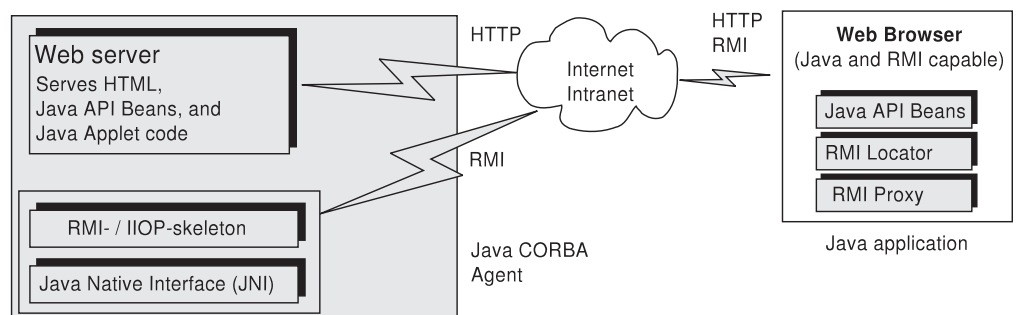
In this case, the LOC_LOCATOR policy of the Java CORBA Agent is used. The Java API Beans must have been installed.

Java in the Internet (servlet)



In this case, a LOC_LOCATOR policy must be used. The API Beans must have been installed.

Java in the Internet (applet/RMI)

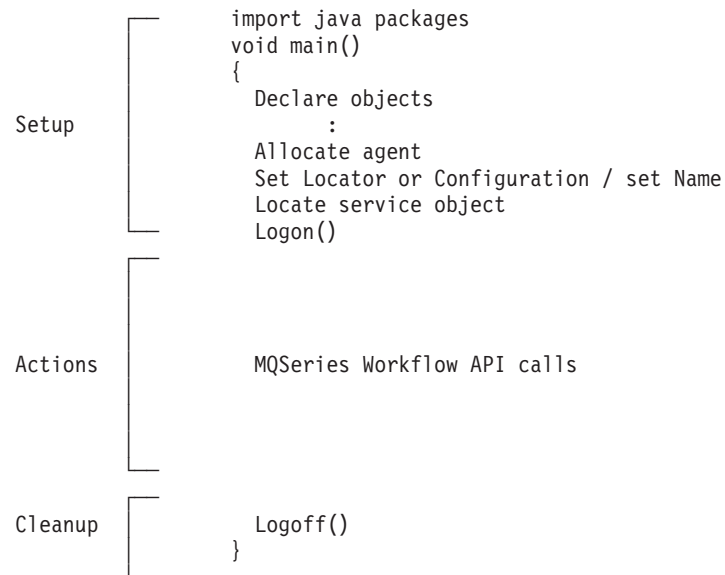


In this case, an RMI_LOCATOR policy must be used and an RMI agent must be specified. An RMI Agent and the Java API Beans must have been installed. The applet must be specified in the context of the agent object.

Coding an MQSeries Workflow client application in Java

Coding an MQSeries Workflow Java client application typically contains the following parts (which may not be delimited this clearly, however):

Java interface



To set up your program, you typically declare the program variables or objects you are going to use and you import the MQSeries Workflow Java API packages you need for your actions.

You then need to access a service object which represents the server you are going to ask services from. You do this by locating it via an appropriate agent. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your service object. All subsequent calls requiring client/server communication run through this session.

After a successful logon, you can issue action or program execution management methods in order to query or manage MQSeries Workflow objects you are authorized for.

At the end of your program, you log off in order to close the session to the server.

Coding an MQSeries Workflow activity implementation in Java

An MQSeries Workflow Java activity implementation typically contains the following parts:

| | | |
|---------|---|--|
| Setup | { | <pre> import java packages void main() { Declare objects : Allocate agent set Locator / set Name get execution agent InContainer() OutContainer() </pre> |
| Actions | { | <pre> read values set values </pre> |
| Cleanup | } | <pre> SetOutContainer() System.exit(rc) } </pre> |

To set up your program, you typically declare the program variables or objects you are going to use and you import the MQSeries Workflow Java API packages you need for your actions.

You then need to locate your execution agent object. You do this by allocating and asking the appropriate agent.

On platforms other than OS/390, an activity implementation can then retrieve the activity's input and output containers from the MQSeries Workflow program execution agent that started this program. On OS/390, the activity implementation in CICS or IMS gets input and output containers when it is started by the PES.

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQSeries Workflow program execution server. The return value tells the program execution server about the overall outcome of your program.

The output container is passed back to the MQSeries Workflow server which requested the execution of the activity implementation. The return code (`_RC`) can be used in exit or transition conditions in order to guide MQSeries Workflow navigation.

Your activity implementation can as well behave like a client application (see "Coding an MQSeries Workflow client application in C or C++" on page 123) and request services from an MQSeries Workflow server, normally the server from where its execution had been triggered. The `Passthrough()` method is then used instead of the `Logon()` method in order to log on to the server which caused the program execution with the user identification and authority known to the server from the work item start request.

Note: An activity implementation currently supports the `LOC_LOCATOR` policy only.

Java interface

Compiling

All programs developed for use with the MQSeries Workflow Java API Beans must import the packages provided by MQSeries Workflow. These files have been installed on your system.

JDK 1.1.x (x=6 or higher) can be used to compile and run your applications accessing the MQSeries Workflow Java API. A sample compile statement is:

```
javac -O <java file>.java
```

-O is an optional parameter denoting an optimized build. The CLASSPATH must point to fmcojapi.jar.

See *MQSeries Workflow for OS/390: Customization and Administration* for information on how to set up your system to use Java for OS/390.

Object management

Workflow process models, their instances, and resulting work items are all objects persistently stored in an MQSeries Workflow database. This means that they exist independently from an application program.

When persistent objects are queried by an application program, they are represented by *transient objects* which carry the states of the persistent objects at the time of the query. When multiple queries are issued, there can be multiple transient objects representing the same persistent object, even representing different states of that object.

The lifetime of transient objects is *fully managed* by you, because you know best when those objects are no longer needed, that is, when objects are unreferenced. Transient objects are, however, no longer available when your application program ends.

Some transient objects are *explicitly allocated* by you. These support objects that do not reflect persistent ones. Examples are the Agent or the ExecutionService object, which allows services to be requested from an execution server.

Transient objects, which do reflect persistent objects, are *implicitly allocated* by you when you create or retrieve persistent objects, for example, by querying.

Although the lifetime of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQSeries Workflow API.

As all resource memory is, in the end, owned by the application process itself, you can access all objects from different threads within that process. MQSeries Workflow does not hinder you from using threads, if they are supported; it is coded reentrantly. On the other hand, MQSeries Workflow does not explicitly support threads. If you want to access the same transient object from within different threads, you must synchronize the access to that object. Objects are **not** thread-safe.

Garbage collection when using Java API Beans

Garbage collection normally runs in the background without intervention by the Java programmer. This is also true in a distributed Java environment when objects communicate via the RMI transmission protocol. However, for other protocols, like CORBA's IIOP, provisions to remove unreferenced objects on the agent side have

to be made. When CORBA is used, then the memory management implicitly run by a Java Virtual Machine does not synchronize object removal on a client and the agent. Agent-side counterparts of unreferenced client objects are not automatically marked for removal. The Object Request Broker (ORB) cannot determine if any client is holding or not holding references to objects that it has registered (some ORBs can in fact do that, but they are using proprietary CORBA extensions to achieve this). Agent-side counterparts of client objects registered with an ORB by using a connect method have to be disconnected explicitly. When using MQSeries Workflow Java API Beans, the user is provided with a build-in garbage collection mechanism, the MQSeries Workflow Java API Beans Reaper, which does housekeeping when the transmission of data is done by a CORBA Object Request Broker (ORB). Before starting the MQSeries Workflow Java API Beans Agent a set of parameters controlling the reaper have to be set. These control parameters are:

- The reaper cycle time value, defined in milliseconds, is valid for both the client's reaper and the server's reaper. The default value is 300000 ms (5 minutes).
- The reaper threshold value is set to determine a maximum count for accumulated objects that are no longer referenced. The threshold takes precedence over the cycle time. Default value is 1000.
- The reaper ratio defines the relation between cycle times of both, client side reaper and server side reaper. The ratio is used as a multiplier for the server's reaper cycle, to calculate the cycle time for the client's reaper. The default value is 90, that means in fact 90% of the server's reaper cycle time. This ensures that the client side reaper actions always precede the server's side reaper actions.

The parameters are initially set at configuration time.

COBOL interface

The COBOL API runs as a layer on top of the C API, i.e., the COBOL CALL statements are effectively translated into the C functions with the same name. For this reason, what is said about the C API in this book generally applies to COBOL as well.

There are two ways to use the COBOL API:

1. directly through the Language Environment (LE) "CALL" mechanism.
2. by using the FMCPERF copybook and the COBOL "PERFORM" mechanism, which in turn uses the "CALL" mechanism.

Note: The COBOL calling sequences illustrated in the individual API descriptions are excerpts from FMCPERF.CPY. The paragraph name shown in each case is the name to be used as the PERFORM operand.

Calling the API

The COBOL API uses Language Environment (LE) interlanguage calls (ILCs) to call the C API. Therefore the compiler option PGMNAME(LM) must be used to allow

- a) calls to functions with names longer than 30 characters
- b) case sensitivity in function names

String handling

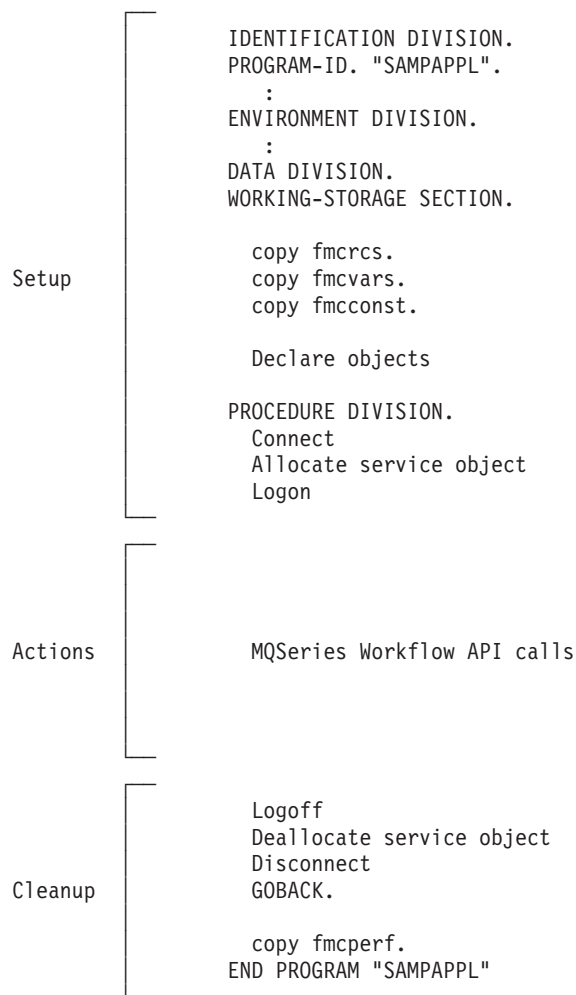
Since C strings (char *) are null-terminated, the COBOL programmer must provide null-terminated string parameters. String output parameters must be checked for

COBOL interface

the first occurrence of X'00' to get the correct value. A function cannot be called with String/PIC X(n) constants but rather must use a pointer referencing such a null-terminated PIC X(n).

Coding an MQSeries Workflow client application in COBOL

An MQSeries Workflow client application typically contains the following parts (which may not be delimited this clearly, however):



To set up your program, you typically declare the program variables or objects you are going to use and copy the MQSeries Workflow API copybooks you need for your actions.

You should then initialize the MQSeries Workflow API via the Connect API call so that resources held by the API are allocated correctly. Connect and Disconnect are to be called at the beginning and end of each thread, respectively.

You then need to allocate a service object which represents the server you are going to ask services from. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your service object. All subsequent calls requiring client/server communication run through this session.

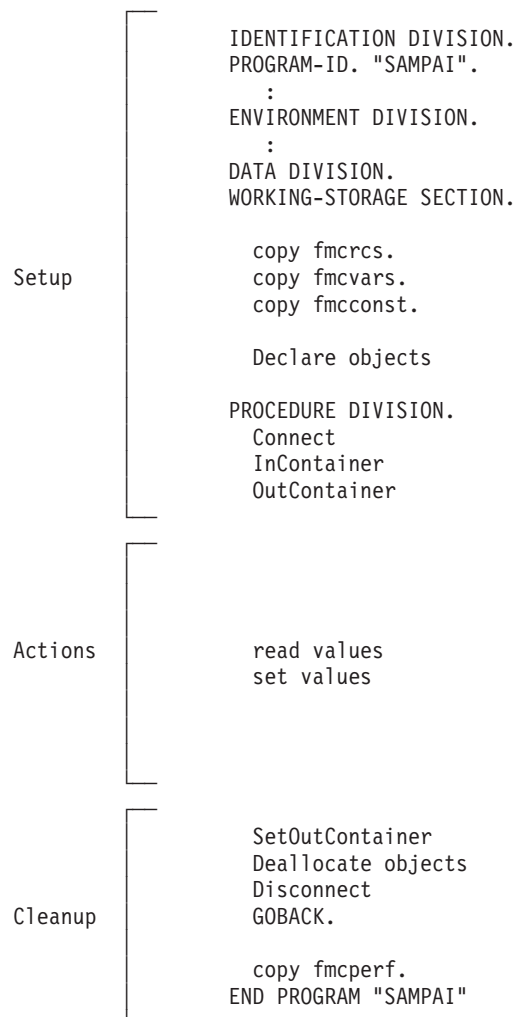
After a successful logon, you can issue action or program execution management API calls in order to query or manage MQSeries Workflow objects you are authorized for.

At the end of your program, you log off in order to close the session to the server and you deallocate any resources held by your program, especially the service object.

As a last step, you disconnect from the MQSeries Workflow API so that resources held by the API are deallocated correctly.

Coding an MQSeries Workflow activity implementation in COBOL

An MQSeries Workflow or activity implementation typically contains the following parts:



To set up your program, you typically declare the program variables or objects you are going to use and copy the MQSeries Workflow API copybooks you need for your actions.

COBOL interface

You should then initialize the MQSeries Workflow API via the Connect API call so that resources held by the API are allocated correctly. Connect and Disconnect are to be called at the beginning and end of each thread, respectively.

An activity implementation can then retrieve the activity's input and output containers from the MQSeries Workflow program execution server that started this program.

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQSeries Workflow program execution server. Any resources held by your program are deallocated. The return value tells the program execution server about the overall outcome of your program.

The output container as well as the return code of your program are passed back to the MQSeries Workflow server which requested the execution of the activity implementation. The return code (`_RC`) can be used in exit or transition conditions in order to guide MQSeries Workflow navigation.

As a last step, you disconnect from the MQSeries Workflow API so that resources held by the API are deallocated correctly.

Your activity implementation can also act like a client application (see "Coding an MQSeries Workflow client application in C or C++" on page 123) and request services from an MQSeries Workflow server, normally the server from where its execution was triggered. The Passthrough API call is then used instead of Logon in order to log on to the server which initiated program execution, with the user identification and authority known to the server from the work item start request.

Compiling and linking

COBOL programs must be compiled using IBM COBOL for OS/390 and VM, Version 2 Release 4 or higher.

The following copybooks are delivered with MQSeries Workflow for OS/390:

Table 4. Copybooks provided for COBOL programs

| Copybook | Contents |
|----------|----------------------------------|
| FMCCONST | Constants |
| FMCRCS | Return codes |
| FMCPERF | Subprograms of full API |
| FMCPERFL | Subprograms of the Container API |
| FMCVARS | Variables |

If you use the PERFORM mechanism, you must include FMCCONST, FMCRCS, FMCVARS, and either FMCPERF (if your program is using the full API) or FMCPERFL (if your program is an activity implementation using only the Container API).

If you use the CALL mechanism, you need not include any copybooks, but using FMCCONST and FMCRCS will provide you with all the values that MQSeries Workflow defines. FMCVARS can spare you some effort in declaring variables. FMCPERF and FMCPERFL are not useful in this case.

The following JCLs are provided as samples for the development and execution of MQSeries Workflow applications. They are located in the SFMCCNTL library delivered with MQSeries Workflow.

Table 5. JCLs provided for COBOL programs

| Job | Sample |
|--|----------|
| Native OS/390 COBOL Full API compile job | FMCHJ1BF |
| Native OS/390 COBOL API run job | FMCHJ1BR |
| CICS COBOL Full API compile job | FMCHJ2BF |
| CICS COBOL Container API Compile Job | FMCHJ2BC |
| IMS COBOL Container API Compile Job | FMCHJ3BC |

For more information about CICS/IMS specifics like stubs or precompiler, refer to the documentation of these components.

The compiler given as a prerequisite or newer versions can be used to compile and link your applications accessing the MQSeries Workflow API.

Mapping C to COBOL data types

Table 6 shows how to map C to COBOL data types:

Table 6. Mapping C to COBOL data types

| Type in C | Type in COBOL | BY VALUE / BY REFERENCE |
|------------------------------|---|------------------------------|
| XxxHandle | 01 ptr USAGE IS POINTER. (pointing to an object) | BY VALUE |
| XxxHandle * | 01 ptr USAGE IS POINTER. (pointing to a pointer to an object) | BY REFERENCE |
| char *, char const * | 01 ptr USAGE IS POINTER. (pointing to a PIC X(n)) | BY VALUE |
| FmcjCorrelID * | 01 ptr USAGE IS POINTER. (pointing to a PIC X(24)) | BY VALUE |
| FmcjBinary * | 01 ptr USAGE IS POINTER. | BY VALUE |
| FmcjCDateTime const * | 01 FmcjCDateTime. 05 the-year PIC 9(4) BINARY. 05 the-month PIC 9(4) BINARY. 05 the-day PIC 9(4) BINARY. 05 the-hour PIC 9(4) BINARY. 05 the-minute PIC 9(4) BINARY. 05 the-second PIC 9(4) BINARY. | BY REFERENCE |
| int, long, signed long, enum | 01 int PIC S9(9) BINARY. | BY VALUE |
| APIRET | 01 int PIC S9(9) BINARY. | n/a (not used as parameters) |
| unsigned long | 01 ulong PIC 9(9) BINARY. | BY VALUE |
| unsigned short | 01 ushort PIC 9(4) BINARY. | BY VALUE |

COBOL interface

Table 6. Mapping C to COBOL data types (continued)

| Type in C | Type in COBOL | BY VALUE / BY REFERENCE |
|-----------------------|---------------------------|-------------------------|
| double | 01 double COMP-2. | BY VALUE |
| bool (1=true 0=false) | 01 bool PIC 9 BINARY. | BY VALUE |
| long * | 01 int PIC S9(9) BINARY. | BY REFERENCE |
| double * | 01 double COMP-2. | BY REFERENCE |
| unsigned long const * | 01 ulong PIC 9(9) BINARY. | BY VALUE ⁴ |

Name changes between COBOL and C

For some of the C functions, synonyms are declared in the form of

```
#define functionA functionB
```

This is to reflect the fact that routines belonging to a superclass are already available to perform the desired function.

In these cases, if you want to call functionA directly via CALL in COBOL, you must use functionB instead. All functions belonging to this category are listed in the table below.

Note: If you use the PERFORM statement with the FMCPERF copybook, you must perform a paragraph whose name is an abbreviated version (see Table 8 on page 144) of the name shown in the left-hand column of the following table. This paragraph then contains a call to the proper COBOL subprogram. For example, to call "FmcjActivityInstanceNotificationSetDescription" via PERFORM, you would code:

```
PERFORM FmcjAINSetDescription
```

which contains

```
CALL FmcjItemSetDescription
```

Table 7. Function name mapping

| C function | Corresponding COBOL subprogram |
|---|--------------------------------|
| FmcjActivityInstanceNotificationCategory | FmcjItemCategory |
| FmcjActivityInstanceNotificationCategoryIsNull | FmcjItemCategoryIsNull |
| FmcjActivityInstanceNotificationCreationTime | FmcjItemCreationTime |
| FmcjActivityInstanceNotificationDelete | FmcjItemDelete |
| FmcjActivityInstanceNotificationDescription | FmcjItemDescription |
| FmcjActivityInstanceNotificationDescriptionIsNull | FmcjItemDescriptionIsNull |
| FmcjActivityInstanceNotificationDocumentation | FmcjItemDocumentation |
| FmcjActivityInstanceNotificationDocumentationIsNull | FmcjItemDocumentationIsNull |
| FmcjActivityInstanceNotificationEndTime | FmcjItemEndTime |
| FmcjActivityInstanceNotificationEndTimeIsNull | FmcjItemEndTimeIsNull |
| FmcjActivityInstanceNotificationEqual | FmcjItemEqual |

4. The respective C routine expects the actual value to be passed rather than a pointer thereto. This is apparently the result of internal optimization when the C routine is compiled.

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|--|--|
| FmcjActivityInstanceNotificationIcon | FmcjItemIcon |
| FmcjActivityInstanceNotificationInContainerName | FmcjItemInContainerName |
| FmcjActivityInstanceNotificationIsComplete | FmcjItemIsComplete |
| FmcjActivityInstanceNotificationKind | FmcjItemKind |
| FmcjActivityInstanceNotificationLastModificationTime | FmcjItemLastModificationTime |
| FmcjActivityInstanceNotificationName | FmcjItemName |
| FmcjActivityInstanceNotificationObtainProcessInstanceMonitor | FmcjItemObtainProcessInstanceMonitor |
| FmcjActivityInstanceNotificationOutContainerName | FmcjItemOutContainerName |
| FmcjActivityInstanceNotificationOwner | FmcjItemOwner |
| FmcjActivityInstanceNotificationPersistentOid | FmcjItemPersistentOid |
| FmcjActivityInstanceNotificationProcessAdmin | FmcjItemProcessAdmin |
| FmcjActivityInstanceNotificationProcessInstance | FmcjItemProcessInstance |
| FmcjActivityInstanceNotificationProcessInstanceName | FmcjItemProcessInstanceName |
| FmcjActivityInstanceNotificationProcessInstanceState | FmcjItemProcessInstanceState |
| FmcjActivityInstanceNotificationProcessInstanceSystemGroupName | FmcjItemProcessInstanceSystemGroupName |
| FmcjActivityInstanceNotificationProcessInstanceSystemName | FmcjItemProcessInstanceSystemName |
| FmcjActivityInstanceNotificationReceivedAs | FmcjItemReceivedAs |
| FmcjActivityInstanceNotificationReceivedTime | FmcjItemReceivedTime |
| FmcjActivityInstanceNotificationRefresh | FmcjItemRefresh |
| FmcjActivityInstanceNotificationSetDescription | FmcjItemSetDescription |
| FmcjActivityInstanceNotificationSetName | FmcjItemSetName |
| FmcjActivityInstanceNotificationStartTime | FmcjItemStartTime |
| FmcjActivityInstanceNotificationStartTimeIsNull | FmcjItemStartTimeIsNull |
| FmcjActivityInstanceNotificationTransfer | FmcjItemTransfer |
| FmcjActivityInstanceNotificationUpdate | FmcjItemUpdate |
| FmcjExecutionServiceIsLoggedOn | FmcjServiceIsLoggedOn |
| FmcjExecutionServiceRefresh | FmcjServiceRefresh |
| FmcjExecutionServiceSetPassword | FmcjServiceSetPassword |
| FmcjExecutionServiceSetTimeout | FmcjServiceSetTimeout |
| FmcjExecutionServiceSystemGroupName | FmcjServiceSystemGroupName |
| FmcjExecutionServiceSystemName | FmcjServiceSystemName |
| FmcjExecutionServiceTimeout | FmcjServiceTimeout |
| FmcjExecutionServiceUserID | FmcjServiceUserID |
| FmcjExecutionServiceUserSettings | FmcjServiceUserSettings |
| FmcjProcessInstanceListDelete | FmcjPersistentListDelete |
| FmcjProcessInstanceListDescription | FmcjPersistentListDescription |
| FmcjProcessInstanceListDescriptionIsNull | FmcjPersistentListDescriptionIsNull |
| FmcjProcessInstanceListFilter | FmcjPersistentListFilter |
| FmcjProcessInstanceListFilterIsNull | FmcjPersistentListFilterIsNull |
| FmcjProcessInstanceListName | FmcjPersistentListName |

COBOL interface

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|---|--|
| FmcjProcessInstanceListOwnerOfList | FmcjPersistentListOwnerOfList |
| FmcjProcessInstanceListOwnerOfListIsNull | FmcjPersistentListOwnerOfListIsNull |
| FmcjProcessInstanceListRefresh | FmcjPersistentListRefresh |
| FmcjProcessInstanceListSetDescription | FmcjPersistentListSetDescription |
| FmcjProcessInstanceListSetFilter | FmcjPersistentListSetFilter |
| FmcjProcessInstanceListSetSortCriteria | FmcjPersistentListSetSortCriteria |
| FmcjProcessInstanceListSetThreshold | FmcjPersistentListSetThreshold |
| FmcjProcessInstanceListSortCriteria | FmcjPersistentListSortCriteria |
| FmcjProcessInstanceListSortCriteriaIsNull | FmcjPersistentListSortCriteriaIsNull |
| FmcjProcessInstanceListThreshold | FmcjPersistentListThreshold |
| FmcjProcessInstanceListThresholdIsNull | FmcjPersistentListThresholdIsNull |
| FmcjProcessInstanceListType | FmcjPersistentListType |
| FmcjProcessInstanceMonitorActivityInstances | FmcjBlockInstanceMonitorActivityInstances |
| FmcjProcessInstanceMonitorControlConnectorInstances | FmcjBlockInstanceMonitor ControlConnectorInstances |
| FmcjProcessInstanceMonitorObtainBlockInstanceMonitor | FmcjBlockInstanceMonitor ObtainBlockInstanceMonitor |
| FmcjProcessInstanceMonitorObtainProcessInstanceMonitor | FmcjBlockInstanceMonitor ObtainProcessInstanceMonitor |
| FmcjProcessInstanceMonitorRefresh | FmcjBlockInstanceMonitorRefresh |
| FmcjProcessInstanceNotificationCategory | FmcjItemCategory |
| FmcjProcessInstanceNotificationCategoryIsNull | FmcjItemCategoryIsNull |
| FmcjProcessInstanceNotificationCreationTime | FmcjItemCreationTime |
| FmcjProcessInstanceNotificationDelete | FmcjItemDelete |
| FmcjProcessInstanceNotificationDescription | FmcjItemDescription |
| FmcjProcessInstanceNotificationDescriptionIsNull | FmcjItemDescriptionIsNull |
| FmcjProcessInstanceNotificationDocumentation | FmcjItemDocumentation |
| FmcjProcessInstanceNotificationDocumentationIsNull | FmcjItemDocumentationIsNull |
| FmcjProcessInstanceNotificationEndTime | FmcjItemEndTime |
| FmcjProcessInstanceNotificationEndTimeIsNull | FmcjItemEndTimeIsNull |
| FmcjProcessInstanceNotificationEqual | FmcjItemEqual |
| FmcjProcessInstanceNotificationIcon | FmcjItemIcon |
| FmcjProcessInstanceNotificationInContainerName | FmcjItemInContainerName |
| FmcjProcessInstanceNotificationIsComplete | FmcjItemIsComplete |
| FmcjProcessInstanceNotificationKind | FmcjItemKind |
| FmcjProcessInstanceNotificationLastModificationTime | FmcjItemLastModificationTime |
| FmcjProcessInstanceNotificationName | FmcjItemName |
| FmcjProcessInstanceNotificationObtainProcessInstanceMonitor | FmcjItemObtainProcessInstanceMonitor |
| FmcjProcessInstanceNotificationOutContainerName | FmcjItemOutContainerName |
| FmcjProcessInstanceNotificationOwner | FmcjItemOwner |
| FmcjProcessInstanceNotificationPersistentOid | FmcjItemPersistentOid |

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|---|--|
| FmcjProcessInstanceNotificationProcessAdmin | FmcjItemProcessAdmin |
| FmcjProcessInstanceNotificationProcessInstance | FmcjItemProcessInstance |
| FmcjProcessInstanceNotificationProcessInstanceName | FmcjItemProcessInstanceName |
| FmcjProcessInstanceNotificationProcessInstanceState | FmcjItemProcessInstanceState |
| FmcjProcessInstanceNotificationProcessInstanceSystemGroupName | FmcjItemProcessInstanceSystemGroupName |
| FmcjProcessInstanceNotificationProcessInstanceSystemName | FmcjItemProcessInstanceSystemName |
| FmcjProcessInstanceNotificationReceivedAs | FmcjItemReceivedAs |
| FmcjProcessInstanceNotificationReceivedTime | FmcjItemReceivedTime |
| FmcjProcessInstanceNotificationRefresh | FmcjItemRefresh |
| FmcjProcessInstanceNotificationSetDescription | FmcjItemSetDescription |
| FmcjProcessInstanceNotificationSetName | FmcjItemSetName |
| FmcjProcessInstanceNotificationStartTime | FmcjItemStartTime |
| FmcjProcessInstanceNotificationStartTimeIsNull | FmcjItemStartTimeIsNull |
| FmcjProcessInstanceNotificationTransfer | FmcjItemTransfer |
| FmcjProcessInstanceNotificationUpdate | FmcjItemUpdate |
| FmcjProcessTemplateInContainer | FmcjProcessTemplateInitialInContainer |
| FmcjProcessTemplateListDelete | FmcjPersistentListDelete |
| FmcjProcessTemplateListDescription | FmcjPersistentListDescription |
| FmcjProcessTemplateListDescriptionIsNull | FmcjPersistentListDescriptionIsNull |
| FmcjProcessTemplateListFilter | FmcjPersistentListFilter |
| FmcjProcessTemplateListFilterIsNull | FmcjPersistentListFilterIsNull |
| FmcjProcessTemplateListName | FmcjPersistentListName |
| FmcjProcessTemplateListOwnerOfList | FmcjPersistentListOwnerOfList |
| FmcjProcessTemplateListOwnerOfListIsNull | FmcjPersistentListOwnerOfListIsNull |
| FmcjProcessTemplateListRefresh | FmcjPersistentListRefresh |
| FmcjProcessTemplateListSetDescription | FmcjPersistentListSetDescription |
| FmcjProcessTemplateListSetFilter | FmcjPersistentListSetFilter |
| FmcjProcessTemplateListSetSortCriteria | FmcjPersistentListSetSortCriteria |
| FmcjProcessTemplateListSetThreshold | FmcjPersistentListSetThreshold |
| FmcjProcessTemplateListSortCriteria | FmcjPersistentListSortCriteria |
| FmcjProcessTemplateListSortCriteriaIsNull | FmcjPersistentListSortCriteriaIsNull |
| FmcjProcessTemplateListThreshold | FmcjPersistentListThreshold |
| FmcjProcessTemplateListThresholdIsNull | FmcjPersistentListThresholdIsNull |
| FmcjProcessTemplateListType | FmcjPersistentListType |
| FmcjProgramDataExecutionMode | FmcjProgramTemplateExecutionMode |
| FmcjProgramDataExecutionUser | FmcjProgramTemplateExecutionUser |
| FmcjProgramDataProgramTrusted | FmcjProgramTemplateProgramTrusted |
| FmcjReadOnlyContainerAllLeafCount | FmcjContainerAllLeafCount |
| FmcjReadOnlyContainerAllLeaves | FmcjContainerAllLeaves |
| FmcjReadOnlyContainerArrayBinaryLength | FmcjContainerArrayBinaryLength |

COBOL interface

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|---|--------------------------------|
| FmcjReadOnlyContainerArrayBinaryValue | FmcjContainerArrayBinaryValue |
| FmcjReadOnlyContainerArrayFloatValue | FmcjContainerArrayFloatValue |
| FmcjReadOnlyContainerArrayLongValue | FmcjContainerArrayLongValue |
| FmcjReadOnlyContainerArrayStringLength | FmcjContainerArrayStringLength |
| FmcjReadOnlyContainerArrayStringValue | FmcjContainerArrayStringValue |
| FmcjReadOnlyContainerBinaryLength | FmcjContainerBinaryLength |
| FmcjReadOnlyContainerBinaryValue | FmcjContainerBinaryValue |
| FmcjReadOnlyContainerFloatValue | FmcjContainerFloatValue |
| FmcjReadOnlyContainerGetElement | FmcjContainerGetElement |
| FmcjReadOnlyContainerLeafCount | FmcjContainerLeafCount |
| FmcjReadOnlyContainerLeaves | FmcjContainerLeaves |
| FmcjReadOnlyContainerLongValue | FmcjContainerLongValue |
| FmcjReadOnlyContainerMemberCount | FmcjContainerMemberCount |
| FmcjReadOnlyContainerStringLength | FmcjContainerStringLength |
| FmcjReadOnlyContainerStringValue | FmcjContainerStringValue |
| FmcjReadOnlyContainerStructMembers | FmcjContainerStructMembers |
| FmcjReadOnlyContainerType | FmcjContainerType |
| FmcjReadWriteContainerAllLeafCount | FmcjContainerAllLeafCount |
| FmcjReadWriteContainerAllLeaves | FmcjContainerAllLeaves |
| FmcjReadWriteContainerArrayBinaryLength | FmcjContainerArrayBinaryLength |
| FmcjReadWriteContainerArrayBinaryValue | FmcjContainerArrayBinaryValue |
| FmcjReadWriteContainerArrayFloatValue | FmcjContainerArrayFloatValue |
| FmcjReadWriteContainerArrayLongValue | FmcjContainerArrayLongValue |
| FmcjReadWriteContainerArrayStringLength | FmcjContainerArrayStringLength |
| FmcjReadWriteContainerArrayStringValue | FmcjContainerArrayStringValue |
| FmcjReadWriteContainerBinaryLength | FmcjContainerBinaryLength |
| FmcjReadWriteContainerBinaryValue | FmcjContainerBinaryValue |
| FmcjReadWriteContainerFloatValue | FmcjContainerFloatValue |
| FmcjReadWriteContainerGetElement | FmcjContainerGetElement |
| FmcjReadWriteContainerLeafCount | FmcjContainerLeafCount |
| FmcjReadWriteContainerLeaves | FmcjContainerLeaves |
| FmcjReadWriteContainerLongValue | FmcjContainerLongValue |
| FmcjReadWriteContainerMemberCount | FmcjContainerMemberCount |
| FmcjReadWriteContainerStringLength | FmcjContainerStringLength |
| FmcjReadWriteContainerStringValue | FmcjContainerStringValue |
| FmcjReadWriteContainerStructMembers | FmcjContainerStructMembers |
| FmcjReadWriteContainerType | FmcjContainerType |
| FmcjWorkitemCategory | FmcjItemCategory |
| FmcjWorkitemCategoryIsNull | FmcjItemCategoryIsNull |
| FmcjWorkitemCreationTime | FmcjItemCreationTime |

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|--|--|
| FmcjWorkitemDelete | FmcjItemDelete |
| FmcjWorkitemDescription | FmcjItemDescription |
| FmcjWorkitemDescriptionIsNull | FmcjItemDescriptionIsNull |
| FmcjWorkitemDocumentation | FmcjItemDocumentation |
| FmcjWorkitemDocumentationIsNull | FmcjItemDocumentationIsNull |
| FmcjWorkitemEndTime | FmcjItemEndTime |
| FmcjWorkitemEndTimeIsNull | FmcjItemEndTimeIsNull |
| FmcjWorkitemEqual | FmcjItemEqual |
| FmcjWorkitemIcon | FmcjItemIcon |
| FmcjWorkitemInContainerName | FmcjItemInContainerName |
| FmcjWorkitemIsComplete | FmcjItemIsComplete |
| FmcjWorkitemKind | FmcjItemKind |
| FmcjWorkitemLastModificationTime | FmcjItemLastModificationTime |
| FmcjWorkitemName | FmcjItemName |
| FmcjWorkitemObtainProcessInstanceMonitor | FmcjItemObtainProcessInstanceMonitor |
| FmcjWorkitemOutContainerName | FmcjItemOutContainerName |
| FmcjWorkitemOwner | FmcjItemOwner |
| FmcjWorkitemPersistentOid | FmcjItemPersistentOid |
| FmcjWorkitemProcessAdmin | FmcjItemProcessAdmin |
| FmcjWorkitemProcessInstance | FmcjItemProcessInstance |
| FmcjWorkitemProcessInstanceName | FmcjItemProcessInstanceName |
| FmcjWorkitemProcessInstanceState | FmcjItemProcessInstanceState |
| FmcjWorkitemProcessInstanceSystemGroupName | FmcjItemProcessInstanceSystemGroupName |
| FmcjWorkitemProcessInstanceSystemName | FmcjItemProcessInstanceSystemName |
| FmcjWorkitemReceivedAs | FmcjItemReceivedAs |
| FmcjWorkitemReceivedTime | FmcjItemReceivedTime |
| FmcjWorkitemRefresh | FmcjItemRefresh |
| FmcjWorkitemSetDescription | FmcjItemSetDescription |
| FmcjWorkitemSetName | FmcjItemSetName |
| FmcjWorkitemStartTime | FmcjItemStartTime |
| FmcjWorkitemStartTimeIsNull | FmcjItemStartTimeIsNull |
| FmcjWorkitemTransfer | FmcjItemTransfer |
| FmcjWorkitemUpdate | FmcjItemUpdate |
| FmcjWorklistDelete | FmcjPersistentListDelete |
| FmcjWorklistDescription | FmcjPersistentListDescription |
| FmcjWorklistDescriptionIsNull | FmcjPersistentListDescriptionIsNull |
| FmcjWorklistFilter | FmcjPersistentListFilter |
| FmcjWorklistFilterIsNull | FmcjPersistentListFilterIsNull |
| FmcjWorklistName | FmcjPersistentListName |
| FmcjWorklistOwnerOfList | FmcjPersistentListOwnerOfList |

COBOL interface

Table 7. Function name mapping (continued)

| C function | Corresponding COBOL subprogram |
|--------------------------------|--------------------------------------|
| FmcjWorklistOwnerOfListIsNull | FmcjPersistentListOwnerOfListIsNull |
| FmcjWorklistRefresh | FmcjPersistentListRefresh |
| FmcjWorklistSetDescription | FmcjPersistentListSetDescription |
| FmcjWorklistSetFilter | FmcjPersistentListSetFilter |
| FmcjWorklistSetSortCriteria | FmcjPersistentListSetSortCriteria |
| FmcjWorklistSetThreshold | FmcjPersistentListSetThreshold |
| FmcjWorklistSortCriteria | FmcjPersistentListSortCriteria |
| FmcjWorklistSortCriteriaIsNull | FmcjPersistentListSortCriteriaIsNull |
| FmcjWorklistThreshold | FmcjPersistentListThreshold |
| FmcjWorklistThresholdIsNull | FmcjPersistentListThresholdIsNull |
| FmcjWorklistType | FmcjPersistentListType |

To cope with the COBOL restriction of 30 characters per COBOL word, some class name prefixes and function and constant names have been abbreviated. The abbreviations for the class name prefixes are listed in the table below.

Table 8. Class prefix abbreviations

| Class Name | Abbreviation |
|--|--------------|
| FmcjActivityInstance | FmcjAI |
| FmcjActivityInstanceList | FmcjAIL |
| FmcjActivityInstanceNotification | FmcjAIN |
| FmcjActivityInstanceNotificationVector | FmcjAINV |
| FmcjActivityInstanceVector | FmcjAIV |
| FmcjBlockInstanceMonitor | FmcjBIM |
| FmcjContainer | FmcjC |
| FmcjContainerElement | FmcjCE |
| FmcjContainerElementVector | FmcjCEV |
| FmcjControlConnectorInstance | FmcjCCI |
| FmcjControlConnectorInstanceVector | FmcjCCIV |
| FmcjDllOptions | FmcjDO |
| FmcjExecutionData | FmcjED |
| FmcjExecutionService | FmcjES |
| FmcjExeOptions | FmcjExeO |
| FmcjExternalOptions | FmcjExtO |
| FmcjImplementationData | FmcjID |
| FmcjImplementationDataVector | FmcjIDV |
| FmcjPersistentList | FmcjPL |
| FmcjPerson | FmcjP |
| FmcjPoint | FmcjPnt |
| FmcjPointVector | FmcjPntV |
| FmcjProcessInstance | FmcjPI |
| FmcjProcessInstanceList | FmcjPIL |

Table 8. Class prefix abbreviations (continued)

| Class Name | Abbreviation |
|---------------------------------------|--------------|
| FmcjProcessInstanceListVector | FmcjPILV |
| FmcjProcessInstanceMonitor | FmcjPIM |
| FmcjProcessInstanceNotification | FmcjPIN |
| FmcjProcessInstanceNotificationVector | FmcjPINV |
| FmcjProcessInstanceVector | FmcjPIV |
| FmcjProcessTemplate | FmcjPT |
| FmcjProcessTemplateList | FmcjPTL |
| FmcjProcessTemplateListVector | FmcjPTLV |
| FmcjProcessTemplateVector | FmcjPTV |
| FmcjProgramData | FmcjPD |
| FmcjProgramTemplate | FmcjPgT |
| FmcjReadOnlyContainer | FmcjROC |
| FmcjReadWriteContainer | FmcjRWC |
| FmcjService | FmcjSrv |
| FmcjStringVector | FmcjStrV |
| FmcjSymbolLayout | FmcjSL |
| FmcjWorkitem | FmcjWI |
| FmcjWorkitemVector | FmcjWIV |
| FmcjWorklist | FmcjWL |

The abbreviations for function, constant, and error code names are listed in the table below (note that these abbreviations apply after the class name prefix abbreviations). Instead of constructing the names with the help of this table you can also search the FMCPERF copybook for the C function name to get the corresponding COBOL function and variable names.

Table 9. Abbreviations for COBOL naming

| Original String | Abbreviation |
|-----------------|--------------|
| Activity | Act |
| ACTIVITY | ACT |
| Administration | Admin |
| ADMINSTRATION | ADMIN |
| Already | Alr |
| ALREADY | ALR |
| Authorization | Auth |
| AUTHORIZATION | AUTH |
| Authorized | Auth |
| AUTHORIZED | AUTH |
| Backward | Backw |
| BACKWARD | BACKW |
| Categories | Categs |
| CATEGORIES | CATEGS |

COBOL interface

Table 9. Abbreviations for COBOL naming (continued)

| Original String | Abbreviation |
|----------------------|---------------|
| Category | Categ |
| CATEGORY | CATEG |
| CHECKOUT | CHKOUT |
| Container | Ctnr |
| CONTAINER | CTNR |
| ControlConnector | ContrConn |
| CONTROLCONNECTOR | CONTRCONN |
| DATA-MEMBER | D-M |
| Definition | Def |
| DEFINITION | DEF |
| Directory | Dir |
| DIRECTORY | DIR |
| DOCUMENT | DOC |
| Executable | Exec |
| EXECUTABLE | EXEC |
| EXTERNAL | EXT |
| ForeGround | ForeGr |
| FOREGROUND | FOREGR |
| Forward | Forw |
| FORWARD | FORW |
| FOUND-FOR-AUTO-START | FND-FR-AUT-ST |
| FROM | FRM |
| IMPLEMENTATION | IMPL |
| Instance | Inst |
| INSTANCE | INST |
| INVALID | INVAL |
| Invocation | Inv |
| INVOCATION | INV |
| Location | Loc |
| LOCATION | LOC |
| Mapping | Map |
| MAPPING | MAP |
| MESSAGE | MSG |
| Monitor | Mon |
| MONITOR | MON |
| Notification | Notif |
| NOTIFICATION | NOTIF |
| Notified | Notif |
| NOTIFIED | NOTIF |
| Object | Obj |

Table 9. Abbreviations for COBOL naming (continued)

| Original String | Abbreviation |
|-----------------|--------------|
| OBJECT | OBJ |
| Organization | Org |
| ORGANIZATION | ORG |
| Parameter | Parm |
| PARAMETER | PARM |
| PersistentOidOf | PersOidOf |
| PERSISTENTOIDOF | PERSOIDOF |
| Persons | Pers |
| PERSON | PERS |
| Process | Proc |
| PROCESS | PROC |
| Program | Prog |
| PROGRAM | PROG |
| Second | Sec |
| SECOND | SEC |
| Service | Serv |
| SERVICE | SERV |
| Started | Strtd |
| STARTED | STRTD |
| STRUCTURE | STR |
| SUB-PROC | SB-PRC |
| SUPPORTED | SUPP |
| Suspension | Susp |
| SUSPENSION | SUSP |
| System | Syst |
| SYSTEM | SYST |
| Template | Templ |
| TEMPLATE | TEMPL |
| Terminated | Term |
| TERMINATED | TERM |
| Transition | Trans |
| TRANSITION | TRANS |

The C API uses some variable names that are reserved words in COBOL. These variable names were changed in the COBOL API by appending "Value" to the variable name. All variables belonging to this category are listed below.

```

year      month   day      hour    minute  second  data
file      function index   input   line    output  owner
password  program  service  time    type    timeout
    
```

Finally, the variable name "value" is used in the C API with different types. In COBOL this variable is renamed according to its type:

COBOL interface

intValue, doubleValue, pointerValue

Example of the use of strings

The following code calls a C function with the signature `char* cfunc(char* x)`.

Note: The SETADDR routine sets a pointer variable to the address of a local string. The GETADDR routine copies a string referred to by a pointer variable to a local string.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "STRTEST".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PTR1 USAGE IS POINTER VALUE NULL.
01 PTR2 USAGE IS POINTER VALUE NULL.
01 STRING-STRUCT1.
05 X PIC X(20).
01 STRING-STRUCT2.
05 Y PIC X(20).
01 STRLEN PIC 99 VALUE 0.
PROCEDURE DIVISION.
MOVE z"Initial String" TO X.
CALL "SETADDR" USING STRING-STRUCT1 PTR1.
CALL "cfunc" USING BY VALUE PTR1
RETURNING PTR2.
CALL "GETADDR" USING STRING-STRUCT2 PTR2.
INSPECT Y TALLYING STRLEN FOR CHARACTERS
BEFORE INITIAL X"00".
DISPLAY "Y is " Y(1:STRLEN).
STOP RUN.
END PROGRAM "STRTEST".

IDENTIFICATION DIVISION.
PROGRAM-ID. "SETADDR".
DATA DIVISION.
LINKAGE SECTION.
01 PTR3 USAGE IS POINTER.
01 STRING-STRUCT3.
05 Z PIC X(20).
PROCEDURE DIVISION USING BY REFERENCE STRING-STRUCT3 PTR3.
SET PTR3 TO ADDRESS OF Z.
GOBACK.
END PROGRAM "SETADDR".

IDENTIFICATION DIVISION.
PROGRAM-ID. "GETADDR".
DATA DIVISION.
LINKAGE SECTION.
01 PTR4 USAGE IS POINTER.
01 STRING-STRUCT4.
05 Z PIC X(20).
01 DUMMY-STRUCT.
05 W PIC X(20).
PROCEDURE DIVISION USING BY REFERENCE STRING-STRUCT4 PTR4.
SET ADDRESS OF DUMMY-STRUCT TO PTR4.
MOVE W TO Z.
GOBACK.
END PROGRAM "GETADDR".
```


XML message interface

The following sections provide a description of the MQSeries Workflow XML message-based interface. It explains the format of a message and how XML can be used for:

- Sending requests to MQSeries Workflow

An action can be started on the execution server by sending a message to the MQSeries Workflow XML input queue.

This allows any application that supports the MQSeries Workflow XML message format to request an action from MQSeries Workflow.

- Invoking an activity implementation

An activity implementation is invoked by MQSeries Workflow by sending an appropriate message to a user-defined MQSeries queue.

This allows you to start any application listening on an MQSeries queue. The queue can be input to any MQSeries application that can handle XML messages. This can be your own application or a commercial program, such as MQSeries Integrator V2.

The MQSeries Workflow message

MQSeries Workflow uses MQSeries to exchange messages. An MQSeries message consists of two parts:

1. The MQSeries message descriptor (MQMD), which contains structured data describing the message
2. The application data, which contains the MQSeries Workflow XML message itself

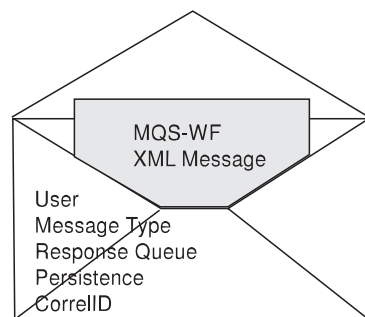


Figure 25. MQSeries Workflow message

Refer to "MQSeries messages" in the *IBM MQSeries Application Programming Guide*.

Relevant MQSeries Message Descriptor (MQMD) fields

The following fields of the MQSeries message descriptor are used by MQSeries Workflow:

- UserIdentifier

The user who sent the message. For request messages sent to MQSeries Workflow, this information is used as the MQSeries Workflow user on whose behalf the request is performed. Also, authorization checks are performed using this user ID. For invoke messages sent by MQSeries Workflow, this field contains the user on whose behalf the activity implementation is to be started.

- MsgId

XML message interface

A fixed string indicating that this message contains an MQSeries Workflow XML message. Its value is "FMCXML".

- ReplyToQ/ReplyToQMgr
Specifies the queue and queue manager the response should be sent to.
- Persistence
Specifies whether the message is persistent or transient. For requests sent to MQSeries Workflow, the MQSeries Workflow response has the same persistence as the request. XML requests sent by MQSeries Workflow are persistent and responses sent by invoked activity implementations should also be persistent.
- Expiration
Can be set to a period of time expressed in tenths of a second for transient messages; should be set to unlimited (MQEI_UNLIMITED) for persistent messages. For requests sent to MQSeries Workflow, the expiration of the MQSeries Workflow response is set to the expiration value in the request minus the time spent on execution. XML requests sent by MQSeries Workflow have an unlimited expiration time.
- CorrelID
Data that can be used to relate a response message to a request message. For requests sent to MQSeries Workflow, the MQSeries Workflow response contains the same correlation ID as the request. XML requests sent by MQSeries Workflow contain a correlation ID and responses sent by an application should return that correlation ID.

For detailed information, refer to the *MQSeries System Administration*.

The application data

MQSeries Workflow uses the XML 1.0 standard for message description. Refer to <http://www.w3.org/TR/REC-xml>

for the XML Reference.

In general, an MQSeries Workflow XML message contains the following information:

- An MQSeries Workflow XML message header, that is, information that is common for all messages, for example, the user context
- The request or response, for example, a "ProcessTemplateExecute" request
- The parameters needed to execute the request or to analyze the response, for example, an input container.

When processing an MQSeries Workflow XML message, MQSeries Workflow checks if the message has the correct format.

The XML message header: The MQSeries Workflow XML message header contains the following information:

- If a response should be sent.
With a message-based interface, both synchronous and asynchronous request/response scenarios can occur. That is why the creation of a response to a given request is made optional. However, even if responses are generally not desired, an exceptional response to report an error can still be required. Such options are provided to request:
 - No ("No")
 - Only error ("IfError")

- All ("Yes")

responses which are sent to the response queue specified in the MQMD of the request message.

- The user context

In this field, you can specify up to 254 bytes of context data that can be used for correlating a request and a response. The user context data specified in a request to MQSeries Workflow is returned in the associated response.

Therefore, the necessity to keep state information in the component sending the message is avoided. For example, when a message is routed through an intermediary like MQSeries Integrator V2, it can be desirable to route the response back through the intermediary, which then in turn will send the message back to the original requester. The user context data can contain information in order to keep the original requester, or even an entire route, without requiring the intermediary to maintain state information.

Container data: For a general introduction on containers, refer to "Handling containers" on page 30. The following example shows a container with two data structures and one data member that is an array:

<!-- Note: The container structure represented here has the following layout:

```

    CreditData
      Customer: CustomerData
      Amount: Integer
      Currency: String
    End CreditData

    CustomerData
      Name: String
      Account[2]:Integer
    End CustomerData
-->
<CreditData>
  <Customer>
    <Name>User1</Name>
    <Account>4711</Account>
    <Account>0007</Account>
  </Customer>
  <Amount>100000</Amount>
  <Currency>CurrencyX</Currency>
</CreditData>
```

The following rules apply to containers in the message-based interface:

- A container is identified by its type, that is, the name of the associated data structure. Container elements are specified by their name; their type is not part of the XML message.
- There are both atomic and complex data structures. Atomic data is mapped to parsed character data (PCDATA) elements, while complex data structures are decomposed into elements according to their structure.
- The structure of XML elements representing a container reflects the associated data structure. Therefore, data member names are not prefixed; there is no need for a dotted name representation.

Note that the context-free nature of XML does not allow for data structures having the same names as data members. Also, two data members with the same name must be of the same type even if they are contained in different data structures. When the MQSeries Workflow Buildtime Verification encounters one of these situations, it issues a warning.

- Atomic elements containing strings and numbers can be coded directly.

XML message interface

- For boolean types, values "false" and "true" (case insensitive) should be used. Values 0 and 1 are also supported.
- Binary data must be encoded into its printable version (Base 64 encoding). Note that this encoding preserves length information.
- Arrays are represented as a sequence of elements.

The following example shows an XML message that requests the execution of a process instance:

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
  <WfHeader>
    <ResponseRequired>Yes</ResponseRequired>
    <UserContext>This data is sent back in response</UserContext>
  </WfHeader>
  <ProcessTemplateExecute>
    <ProcTemplateName>OnlineCreditRequest</ProcTemplateName>
    <ProcInstName>Credit Request #658321</ProcInstName>
    <KeepName>true</KeepName>
    <ProcInstInputData>
      <CreditData>
        <Customer>
          <Name>User1</Name>
          <Account>4711</Account>
          <Account>0007</Account>
        </Customer>
        <Amount>100000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage>
```

Code page support

XML allows for the specification of messages in Unicode, as well as in ISO-defined character sets. XML messages sent to MQSeries Workflow are converted from their format as specified in the encoding keyword to the MQSeries Workflow code page as necessary. XML messages sent by MQSeries Workflow (responses and activity implementation invocation requests) are always encoded in Unicode.

Sending requests to MQSeries Workflow

The MQSeries Workflow message-based interface is used to request services from MQSeries Workflow. This is depicted in the following figure:

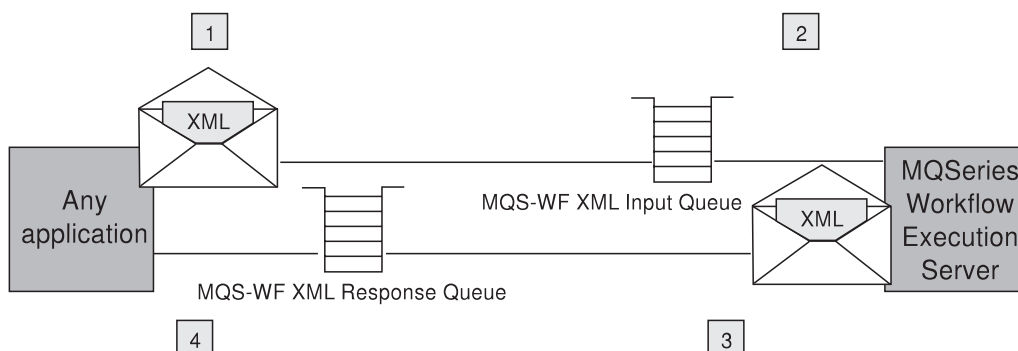


Figure 26. Sending requests to MQSeries Workflow

1. An application creates an MQSeries Workflow XML message and puts it into the MQSeries Workflow XML input queue.

2. The MQSeries Workflow execution server reads the XML message out of the XML input queue and processes the request.
3. The MQSeries Workflow execution server creates an XML message response and puts it into the response queue. The response queue information is part of the MQMD of the incoming XML message.
4. The application reads the incoming message and processes the response.

Supported functions

The following requests are supported by the XML message interface:

- “CreateAndStartInstance()” on page 434.
- “ExecuteProcessInstance()” on page 444.

XML input queue

The XML input queue

```
<prefix>.<SystemGroupName>.<SystemName>.EXE.XML
```

or

```
<prefix>.<SystemGroupName>.EXE.XML
```

is an MQSeries queue to which the MQSeries Workflow execution server is listening.

Only XML messages are accepted as input to this queue. The XML message must conform to the MQSeries Workflow XML message format. If it does not conform, a GeneralError XML message is put into the response queue.

Refer to *MQSeries Installation* for more information about the XML input queue.

Authentication and authorization

For authentication, MQSeries Workflow’s message-based interface relies on MQSeries. MQSeries Workflow does not perform any additional authentication. For setting up MQSeries security, refer to “Protecting MQSeries Objects” in *MQSeries System Administration*.

The user ID from the MQMD is used as the MQSeries Workflow user on whose behalf the request is to be performed. Authorization checks for that user are performed as usual.

Note: MQSeries user ID constraints differ from the ones defined for the MQSeries Workflow system. Since authorization is checked by MQSeries Workflow, the user ID in the MQMD of an XML message must be a valid MQSeries Workflow user. This has to be ensured by the application programmer and MQSeries Workflow administrator.

Invoking an activity implementation

Activity implementations are usually started by MQSeries Workflow by sending an internal invocation request message to a program execution agent or program execution server. They, in turn, invoke the program that was modeled to implement the activity. Using the message-based interface, it is also possible for MQSeries Workflow to send that invocation request message in XML format to a user-defined MQSeries queue.

From the point of view of MQ Workflow, the MQSeries application listening on that queue has to act like a program execution server. All the necessary

XML message interface

information is passed to invoke the activity implementation. The MQSeries application must return with an appropriate response, if requested by MQSeries Workflow.

Therefore, such an application is called a *user-defined program execution server* (UPES). A user-defined program execution server can be any application you write, provided it can deal with the MQSeries Workflow XML message format, or a program such as MQSeries Integrator V2.

MQSeries Workflow can send an invocation request in XML format to a user-defined program execution server if the program activity is modeled to be performed by that UPES. This is done in MQSeries Workflow Buildtime, using the activity property sheet.

Two invocation modes for the activity implementation can also be modeled:

- Synchronous invocation (the standard case), where MQSeries Workflow waits for a completion message containing result data from the UPES before the activity instance is considered to be complete.
- Asynchronous invocation, where no completion message is required and the activity instance is considered to be complete immediately after the invocation message has been sent. No result data can be returned in this case.

The following figure depicts the invocation of an activity implementation:

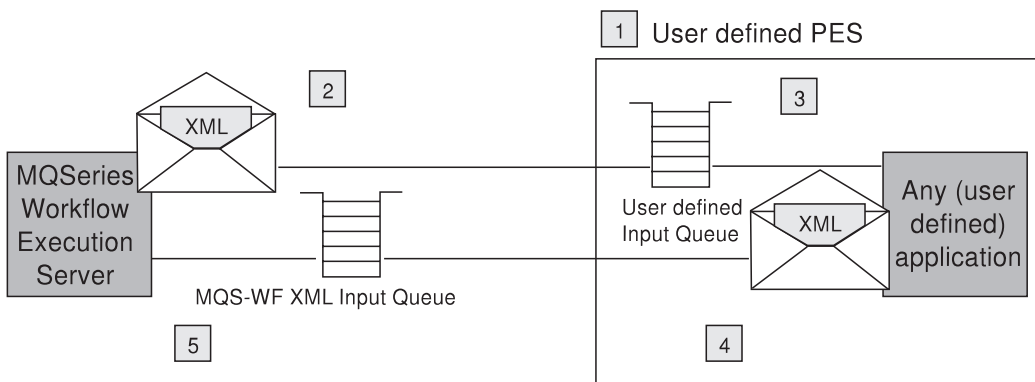


Figure 27. Starting an activity implementation via XML

1. A UPES must have been defined using MQSeries Workflow Buildtime.
2. When an activity implementation is to be started, the MQSeries Workflow execution server sends a program invocation message to the UPES.
3. An application listening to the UPES queue reads the XML message and performs the appropriate action. Possible actions are:
 - Transform the message into another format and route it to another recipient, for example, send an EDI message to another company.
 - Perform a transaction that involves the *get* of the request from the queue, the update of one or more DBMS or other resource managers, and the *put* of the response, in a single unit of work.
 - Invoke the specified activity implementation, for example, call a program on a platform not yet supported by MQSeries Workflow.
4. When the activity implementation has finished, the application creates a response MQSeries Workflow XML message, if required, and puts it into the

response queue. Note that the response queue information is part of the MQMD of the incoming XML invocation message.

5. The MQSeries Workflow execution server reads the response message, processes it, and changes the state of the activity accordingly.

User-defined program execution server (UPES)

A UPES is defined and configured for an MQSeries Workflow system by modeling it in MQSeries Workflow Buildtime. Essential attributes are the name and queue it represents. For more information, refer to the online help of MQSeries Workflow Buildtime.

The application that is listening to the UPES queue is not managed by MQSeries Workflow. A system administrator is responsible for administering the application. From an MQSeries Workflow point of view, the invocation of an activity implementation is successful when the invocation message is successfully put into the UPES queue.

Depending on the nature of the activity instance, the activity implementation may only need to be triggered and then runs asynchronously to the MQSeries Workflow process instance, or the process instance navigation has to be synchronized with its completion. In the latter case, a completion message has to be sent to MQSeries Workflow to inform it about the result of execution. In the former case, the activity instance is considered finished as soon as the invocation request is successfully sent. The indication of whether an implementation is to be started synchronously or asynchronously is modeled in Buildtime:

- Synchronous

The activity implementation is started and the activity instance put into state *Running*. When the activity implementation ends and the MQSeries Workflow execution server receives a completion message, the activity instance is set into the appropriate state, for example, *Finished*.

Correlation between the request and the response is done by means of the activity implementation correlation ID, which is passed in the invocation request by MQSeries Workflow, and must be passed back in the response

- Asynchronous

The activity implementation is started and the activity instance is put into the appropriate state, for example, *Finished*. No information on the completion of the activity implementation is expected. If a completion message is received, it is ignored.

Completion message

If the activity implementation is specified to run asynchronously, no completion message is expected. In that case, the successful *put* of the outgoing start activity implementation message is considered to be the complete invocation.

If the activity implementation is specified to run synchronously, a completion message is expected by the MQSeries Workflow execution server. This message can either report the successful execution of the activity implementation, passing the return code and output container, or it can report a failure passing the error and reason code. The error and reason codes must be understood by MQSeries Workflow. See *fmcmretc.h* or *fmcrs.cpy* for a list of valid codes.

XML message interface

Authorization

For invocation messages sent by MQSeries Workflow, the message header contains the user ID of the user on whose behalf the invocation is to be started. The UPES applications can use this information to implement their own authorization schemes.

Example

```
<ActivityImplInvoke>
  <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
  <Starter>user@systemGroup</Starter>
  <ProgramID>
    <ProcTempID>84848484FEFEFEFE</ProcTempID>
    <ProgramName>PerformOrder</ProgramName>
    <ImplementationData>
      <ImplementationPlatform>AIX</ImplementationPlatform>
      <ProgramParameters>/custNo=1234</ProgramParameters>
      <ExeOptions>
        <PathAndFileName>/usr/local/bin/perforder</PathAndFileName>
        <WorkingDirectoryName>/usr/local/data</WorkingDirectoryName>
        <InheritEnvironment>true</InheritEnvironment>
        <StartInForeground>true</StartInForeground>
        <AutomaticClose>true</AutomaticClose>
        <WindowStyleVisible>true</Visible>
        <RunInXTerm>true</RunInXTerm>
      </ExeOptions>
    </ImplementationData>
    <ImplementationData>
      <ImplementationPlatform>OS390</ImplementationPlatform>
      <ProgramParameters>/custNo=1234</ProgramParameters>
      <ExternalOptions>
        <ServiceName>CICS42</ServiceName>
        <ServiceType>CICS</ServiceType>
        <InvocationType>EXCI</InvocationType>
        <ExecutableName>ORDR</ExecutableName>
        <ExecutableType>REG1</ExecutableType>
        <IsLocalUser>true</IsLocalUser>
        <IsSecurityRoutineCall>true</IsSecurityRoutineCall>
        <CodePage>850</CodePage>
        <TimeoutPeriod>TimeInterval</TimeoutPeriod>
        <TimeoutInterval>60</TimeoutInterval>
        <IsMappingRoutineCall>>false</IsMappingRoutineCall>
      </ExternalOptions>
    </ImplementationData>
    <ProgramInputData>
      <!-- Another container goes here... -->
    </ProgramInputData>
    <ProgramOutputDataDefaults>
      <!-- And yet another container ... -->
    </ProgramOutputDataDefaults>
  </ActivityImplInvoke>

<ActivityImplInvokeResponse>
  <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
  <ProgramRC>0</ProgramRC>
  <ProgramOutputData>
    <!-- Another container comes here... -->
  </ProgramOutputData>
</ActivityImplInvokeResponse>
```

Figure 28. Sample activity implementation using XML

The MQSeries Workflow XML message format

The following XML syntax is a *document type definition* (DTD) used to describe the format of MQSeries Workflow messages. Note that the following format of a container only contains a suggestion, because the format can vary depending on your setup. Therefore, you cannot use this DTD description to validate your XML message without adding the appropriate specifications for the data structures you use.

You do not have to specify your containers. You are, however, encouraged to do so for future use or to validate them by any other XML application.

XML message interface

```
<!-- FmcXMLIF.dtd == DTD for MQSeries Workflow messages -->
<!-- Message ===== -->
<!ELEMENT WfMessage
  ( WfMessageHeader?,
    ( ProcessTemplateCreateAndStartInstance
      | ProcessTemplateCreateAndStartInstanceResponse
      | ProcessTemplateExecute
      | ProcessTemplateExecuteResponse
      | ActivityImplInvoke
      | ActivityImplInvokeResponse
      | GeneralError ) ) >
<!-- =====
      Workflow Message Header
===== -->
<!ELEMENT WfMessageHeader      (ResponseRequired?,UserContext?)>
<!-- Opaque -->
<!ELEMENT UserContext          (#PCDATA) >
<!-- Enumerated type -->
<!ELEMENT ResponseRequired     (#PCDATA)>
                                   <!-- Expected values: {No,IfError,Yes} -->

<!-- =====
      Specific Messages
===== -->
<!-- ProcessTemplateCreateAndStart ===== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
  ( ProcTemplName,
    ProcInstName,
    KeepName,
    ProcInstInputData ) >
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
  ( ProcessInstance
  | Exception ) >

<!-- ProcessTemplateExecute ===== -->
<!ELEMENT ProcessTemplateExecute
  ( ProcTemplName,
    ProcInstName,
    KeepName,
    ProcInstInputData ) >
<!ELEMENT ProcessTemplateExecuteResponse
  ( ( ProcessInstance,
    ProcInstOutputData )
  | Exception ) >

<!-- ActivityImplInvoke ===== -->
<!ELEMENT ActivityImplInvoke
  ( ActImplCorrelID,
    Starter,
    ProgramID,
    (ImplementationData)*,
    ProgramInputData,
    ProgramOutputDataDefaults ) >
<!ELEMENT ActivityImplInvokeResponse
  ( ActImplCorrelID,
    ( ProgramRC,
    ProgramOutputData )
```

Figure 29. Document type definition (DTD) for MQSeries Workflow XML messages (Part 1 of 5)

```

| Exception ) >

<!-- GeneralError ===== -->
<!ELEMENT GeneralError (Exception) >

<!-- =====
      Data Structures
      ===== -->
<!-- Named Entities ===== -->
<!ENTITY %CONTAINER "CreditData | InsuranceData | Address | Customer|...">
<!ELEMENT ProcInstInputData (%CONTAINER;) >
<!ELEMENT ProcInstOutputData (%CONTAINER;) >
<!ELEMENT ProgramInputData (%CONTAINER;) >
<!ELEMENT ProgramOutputData (%CONTAINER;) >
<!ELEMENT ProgramOutputDataDefaults (%CONTAINER;) >

<!-- Process Instance ===== -->
<!ELEMENT ProcessInstance
( ProcInstID,
  ProcInstName,
  ProcInstParentName?,
  ProcInstTopLevelName,
  ProcInstDescription?,
  ProcInstState,
  LastStateChangeTime,
  LastModificationTime,
  ProcTemplID,
  ProcTemplName,
  Icon,
  Category? ) >

<!-- Program ID ===== -->
<!ELEMENT ProgramID
( ProcTemplID,
  ProgramName ) >

<!-- Implementation Data ===== -->
<!ELEMENT ImplementationData
( ImplementationPlatform ,
  ProgramParameters,
  ( ExeOptions
    | DllOptions
    | ExternalOptions ) ) >
<!ELEMENT ExeOptions
( PathAndFileName,
  WorkingDirectoryName?,
  Environment?,
  InheritEnvironment,
  StartInForeground,
  AutomaticClose,
  WindowStyle?,
  RunInXTerm ) >

<!ELEMENT DllOptions
( PathAndFileName,

```

Figure 29. Document type definition (DTD) for MQSeries Workflow XML messages (Part 2 of 5)

XML message interface

```
        EntryPointName,
        ExecuteFenced,
        KeepLoaded )

<!ELEMENT ExternalOptions
  ( ServiceName,
    ServiceType,
    InvocationType,
    ExecutableName,
    ExecutableType,
    IsLocalUser,

    IsSecurityRoutineCall,
    CodePage,
    TimeoutPeriod,
    TimeoutInterval?,
    IsMappingRoutineCall,
    MappingType?,
    ForwardMappingFormat?,
    ForwardMappingParameters?,
    BackwardMappingFormat?,
    BackwardMappingParameters? ) >

<!-- Exception ===== -->
<!ELEMENT Exception
  (Rc, Parameters, MessageText?, Origin) >
<!-- Message text is optional, as it will be ignored
  in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
  (Parameter*) >

<!-- Data Elements ===== -->
<!-- Booleans -->
<!ELEMENT AutomaticClose      (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT DllV2Compatible    (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT ExecuteFenced      (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT InheritEnvironment  (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsLocalUser        (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsMappingRoutineCall (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsSecurityRoutineCall (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT KeepLoaded         (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT KeepName           (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT RunInXTerm         (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT StartInForeground   (#PCDATA) > <!-- Expected values: {true, false} -->

<!-- Strings -->
<!ELEMENT BackwardMappingFormat (#PCDATA) >
<!ELEMENT BackwardMappingParameters (#PCDATA) >
<!ELEMENT Category              (#PCDATA) >
<!ELEMENT EntryPointName        (#PCDATA) >
<!ELEMENT Environment           (#PCDATA) >
<!ELEMENT ExecutableName        (#PCDATA) >
<!ELEMENT ExecutableType        (#PCDATA) >
<!ELEMENT ForwardMappingFormat  (#PCDATA) >
<!ELEMENT ForwardMappingParameters (#PCDATA) >
<!ELEMENT Icon                  (#PCDATA) >
<!ELEMENT InvocationType        (#PCDATA) >
```

Figure 29. Document type definition (DTD) for MQSeries Workflow XML messages (Part 3 of 5)

```

<!ELEMENT MappingType                (#PCDATA) >
<!ELEMENT MessageText                (#PCDATA) >
<!ELEMENT Origin                     (#PCDATA) >
<!ELEMENT Parameter                  (#PCDATA) >
<!ELEMENT PathAndFileName            (#PCDATA) >
<!ELEMENT ProcInstDescription        (#PCDATA) >
<!ELEMENT ProcInstName                (#PCDATA) >
<!ELEMENT ProcInstParentName         (#PCDATA) >
<!ELEMENT ProcInstTopLevelName       (#PCDATA) >
<!ELEMENT ProcTemplateName           (#PCDATA) >
<!ELEMENT ProgramName                (#PCDATA) >
<!ELEMENT ProgramParameters          (#PCDATA) >
<!ELEMENT ServiceName                (#PCDATA) >
<!ELEMENT ServiceType                (#PCDATA) >
<!ELEMENT Starter                    (#PCDATA) >
<!ELEMENT WorkingDirectoryName        (#PCDATA) >

<!-- Opaque -->
<!ELEMENT ActImplCorrelID            (#PCDATA) >
<!ELEMENT ProcInstID                 (#PCDATA) >
<!ELEMENT ProcTemplID                (#PCDATA) >

<!-- Numbers -->
<!ELEMENT CodePage                   (#PCDATA) >
<!ELEMENT ProgramRC                  (#PCDATA) >
<!ELEMENT Rc                          (#PCDATA) >
<!ELEMENT TimeoutInterval            (#PCDATA) >

<!-- Timestamps YYYY-MM-DD-hh.mm.ss.000000 (000000 milliseconds) -->
<!ELEMENT LastModificationTime        (#PCDATA) >
<!ELEMENT LastStateChangeTime         (#PCDATA) >

<!-- Enumerated types -->
<!ELEMENT ImplementationPlatform (#PCDATA) > <!-- Expected values:
{ OS2,      AIX,
  HPUX,    Windows95,
  WindowsNT, OS390,
  Solaris } -->

<!ELEMENT ProcInstState                (#PCDATA) > <!-- Expected values:
{ Ready,    Running,
  Finished, Terminated,
  Suspended, Terminating,
  Suspending, Deleted } -->

<!ELEMENT WindowStyle                  (#PCDATA) > <!-- Expected values:
{ Visible,  Invisible,
  Minimized, Maximized } -->

<!ELEMENT TimeoutPeriod                (#PCDATA) > <!-- Expected values:
{ TimeInterval
  Forever   Never } -->

<!-- Container ===== -->

```

Figure 29. Document type definition (DTD) for MQSeries Workflow XML messages (Part 4 of 5)

XML message interface

```
<!ELEMENT CreditData    (...) >  
<!ELEMENT OrderData    (...) >  
<!ELEMENT InsuranceData (...) >  
<!ELEMENT Address      (...) >  
<!ELEMENT Customer     (...) >
```

Figure 29. Document type definition (DTD) for MQSeries Workflow XML messages (Part 5 of 5)

Chapter 3. Interfacing with the Program Execution Server

This chapter describes how to write exit routines to interface with the Program Execution Server to map data for legacy applications and to invoke programs.

CICS considerations

In CICS activity implementations, input/output container data is not retrieved from the PEA as in the LAN version but rather sent with the invocation and stored in the COMMAREA. If the user changes the COMMAREA without using the IBM MQSeries Workflow for OS/390 API or once the output container has been set, this data can no longer be retrieved.

For information on how to enable CICS to work with IBM MQSeries Workflow for OS/390, see *IBM MQSeries Workflow: Concepts and Architecture*.

IMS considerations

In IMS activity implementations, input/output container data is not retrieved from the PEA as in the LAN version but rather sent with the invocation and stored in the I/O AREA. If the user changes the I/O AREA without using the IBM MQSeries Workflow for OS/390 API, or once the output container has been set, this data can no longer be retrieved.

IMS programs can use only the Container API, which is a subset of the full MQSeries Workflow API. The Container API is defined in header files *fmcjcon.h* and *fmcjpcn.hxx* and COBOL copybook *fmcperfl.cpy*.

For information on how to enable IMS to work with IBM MQSeries Workflow for OS/390 see *MQSeries Workflow for OS/390: Customization and Administration*.

Notes:

1. An IMS program using the MQSeries Workflow for OS/390 API must issue *at least one* of the following calls:
 - FmcjContainerInContainer
 - FmcjContainerOutContainer
 - FmcjContainerRemoteInContainer
 - FmcjContainerRemoteOutContainer
2. An IMS program using the MQSeries Workflow for OS/390 API must issue *exactly one* of the following calls to return control to the Workflow system:
 - FmcjContainerSetOutContainer
 - FmcjContainerSetRemoteOutContainer

Program mapping via the Program Execution Server

Introduction

Whenever legacy applications are to be invoked by MQSeries Workflow, a mapping of the MQSeries container data into a format acceptable by the legacy application is needed.

Program mapping

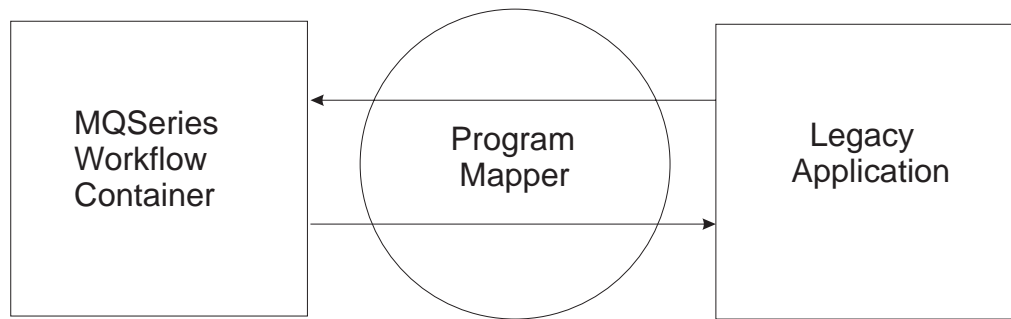


Figure 30. Program mapping illustration.

MQSeries Workflow offers a default program mapper with basic functionality. This section gives a brief overview of the program mapping component of the program execution server (PES). This component does the mapping of MQSeries Workflow containers into a format acceptable by legacy applications. This is a basic mapper so that legacy applications like IMS and CICS are supported. If more complicated mappings are to be done, other mapping tools can be used.

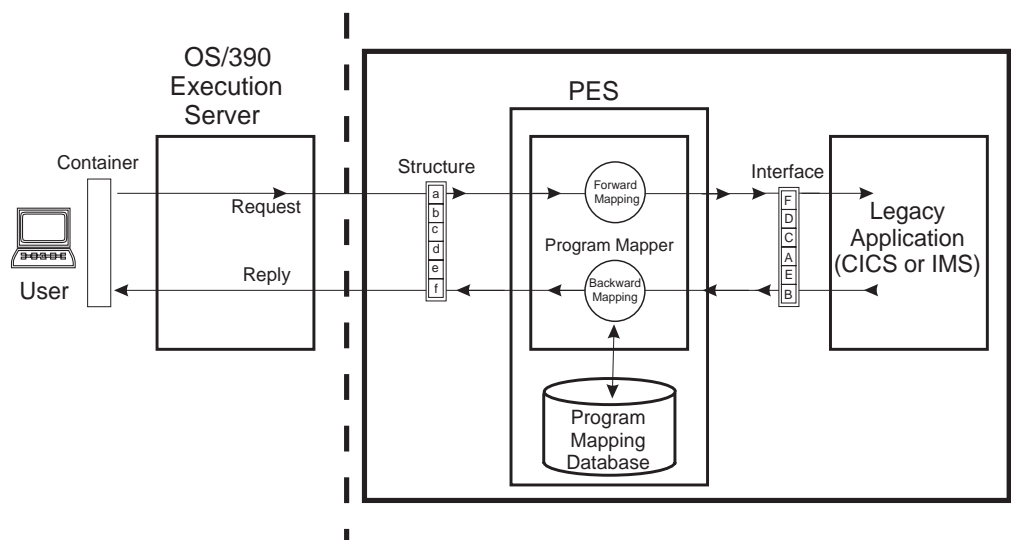


Figure 31. Program mapping control flow

In order to make the format of workflow containers acceptable to legacy applications, the content of the workflow containers is put to an interface called *structure*. The input/output interface for the legacy application is called the *interface*. The task for the program mapper is to convert the data between the structure and interface. Mapping from MQSeries Workflow to the legacy application (a to A, b to B...) is called *forward mapping*, and mapping from legacy applications to MQSeries Workflow (A to a, B to b...) *backward mapping*. If special conversion between structure and interface elements is needed, a usertype exit (which will be explained later) can be used.

Mapping is not necessary if the called application uses Workflow API calls to extract data from the containers.

The way the mapping should be done between structures and interfaces is defined with a **mapping definition language (MDL)**.

To create a mapping, you must first write the definition of the structure and interface elements. You then connect these structures and interfaces with forward/backward mapping definitions, compile the MDL with a parser, and load it into the mapping database. The elements of the mapper are explained in detail in the *MQSeries Workflow for OS/390: Customization and Administration*. The following graphic illustrates the process:



Figure 32. How to create a program mapping.

Program mapping definitions

In this section, the mapping definitions will be explained in more detail. For each definition, a simple example is also given.

Structure definition

A structure defines the MQSeries Workflow container structure that is passed into the program execution server (PES). The structure definition syntax is identical to the structure definition syntax used in the Flowmark definition language (FDL). This allows exporting container definitions from Buildtime into a flat file and copying these structure definitions into the mapping definition language (MDL). A structure mainly consists of a collection of members (structure elements) with a type and cardinality.

Example: This example shows a container in MQSeries Workflow representing an account representative structure. The structure contains the name of the holder of the account (first name and last name defined as a string), the corresponding ZIP (postal) code (defined as long), salary, and tax. The last part of the container is to be filled with the data of customers belonging to the holder of the account. The example for the definition of the CustomerStructure is given later. In order to define the structure you must define each element of the MQSeries Workflow container which is to be passed to the legacy application (the code for a sample legacy application is given in “Additional mapping examples” on page 192).

```

STRUCTURE AccountRepStructure
  LastName:  STRING;
  FirstName: STRING;
  Zip:      LONG;
  Salary:   FLOAT;
  Tax:      FLOAT;
  Customers: CustomerStructure(3);
END AccountRepStructure
  
```

You will find a more detailed example under “Simple data structure with default name mapping” on page 194 and the structure definition grammar under “Structure definition” on page 180.

Interface definition

An interface defines the layout and type of the data accepted by a legacy application. Each interface element has a fixed size and location (offset) and will be filled with converted container elements. There is no way to verify whether the size, location, and type of the elements actually match the size, location, and type expected by the legacy application. This means that the interface definitions must be created carefully. Otherwise, conversion results are unpredictable and runtime mapping errors can occur because of invalid data. Each element of an interface is

Program mapping

mapped to a structure element with the same name. If there is no element with the same name, the interface element is skipped and the container element is unaffected. It is also possible to define a constant for an interface element. See “Constants” on page 171 for more details.

Example: This example shows an interface of a legacy application representing an account representative structure. In this case the name of the holder of the account (first name and last name, defined as a string with a maximum of 50 characters, terminated by hex zero and left-justified with pad character " "), the corresponding ZIP (defined as an unsigned integer with 16 bits), salary, and tax (defined as float with 32 bits). The last part of the container is to be filled with the data of selected customers belonging to the holder of the account. The example for the definition of the CustomerStructure (array for 3 customers using another structure "CustomerInterfaceForCpp") is given later. In order to define the interface, you must define each individual element of the container used by the legacy application. The definition of the interface should be as follows:

```
INTERFACE AccountRepInterfaceForCpp
  LastName:  CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  FirstName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  Zip:       UNSIGNED INTEGER 16;
  Salary:    FLOAT 32;
  Tax:       FLOAT 32;
  Customers: ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
```

You will find a more detailed example under “Simple data structure with default name mapping” on page 194 and the interface definition grammar under “Interface definition” on page 181. If you have an interface element and no corresponding structure element, no mapping will be done by the default mapper. If it is required to have some constants on the legacy application side, each interface element can optionally have a *constant* statement that defines the constant to create for the legacy application. The constant is converted in forward mapping, whether there is a matching structure element or not. If backward mapping occurs, a structure element is set to this constant only if there is an element with the same name or a mapping rule between the structure and the interface with a constant. See “Constants” on page 171 for more information.

Forward/backward mapping definition

The connection between a structure and an interface is done via a forward and backward mapping definitions. Forward mapping is used to map a structure into a format accepted by a legacy application, and backward mapping is used to map legacy application data into a structure. Mapping done for structure and interface elements with identical names is called *default mapping*. In addition, it is possible, by using rules, to do *explicit mapping* of elements which have different names. Structures are mapped as a whole into interfaces, and arrays are mapped as a whole if both the structure and interface array have the same size. It is not possible to map array elements individually. If more powerful mappings are required, use container mapping (see *IBM MQSeries Workflow: Getting Started with Buildtime*).

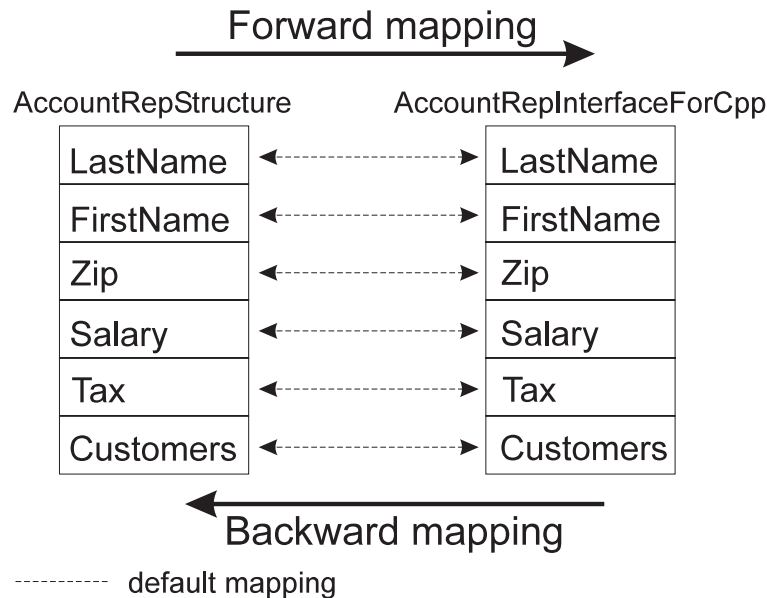


Figure 33. Default forward/backward mapping

To create a forward mapping, you must define which structure is to be mapped to which interface. To create a backward mapping, you must define which interface is to be mapped to which structure. Optionally, you can use rules to map elements with different names. See “Mapping algorithm” on page 168 for more detailed information about default and explicit mapping.

The coding for the mapping according to the diagram would therefore be:

```

/* Mapping from MQSeries Workflow (structure) to legacy appl.(interface) */
FORWARDMAPPING Forward
    FROM AccountRepStructure TO AccountRepInterfaceForCpp
END
/* Mapping from legacy appl. (interface) to MQSeries Workflow (structure) */
BACKWARDMAPPING Backward
    FROM AccountRepInterfaceForCpp TO AccountRepStructure
END
    
```

Note: All structure and interface elements are mapped because they have identical names (default mapping).

You will find a complete mapping in “Example” on page 172.

Usertype definition

A usertype can be used by the program mapper whenever the interface types provided by a default mapper do not offer the required conversion. In this case the actual data conversion must be done by a usertype exit, which must reside in a DLL. See “Usertype” on page 188.

Example: In order to assign a number to a currency with the corresponding symbol, you need to define a usertype (any mapping type would map the number to the exact number in a different format but not assign it to a special currency). It is also possible to define a usertype that calculates a value of currency A used in the structure to a currency B used in the interface (for example U.S. dollars to British pounds or the new European currency, the euro).

```

USERTYPE SampleUsertype LENGTH(4)
    DLL "SAMPUTY", "SampleUsertypeExit"
END
    
```

Program mapping

```
INTERFACE SampleUsertypeInterface
  DESCRIPTION "Sample Usertype Interface"
  SampleElement: USERTYPE SampleUsertype PARMS "$";
END
```

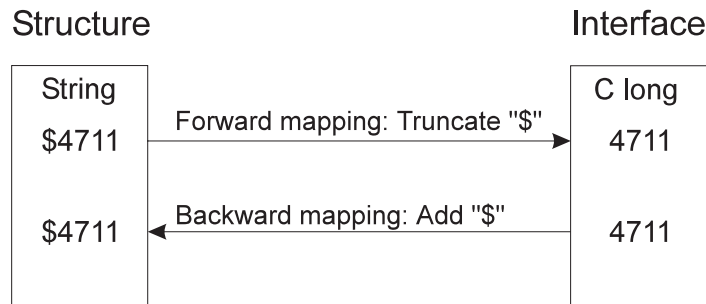


Figure 34. Usertype example

This example shows the functionality of a usertype. An interface element (defined as C long) is mapped to a structure element (defined as a string). In backward mapping, the C long "4711" is converted to a string and prefixed with "\$". In forward mapping the string "\$4711" is truncated to "4711" and then converted to a C long.

Mapping algorithm

All elements in an MQSeries Workflow container have names. The interface elements must also have names. Mapping is done per default on a name-by-name basis if elements have the same name. If element names are different, mapping rules can be used to do explicit mapping.

Structures are mapped as a unit to interfaces if their names are identical. Arrays are also mapped as a unit. It is also possible to define constants that are inserted on the legacy application or container side.

If structure or interface elements are not mapped, the data in the structure element or interface element is not modified.

Note: Each structure element can only be mapped to one interface element and vice versa.

In this example, there are 4 structure and interface elements which are to be mapped by the default mapper.

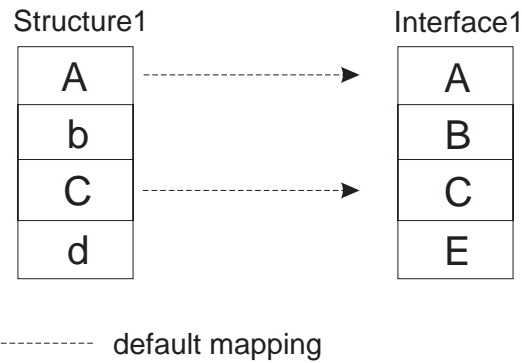


Figure 35. Default forward mapping illustration.

```
FORWARDMAPPING Forward
  FROM Structure1 TO Interface1
END
```

The default mapper maps structure element A to interface element A and structure element C to interface element C. It does *not* map structure element b to interface element B or structure element d to interface element E, because of the different names. See also “Simple data structure with default name mapping” on page 194 for a more detailed example.

If the corresponding structure and interface elements do not have identical names, the mapping must be defined explicitly. In this case you must define an additional mapping rule for the mapping. The mapping definition language (MDL) and the corresponding grammar are explained in “Grammar” on page 177. The following graphic displays a simple forward mapping with some non-identical names of the structure and interface elements.

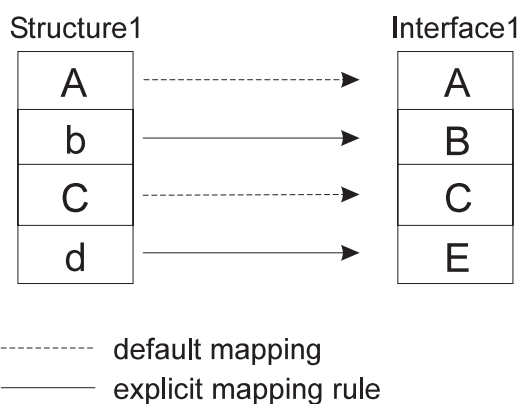


Figure 36. Forward2: Non-default forward mapping illustration.

The mapping rules would follow as:

```
FORWARDMAPPING Forward2 FROM Structure1 TO Interface1
  MAP b TO B;
  MAP d TO E;
END
```

Program mapping

Structure elements A and C do not have to be mapped to interface elements A and C explicitly; this will be done by the default mapper automatically. Refer to “Complex data structure with non-default name mapping” on page 195 for a more detailed example.

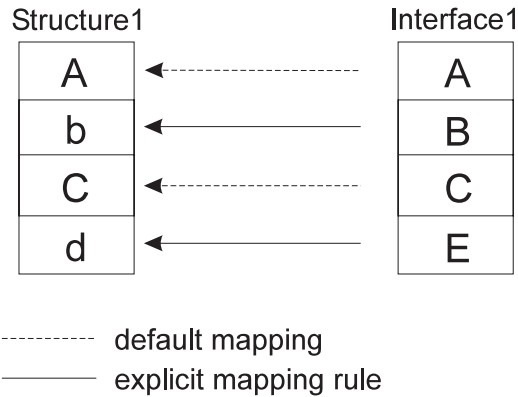


Figure 37. Non-default backward mapping Backward1 illustration.

If you would like a backward mapping according to the above diagram the mapping rules would be:

```
BACKWARDMAPPING Backward1 FROM Interface1 TO Structure1
    MAP B TO b;
    MAP E TO d;
END
```

How you map elements to each other only depends on the definition as long as you do not violate any conversion rules (see “Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types” on page 174). For example, it is not permissible to map an integer to a binary.

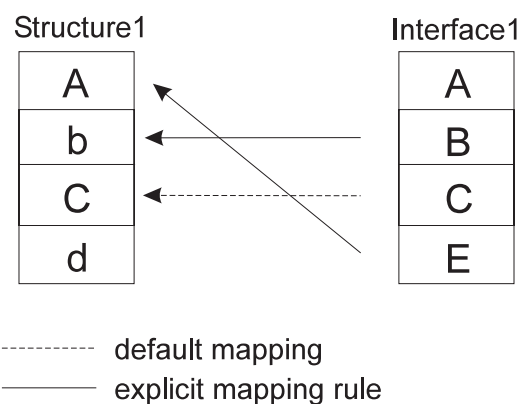


Figure 38. Backward2: Explicit mapping illustration.

In this case the interface element A will not be mapped to structure element A because there is a rule from interface element E to structure element A. Interface element B is mapped because of the explicit rule. Interface element C is mapped to structure element C because they have the same name (default mapping) and

because no explicit mapping for interface element C is defined. Interface element E will be mapped to structure element A because of a mapping rule for interface element E.

```
BACKWARDMAPPING Backward2 FROM Interface1 TO Structure1
    MAP B TO b;
    MAP E TO A;
END
```

Table 10. Rule mapping with no constant definition

| | BACKWARDMAPPING | FORWARDMAPPING |
|--|---|--|
| There is a definition rule for forward/backward mapping | Map interface element to structure element | Map structure element to interface element |
| There is <i>no</i> definition rule for forward/ backward mapping | Interface element not mapped to structure element | Interface element undefined |

Notes:

1. If the mapping rules use invalid or nonexistent interface element names, these rules are ignored during actual mapping. Make sure you use the right names in forward mapping and backward mapping. By contrast, structure element names used in definition rules must exist. Otherwise, runtime errors will occur.
2. Do not map structure elements to interface elements which are used in other arrays. The structure element will contain the interface elements with the largest dimension.

Constants

If it is required to have constants on the legacy application or structure side, each *interface* element can optionally have a constant statement that defines the constant to create for the legacy application or structure element. The constant is converted in forward mapping whether there is a matching structure element or not. If a backward mapping occurs, the structure element is set to this constant only if there is an element with the same name or a rule is defined for this structure element.

Table 11. Mapping with constant definition

| | Backward mapping | Forward mapping |
|--|--|-----------------------------------|
| There is a definition rule for forward/ backward mapping | Structure element set to constant | Interface element set to constant |
| There is <i>no</i> definition rule for forward/ backward mapping | Structure element not set to constant | Interface element set to constant |

Example for non-default forward mapping with constant definitions:

Program mapping

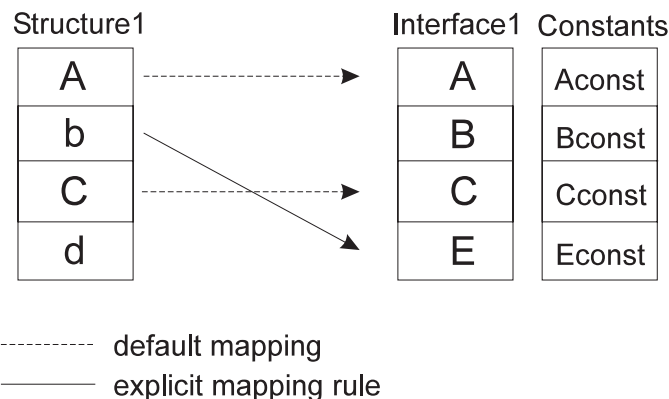


Figure 39. Forward mapping with constants.

Because structure elements A, C and b are mapped to interface elements A, C and E (which all have a constant definition), the interface elements A, C and E are set to Aconst, Cconst and Econst, respectively. The interface element B is set to Bconst because no structure element was mapped to B. So all interface elements are set to their corresponding constant values. In forward mapping, all interface elements with a constant definition are set to their constant whether there is a mapping to this element or not. Only if the interface element has *no* constant definition can a mapping change the value. Refer to “Simple data structure with all interface types with CONSTANTS and usertypes” on page 197 for a more complex example.

Example for default backward mapping with constant definitions:

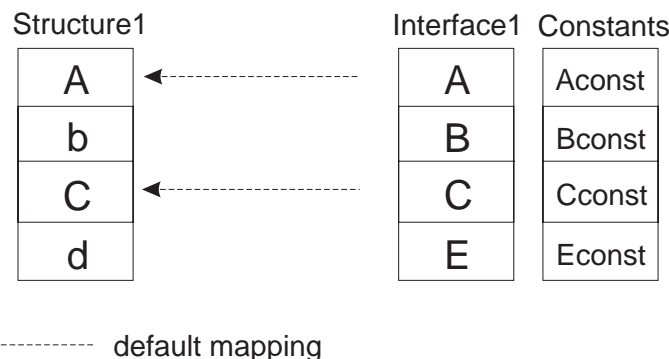


Figure 40. Backward mapping with constants.

Because interface elements B and E are not mapped to structure elements b and d, b and d are *not* set to the constant values of Bconst and Econst. Because interface element A is mapped to structure element A and interface element A has a constant Aconst, the structure element A is set to Aconst (same as structure element C is set to Cconst). Assuming there would be no constant definition for C on the legacy application side, interface element C would have been mapped to structure element C as usual.

Example

In this example, the structure elements and interface elements do not have the same names. Therefore they must be mapped explicitly in the forward/backward mapping definition. If they were not mapped explicitly, no mapping would be done at all, because all structure and interface elements have different names.


```

STRUCTURE AccountRepStructure
    LastName:  STRING;
    FirstName: STRING;
    Zip:       LONG;
    Salary:    FLOAT;
    Tax:       FLOAT;
    Customers: CustomerStructure(3);
END AccountRepStructure
INTERFACE AccountRepInterfaceForCpp
    L: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
    F: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
    Z: UNSIGNED INTEGER 16;
    S: FLOAT 32;
    T: FLOAT 32;
    C: ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
/* Mapping from MQSeries Workflow (structure) to legacy application (interface) */
FORWARDMAPPING Forward FROM AccountRepStructure TO AccountRepInterfaceForCpp
    MAP LastName TO L;
    MAP FirstName TO F;
    MAP Zip TO Z;
    MAP Salary TO S;
    MAP Tax TO T;
    MAP Customers TO C;
END
/* Mapping from legacy appl. (interface) to MQSeries Workflow (structure) */
BACKWARDMAPPING Backward FROM AccountRepInterfaceForCpp TO AccountRepStructure
    MAP L TO LastName;
    MAP F TO FirstName;
    MAP Z TO Zip;
    MAP S TO Salary;
    MAP T To Tax;
    MAP C TO Customers;
END

```

Supported program mapping definition element types

Program mapping structure definition element types

All MQSeries Workflow element types are supported:

- LONG
- FLOAT
- STRING
- BINARY

Program mapping interface definition element types

Characters: Characters have a size in bytes, an optional termination character (hex 0), and justification and padding.

Integer numbers: Integers are either signed or unsigned and have a size in bits. Supported sizes are 16 and 32 bits.

Float numbers: Floats have a size in bits. Supported sizes are 32 and 64 bits.

Packed numbers: Packed numbers have a size, are either signed, with a character for plus and a character for minus, or unsigned with an unsigned character, and have a scale.

Zoned numbers: Zoned numbers have a size, are either signed, with a character for plus and a character for minus, or unsigned with an unsigned character, and have a scale.

Program mapping

Interface: Interfaces have only a name and define another interface used as an interface element. In this way, it is possible to structure the interface in the same way structures can be defined to contain other structures.

Usertypes: Whenever the previous interface types do not match the required types, it is possible to define a usertype. For details see “Grammar” on page 177. Usertypes have a name and an optional parameter string, which can be used to pass additional information to the usertype exit. In order to use a usertype, it must be defined and a usertype DLL with a usertype exit must be provided. (See “Usertype” on page 188 for more details):

Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types: The following table lists all possible combinations of structure elements and interface elements. If they are arrays, they must have the same size. There is one exception: a character of size 1 can be mapped to an MQSeries Workflow LONG. If an invalid combination is used, a runtime error will be generated (see *MQSeries Workflow for OS/390: Messages and Codes* for detailed information).

Table 12. Mapping combinations

| Workflow Type Interface Type | String | Binary | Long | Float |
|---|--------|--------|------|-------|
| CHAR | * | * | * | |
| INTEGER | * | | * | |
| FLOAT | | | | * |
| PACKED | * | | * | * |
| ZONED | * | | * | * |
| USERTYPE | *1 | *1 | *1 | *1 |
| Note: ¹ Only available if this type of combination is supported by the <i>usertype</i> exit | | | | |

Table 13. C/C++ data type mappings (legacy application (C/C++) to FDL types (structure))

| C/C++ type | Interface | Structure | Comment |
|----------------|---|--------------|-------------------|
| char | CHAR(1) JUSTIFY LEFT PAD " " | STRING | No imbedded x'00' |
| char [5] | CHAR(5) JUSTIFY LEFT PAD " " or CHAR(5) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD " " | STRING | No imbedded x'00' |
| char | CHAR(1) JUSTIFY LEFT PAD " " | BINARY | |
| char [5] | CHAR(5) JUSTIFY LEFT PAD " " or CHAR(5) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD "<h00>" | BINARY | |
| char | CHAR(1) JUSTIFY LEFT PAD "<h00>" | LONG | |
| short | SIGNED INTEGER 16 | LONG, STRING | |
| unsigned short | UNSIGNED INTEGER 16 | LONG, STRING | |
| int | SIGNED INTEGER 32 | LONG, STRING | |
| unsigned int | UNSIGNED INTEGER 32 | LONG, STRING | |
| long | SIGNED INTEGER 32 | LONG, STRING | |
| unsigned long | UNSIGNED INTEGER 32 | LONG, STRING | |
| float | FLOAT 32 | FLOAT | |
| double | FLOAT 64 | FLOAT | |

Table 14. COBOL data type mappings (legacy application (COBOL) to FDL types (structure))

| COBOL | Interface | Structure | Comment |
|---|---|---------------------|-------------------------------------|
| PIC X(n) | CHAR(n) JUSTIFY LEFT PAD " " | STRING | No imbedded x'00' |
| PIC X(n) | CHAR(n) JUSTIFY LEFT PAD "<h00>" | BINARY | |
| PIC X(1) | CHAR(1) JUSTIFY LEFT PAD "<h00>" | LONG | |
| PIC S999 PACKED-DECIMAL or COMP-3 | PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE 0 | LONG, STRING, FLOAT | |
| PIC S999PP PACKED-DECIMAL or COMP-3 | PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE 2 | LONG, STRING, FLOAT | |
| PIC S99V9 PACKED-DECIMAL or COMP-3 | PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE -1 | LONG, STRING, FLOAT | |
| PIC S9999 BINARY or COMP-4 | SIGNED INTEGER 16 | LONG, STRING, FLOAT | Up to 4 digits ¹ |
| PIC S9(6) BINARY or COMP-4 | SIGNED INTEGER 32 | LONG, STRING, FLOAT | Between 5 and 9 digits ¹ |
| COMP-1 | FLOAT 32 | FLOAT | |
| COMP-2 | FLOAT 64 | FLOAT | |
| PIC S9(n)V9(m) DISPLAY | ZONED(n+m) SIGNED LAST MINUS "-" PLUS "+" SCALE -m | LONG, STRING, FLOAT | |
| PIC S9(n)V9(m) DISPLAY SIGN LEADING | ZONED(n+m) SIGNED FIRST MINUS "-" PLUS "+" SCALE -m | LONG, STRING, FLOAT | |
| PIC S9(n)V9(m) DISPLAY SIGN TRAILING SEPARATE | ZONED(n+m) SIGNED LAST MINUS "-" PLUS "+" SEPARATE SCALE -m | LONG, STRING, FLOAT | |
| PIC 9(n)V9(m) DISPLAY | ZONED(n) UNSIGNED SCALE -m | LONG, STRING, FLOAT | |

Note: ¹ 8-byte BINARY with 10-18 digits is not supported.

Grammar

This section describes the program mapper grammar. Base elements of the grammar are tokens, keywords in uppercase, constants and comments. By combining the base elements, you can define mapping elements: forward/backward mapping (FM/BM) and structure/interface definitions (IF, ST). The forward/backward mapping consists of rules (RL) which combine the structure and interface elements (IFE, STE). The following graphic is intended to illustrate the relationship between all these elements.

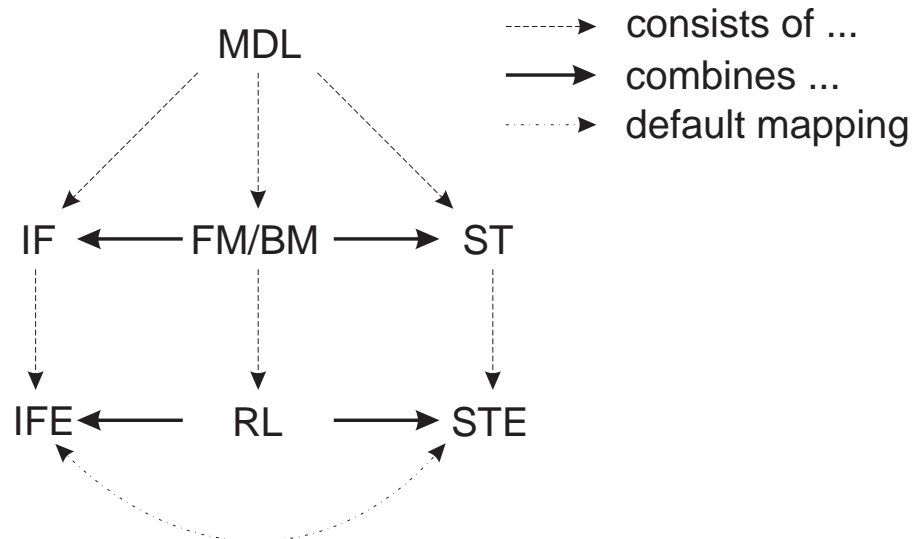


Figure 41. Relationship between mapping elements.

Grammar elements

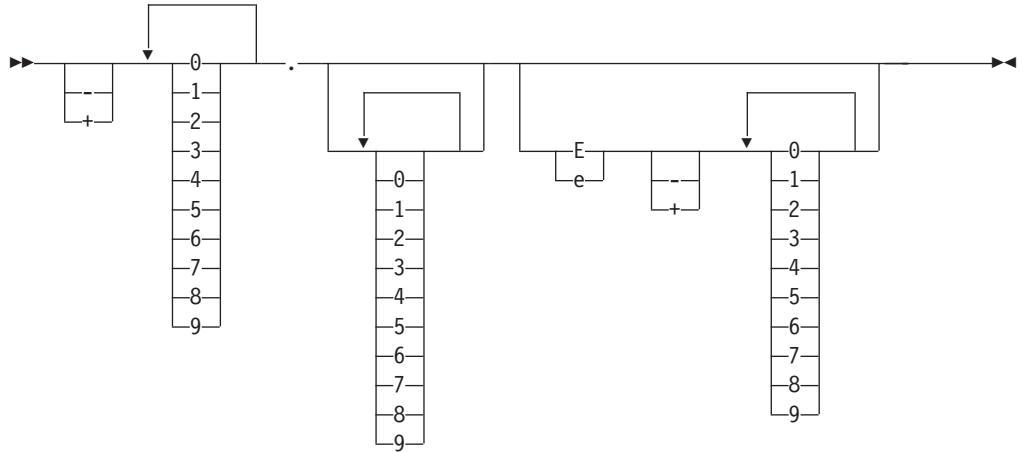
Comments: Comments should be used to document the mapping definition. There are two types of comments: C style, for example `/* 'comment' */` and C++ style, for example `// 'comment'` at the end of a line.

Comments starting with `/*` and ending with `*/` can be located anywhere between syntax tokens. Comments starting with `//` can be at the end of any line. Nesting of `/* ... */` is not allowed.

Tokens: Tokens are the base element of the grammar, and each token is therefore explained in detail. For each token, at least a syntax diagram and an example is given.

`FLOAT_TOKEN:`

Program mapping



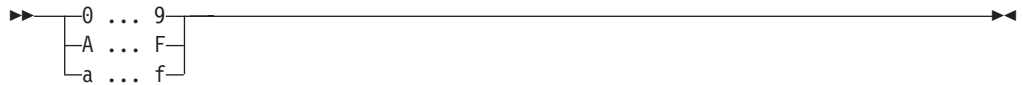
Example: -7.42E-4, which equals -0.000742.

Note: In order to keep the diagrams simple, the possibility of choosing a single number from 0 to 9 will be displayed with



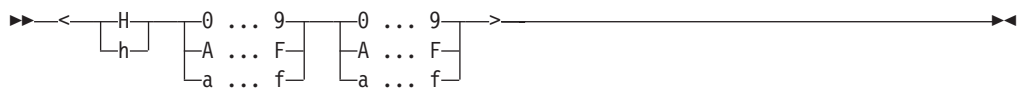
from now on.

Hex_digit:



Examples: 3, F.

Hex_token:



Example: <H4F>

IDENTIFIER:



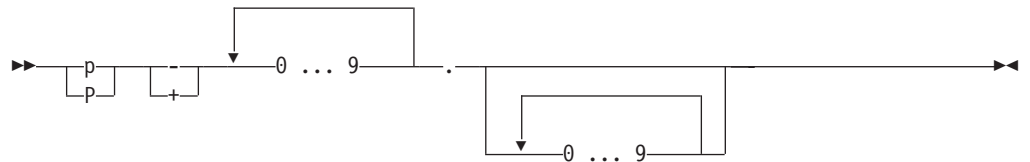
Examples: a_b4_h4, _Z97_bfsd

INT_TOKEN:



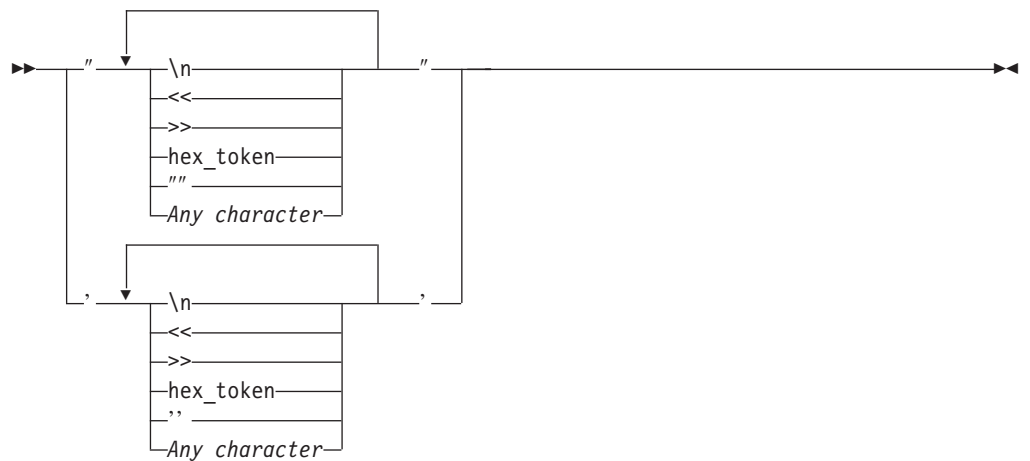
Examples: -89432, 412

PACKED_TOKEN:



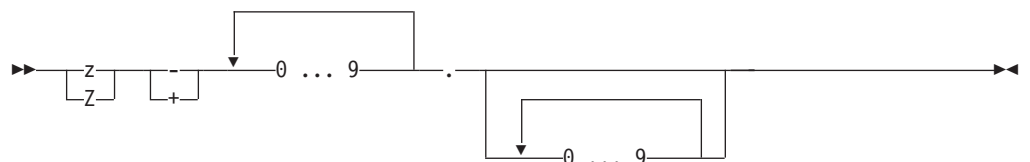
Examples: p+212.2 equals 212.2, p-142.8 equals -142.8

STRING_TOKEN:



Example: "AlbertEinstein", "xyz'a", 'xyz"a', 'other_example<h15><h12>'

ZONED_TOKEN:



Example: z+412.8

Program mapping

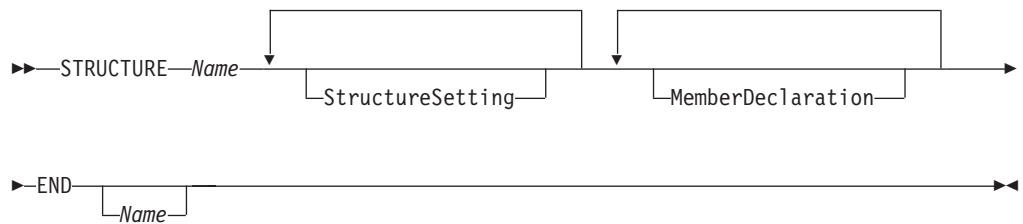
Keywords: Listed below are all available keywords. Do not name variables with these reserved keywords, because this would cause problems when mapping (for example, do not name a structure element 'STRUCTURE').

| | | | |
|----------|-----------------|--------------|----------------|
| ARRAY | BACKWARDMAPPING | BINARY | CHAR |
| CONSTANT | DESCRIPTION | DLL | DOCUMENTATION |
| END | FIRST | FLOAT | FORWARDMAPPING |
| FROM | IGNORE | INTEGER | INTERFACE |
| JUSTIFY | LAST | LEFT | LENGTH |
| LONG | MAP | MAPPING | MINUS |
| PACKED | PAD | PARMS | PLUS |
| RIGHT | SCALE | SEPARATE | SIGNED |
| STRING | STRUCTURE | TERMINATEDBY | TO |
| UNSIGNED | USERTYPE | ZONED | |

Note: All keywords must be in uppercase!

Structure definition:

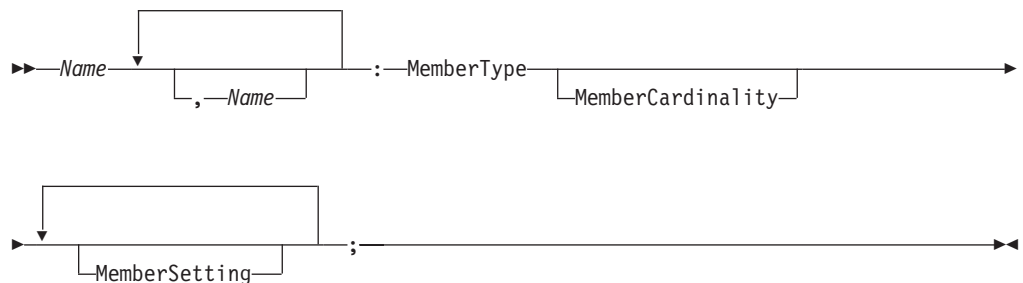
Structure:



StructureSetting:



MemberDeclaration:



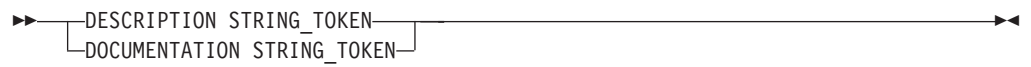
MemberType:



MemberCardinality:



MemberSetting:

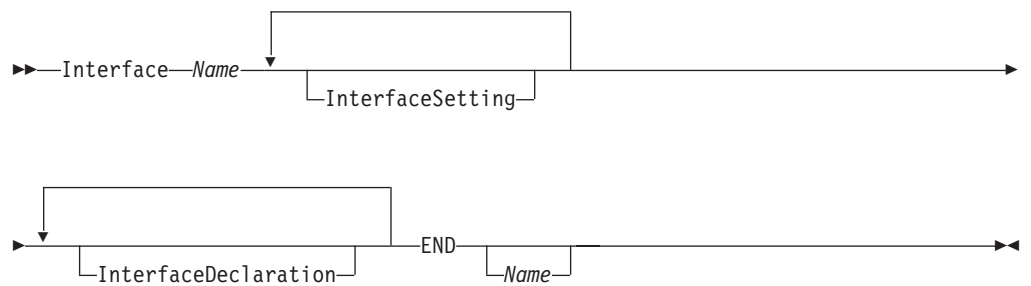


Note:

- Same syntax as structure definitions in FDL.

Interface definition:

Interface:



InterfaceSetting:



InterfaceDeclaration:



Program mapping

▶ MemberSetting ;

InterfaceType:

| |
|---|
| CharInterfaceType |
| CharInterfaceType CONSTANT STRING_TOKEN |
| IntegerInterfaceType |
| IntegerInterfaceType CONSTANT INT_TOKEN |
| FloatInterfaceType |
| FloatInterfaceType CONSTANT FLOAT_TOKEN |
| PackedInterfaceType |
| PackedInterfaceType CONSTANT PACKED_TOKEN |
| ZonedInterfaceType |
| ZonedInterfaceType CONSTANT ZONED_TOKEN |
| Name |
| UserInterfaceType |
| UserInterfaceType CONSTANT STRING_TOKEN |

InterfaceCardinality:

▶ ARRAY (INT_TOKEN)

Note:

- The sequence of elements is significant and defines the sequence of the elements in the data area used for forward and backward mapping. Each element has a fixed offset from the start of the data area. Make sure the interface elements have the size and type of the data the legacy application expects (See “Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types” on page 174 for size information).
- The sequence of member definitions in the structure is not relevant for the mapping. Mapping is done by name from interface elements to structure elements.

Interface types:

PackedInterfaceType:

▶ PACKED (INT_TOKEN) PackedAttributeList

PackedAttributeList:

| | |
|---|-----------------|
| SIGNED MINUS STRING_TOKEN PLUS STRING_TOKEN | SCALE INT_TOKEN |
| UNSigned STRING_TOKEN | |

Examples: p+212.2 equals 212.2, p-142.8 equals -142.8

Program mapping

Scale is used to define the decimal point of the packed number and the factor used by conversion. The packed number is multiplied by 10^{scale} in BACKWARDMAPPING and divided by 10^{scale} in FORWARDMAPPING. Of the plus character, minus character and unsigned character, only the lowermost 4 bits are used, which means that the value has to be $\leq x'0f'$.

Example:

Packed number is 4711, scale is 0. Decimal number is 4711.

Packed number is 4711, scale is 2. Decimal number is 471100.

Packed number is 4711, scale is -2. Decimal number is 47.11.

Format: PACKED(5) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE 1;

Byte0 Byte1 Byte2

DD DD DS

where D is a digit 0-9 and S is the positive or negative sign

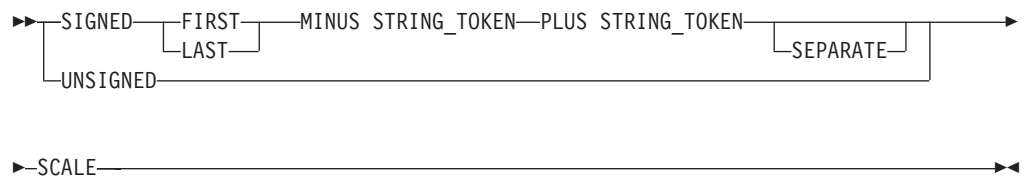
Note:

- The size in bytes used by the packed number is (packed number size + 1) / 2, rounded up to the next integer.
- Packed numbers can create runtime conversion errors if the digits are > 9 or the sign does not match the sign defined for the interface element.

ZonedInterfaceType:

►—ZONED—(INT_TOKEN)—PackedAttributeList—►

ZonedAttributeList:



Examples: z+471.1 equals 471.1, z-142.8 equals -142.8.

The size defines the number of significant digits used by the zoned number. Scale is used to define the decimal point of the zoned number and defines the factor used by conversion. The zoned number is multiplied by 10^{scale} in backward mapping and divided by 10^{scale} in forward mapping. FIRST and LAST define where the sign is located in the number. Of the plus character, minus character and unsigned character, only the lowermost 4 bits are used if the sign is not separate, which means the value has to be $\leq x'0f'$. If the sign is separate, all 8 bits of the first character are used, which means the character has to be $\leq x'ff'$. FIRST and LAST define the location of the sign (see examples below).

Example:

Zoned number is 4711, scale is 0. Decimal number is 4711.

Program mapping

Zoned number is 4711, scale is 2. Decimal number is 471100.

Zoned number is 4711, scale is -2. Decimal number is 47.11.

Format:

ZONED(3) SIGNED LAST LAST MINUS "<h0d>" PLUS "<h0c>" SCALE 2

Byte0 Byte1 Byte2

FD FD SD

where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits.

Format:

ZONED(3) SIGNED FIRST LAST MINUS "<h0d>" PLUS "<h0c>" SCALE 2

Byte0 Byte1 Byte2

SD FD FD

where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits.

Format:

ZONED(3) SIGNED LAST SEPARATE MINUS "-" PLUS "+" SCALE 2

Byte0 Byte1 Byte2 Byte3

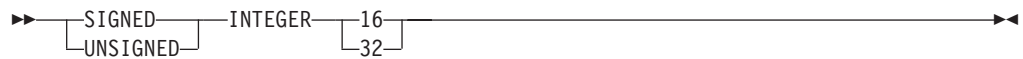
FD FD FD XX

where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits and XX is the sign (either x'4E' or x'60').

Note:

- The size in bytes used by the zoned number is the zoned number size. If the sign is separate, one additional byte is used.
- Zoned numbers can create runtime conversion errors if the digits are > 9 and the zone does not contain x'f' or the sign does not match the sign defined for the interface element.
- FIRST and LAST define where to append the sign.

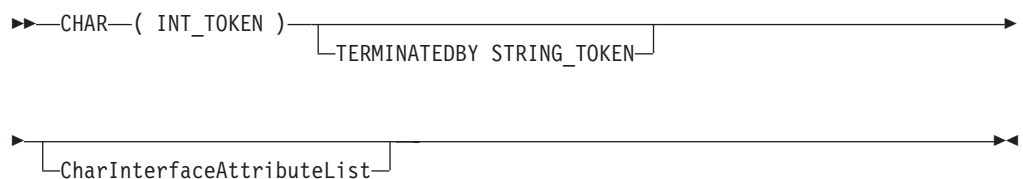
IntegerInterfaceType:



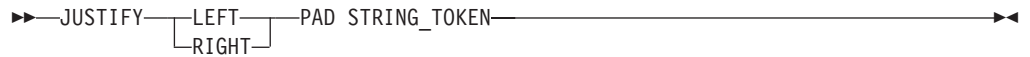
FloatInterfaceType:



CharacterInterfaceType:



CharInterfaceAttributeList:

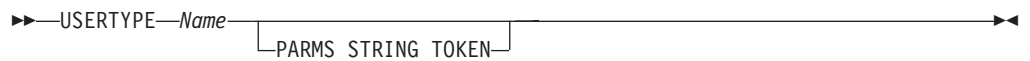


The termination character is inserted in forward mapping and stripped off in backward mapping. Padding, alignment and truncation occurs in forward mapping. The data is not modified in backward mapping. The character size in bytes must include the termination character and the number of bytes converted is one less than the specified character size.

Examples for justification:

| Length | Content | Length | JUSTIFY(Left) | JUSTIFY(RIGHT) |
|--------|---------|--------|---------------|----------------|
| 3 | 'ABC' | 4 | 'ABC ' | ' ABC' |
| 3 | 'AB ' | 4 | 'AB ' | ' AB ' |
| 3 | ' BC' | 4 | ' BC' | ' BC' |
| 4 | 'ABCD' | 4 | 'ABCD' | 'ABCD' |
| 4 | ' BCD' | 4 | ' BCD' | ' BCD' |
| 4 | 'ABC ' | 4 | 'ABC ' | 'ABC ' |
| 5 | 'ABCDE' | 4 | 'ABCD' | 'BCDE' |
| 5 | ' BCDE' | 4 | ' BCD' | 'BCDE' |
| 5 | 'ABCD ' | 4 | 'ABCD' | 'BCD ' |

UserInterfaceType:



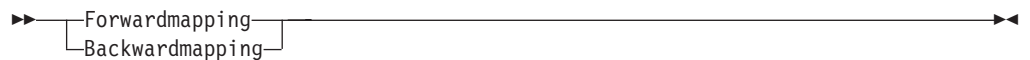
Note: STRING_TOKEN is passed to the usertype exit.

Mapping elements: This section illustrates the formal definition and grammar for the MDL. For each mapping element (structure, interface, forward/backward mapping etc.), there is a syntax diagram which explains how to use the elements correctly. Examples: "Example" on page 172ff and "MDL examples" on page 194ff.

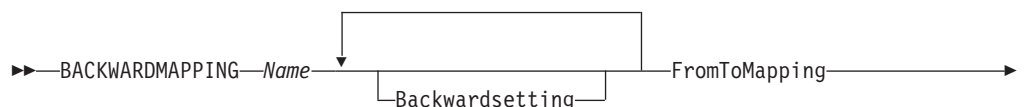
MappingElement:



Mapping:



Backward mapping:



Program mapping



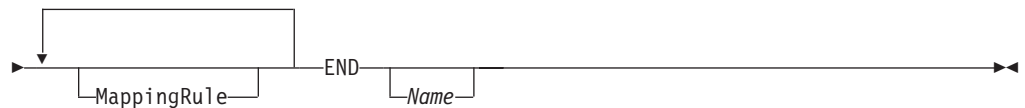
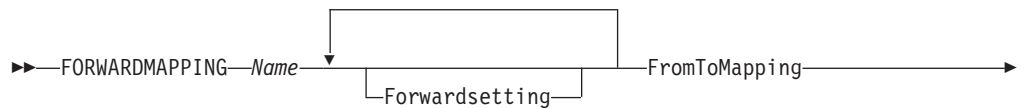
FromToMapping:



BackwardSetting:



Forward mapping:



ForwardSetting:



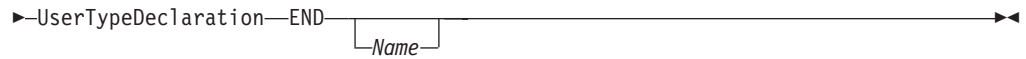
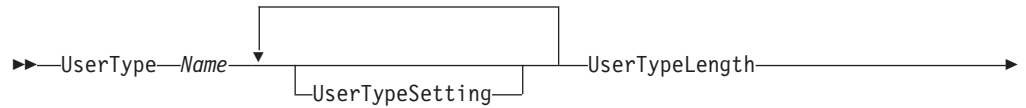
MappingRule:



Note: The first name must be the interface name in backward mapping and the structure element name in forward mapping. The second name must be the interface name in forward mapping and the structure element name in backward mapping (see “Mapping algorithm” on page 168)

Usertype definition:

UserType:

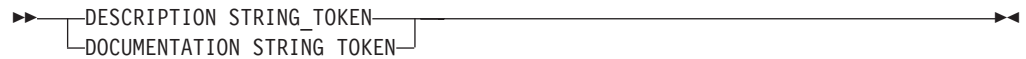


UserTypeLength:



Note: The usertype length defines the size of the usertype in bytes.

UserTypeSetting:



UserTypeDeclaration:



Note: The first string_token in the usertype declaration defines the DLL name, and the second defines the exit entry name. It is therefore possible to use one DLL for multiple usertypes.

Sample MDL for C/C++ and COBOL: In this example, each definition is shown in detail and the variables used have the same name (interface and structure). Therefore the forward and backward mapping definitions are as simple as possible and an explicit mapping as in the previous example is unnecessary.

```

/*
   --- Structure definition ---
*/
STRUCTURE AccountRepStructureBackw
  LastName:  STRING;
  FirstName: STRING;
  Zip:      LONG;
  Salary:   FLOAT;
  Tax:      FLOAT;
  Customers: CustomerStructure [3];
END AccountRepStructureBackw
STRUCTURE AccountRepStructureForw
  LastName:  STRING;
  FirstName: STRING;
  Zip:      LONG;
  Salary:   FLOAT;
  Tax:      FLOAT;
END AccountRepStructureForw
/* In this example the CustomerStructure contains 3 elements
   (last name, first name and telephone number which are defined as

```

Program mapping

```
string) */
STRUCTURE CustomerStructure
  LastName: STRING;
  FirstName: STRING;
  PhoneNumber: STRING;
END CustomerStructure
/*
  --- Interface definition for COBOL ---
*/
INTERFACE AccountRepInterfaceForCOBOL
  LastName: CHAR(50) JUSTIFY LEFT PAD ' ';
  FirstName: CHAR(50) JUSTIFY LEFT PAD ' ';
  Zip: UNSIGNED INTEGER 16;
  Salary: PACKED(8) UNSIGNED '<h0c>' SCALE -2;
  Tax: PACKED(2) UNSIGNED '<h0c>' SCALE -2;
  CustomersOpt: ARRAY(3) CustomerInterfaceForCOBOL;
END AccountRepInterfaceForCOBOL
INTERFACE CustomerInterfaceForCOBOL
  LastName: CHAR(50) JUSTIFY LEFT PAD ' ';
  FirstName: CHAR(50) JUSTIFY LEFT PAD ' ';
  PhoneNumber: CHAR(10) JUSTIFY LEFT PAD ' ';
END CustomerInterfaceForCOBOL
/*
  --- Interface definition for C++ ---
*/
INTERFACE AccountRepInterfaceForCpp
  LastName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  FirstName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  Zip: UNSIGNED INTEGER 16;
  Salary: FLOAT 32;
  Tax: FLOAT 32;
  Customers: ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
INTERFACE CustomerInterfaceForCpp
  LastName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  FirstName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
  PhoneNumber: CHAR(10) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
END CustomerInterfaceForCpp
/*
  -- Forward/backward mapping definition for COBOL ---
*/
FORWARDMAPPING ForwardSampleForCOBOL
  FROM AccountRepStructure TO AccountRepInterfaceForCOBOL
END ForwardSampleForCOBOL
BACKWARDMAPPING BackwardSampleForCOBOL
  FROM AccountRepInterfaceForCOBOL TO AccountRepStructure
END BackwardSampleForCOBOL
/*
  --- Forward/backward mapping definition for C++ ---
*/
FORWARDMAPPING ForwardSampleForCpp
  FROM AccountRepStructure TO AccountRepInterfaceForCpp
END ForwardSampleForCpp
BACKWARDMAPPING BackwardSampleForCpp
  FROM AccountRepInterfaceForCpp TO AccountRepStructure
END BackwardSampleForCpp
```

Note: This sample is distributed as FMCEMDL in SFMCDATA.

Usertype

A usertype allows converting MQSeries Workflow program mapping structure definition elements if the available interface types do not fulfill the required conversion. The program mapper will call a user exit each time a conversion for a usertype is required. It is possible to pass up to 256 characters to the user exit,

which must be defined where the interface element is mapped to the usertype. This allows using the same usertype for different conversions and controlling the functionality of the exit via passed parameters. In addition, it is possible to define parameters at the same time the forward and backward mapping formats are defined at buildtime. These are designated as 'forward mapping parameters' and 'backward mapping parameters', and the user exit has access to these parameters. Usertypes must have a fixed length.

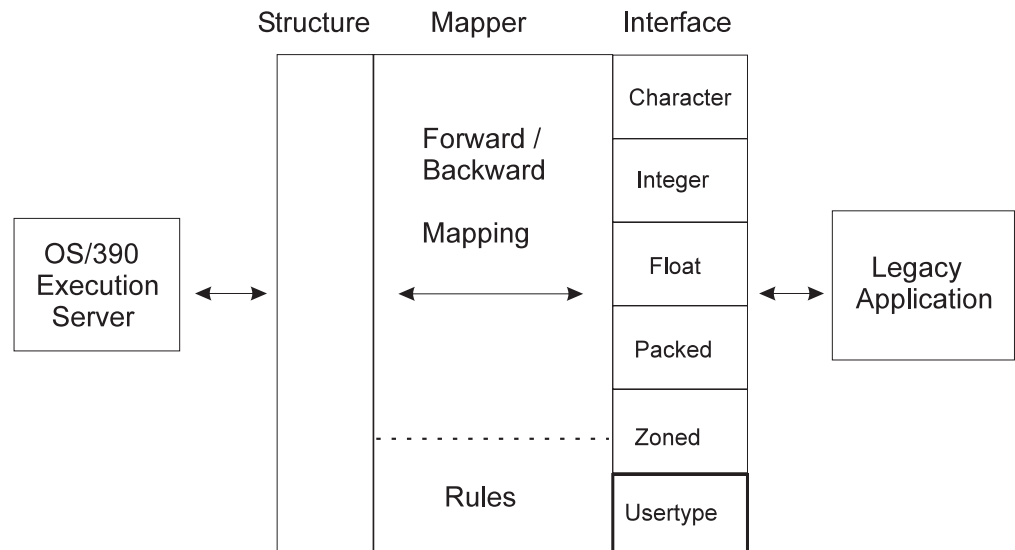


Figure 42. Usertype exit

Exit interface

The exit interface and data structures passed are defined in file `fmcxmeut.h`.

Note: The exit's entry point must have C linkage.

The following parameters are passed:

1. Direction of mapping required by PES

Use defined constants to check whether forward or backward mapping should be done.

```
#define FMC_PROGRAMMAPPING_USERTYPE_BACKWARDMAPPING 0
#define FMC_PROGRAMMAPPING_USERTYPE_FORWARDMAPPING 1
```

2. InterfaceDescriptor

Allows accessing the interface element data via a pointer to the data buffer. In addition the length of the usertype in bytes is passed.

```
typedef struct {
    char* elementData;           // pointer to raw data
    unsigned long elementDataLength; // length of usertype in bytes
} FmcProgrammMappingInterfaceDescriptor;
```

Warning: Make sure that the data written into the data buffer during forward mapping is not longer than the size of the usertype in bytes. Otherwise the results can be unpredictable.

3. StructureDescriptor

Program mapping

Allows accessing the structure element. The element name and length are passed in addition to a handle to the MQSeries Workflow container. The element can be accessed with the MQSeries Workflow Container API by passing the element name.

```
typedef struct {
    const char* elementName;
    //container element name (zero terminated)
    unsigned long elementNameLength;
    // container element name length in bytes
    FmcjContainerHandle containerHandle;
    // container handle
} FmcProgrammMappingStructureDescriptor;
```

The element name contains the qualified element name and can be used to get or set the container element with the container API. The name is zero terminated.

4. BuildTimeParameter, buildTimeParameterLength

At buildtime, it is possible to insert forward and backward mapping parameters. The length in bytes is passed via buildTimeParameterLength. The name is zero terminated.

5. InterfaceParameter, interfaceParameterLength

It is possible to insert a parameter in the interface where the usertype is used. The length in bytes is passed via interfaceParameterLength. The name is zero terminated.

Example:

```
INTERFACE SampleUsertypeInterface
    SampleElement: USERTYPE SampleUsertype PARMS "$";
END
```

6. Return value

A nonzero return value signals an error to the program mapper and the return code is set for the activity implementation.

For a sample, see sample usertype exit FMCHSMUT in SFMCDATA.

Creation of DLL

The usertype exit must be available at PES runtime. Any entry name and DLL name can be used, but these names must be identical to the names used in the usertype definition (See below). Sample JCL FMCHJMUT in SFMCDATA builds the sample usertype DLL. It is possible to have multiple usertype exits in one DLL if they use different function names.

Usertype definition

The usertype definition defines the name of a usertype and the length of the usertype in bytes. In addition the DLL name and exit function name have to be specified.

```
USERTYPE SampleUsertype LENGTH(4)
    DLL "SAMPUTY", "SampleUsertypeExit"
END
```

See FMCHEMUT in SMFCDATA for a sample usertype definition.

Size of program mapping interface definition elements

The interface definition must match exactly the layout of the data the legacy application expects. If there is a mismatch of even one byte, the results are unpredictable!

The following list summarizes which number of bytes are used by the interface definition element types.

Table 15. Interface element size

| Type | Length | Example |
|----------|--|--|
| Char | Size equals length in bytes. | Char(2) has a length of 2 bytes. |
| Integer | 2 bytes for INTEGER 16 and 4 bytes for INTEGER 32. | - |
| Float | 4 bytes for FLOAT 32 and 8 bytes for FLOAT 64. | - |
| Packed | Size used to define the (packed number + 1) divided by 2 and rounded up | Packed(2) equals 2 bytes, Packed(3) equals 2 bytes and Packed(4) equals 3 bytes. |
| Zoned | Size used to define the zoned number if a separate sign is not defined. Otherwise, it is one larger than the size used to define the zoned number. | Zoned(4) equals 4 bytes as Zoned(4) SEPERATE equals 5 bytes. |
| UserType | Size of usertype. | USERTYPE SampleUertype LENGTH(4) equals 4 bytes. |

Note: If there is any alignment done by the compiler used to compile the legacy application, this alignment also must be done in the interface definition.

Example: A C structure defined as follows:

```
struct S {
    int x;
    char y;
    int z;
};
```

might be aligned on 4-byte boundary so that

```
x x x x y      z z z z
0 1 2 3 4 5 6 7 8 9 10 11 Byte
```

and therefore needs following interface definition

```
INTERFACE i
    x: SIGNED INTEGER 32;
    y: CHAR(1) JUSTIFY LEFT PAD " ";
    pad: CHAR(3) JUSTIFY LEFT PAD "<h00>";
    z: SIGNED INTEGER 32;
END
```

Activation of program mapping definitions

The activation of a program mapping definition is described in detail in "Administering program mapping" in *MQSeries Workflow for OS/390: Customization and Administration*. The following contains a short summary about how to activate a program mapping definition:

1. Copy sample job to FMCHJMPR.
2. Create an MDL.
3. Update control statements for the input utility.
4. Run and compile MDL and insert MDL into mapping database.
5. If existing MDL elements were modified, restart the PES to activate the modifications. For new elements, no PES restart is needed.

Program mapping

Troubleshooting

In order to provide as much help to you as possible, this section lists usual problems and typical solutions. For a detailed list of error messages refer to *MQSeries Workflow for OS/390: Messages and Codes*.

Common errors

Element data mapped is incorrect: Most commonly, this is due to a mismatch between the legacy application data layout and the interface definition.

There is no way for the program mapper to check whether the interface maps correctly to the data format and layout the legacy application expects. Each interface should be carefully created and double checked. Runtime conversion errors (for packed and zoned interface types), are usually caused by this. In addition, reflect alignment on the legacy side in the interface (see “Size of program mapping interface definition elements” on page 190 for more details). If the size of one interface element is incorrect (for example integer 16 instead of integer 32) all the following data will be incorrect.

Elements not mapped: Either the element names are different and no mapping rule was specified or a mapping rule uses the wrong element names.

Note: If a mapping rule is used in both directions, the mapping rule arguments have to be switched.

Modified mapping definition is not activated: Mapping definitions are reloaded whenever the PES is restarted. It is not sufficient to import the definition into the program mapping database. New definitions will be used without a PES restart.

Additional mapping examples

Application examples

CICS C++ Application: This C++ application under CICS displays the data, creates new customers and increases the salary by 8 percent. It corresponds to the forward/backward mapping example in “Forward/backward mapping definition” on page 166.

```
#pragma XOPTS(SP)
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
/* --- CustomerStructure --- */
#pragma pack(1)
struct CustomerStructure {
    char LastName[50];
    char FirstName[50];
    char PhoneNumber[10];
};
/* --- AccountRepStructure --- */
struct AccountRepStructure {
    char LastName[50];
    char FirstName[50];
    short Zip;
    float Salary;
    float Tax;
    struct CustomerStructure Customers[3];
};
#pragma pack(1)
int main()
```

```

{
    struct AccountRepStructure *commarea;
    EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
    if (dfheiptr->eibcalen <= 0) {
        cout << "??? Empty commarea ???" << endl;
        EXEC CICS RETURN;
    }
    // Display all data
    cout << "LastName:  " << commarea->LastName << endl;
    cout << "FirstName: " << commarea->FirstName << endl;
    cout << "Zip:       " << commarea->Zip << endl;
    cout << "Salary:    " << commarea->Salary << endl;
    cout << "Tax:      " << commarea->Tax << endl;
    // Create customers
    strcpy(commarea->Customers[0].LastName,"EINSTEIN");
    strcpy(commarea->Customers[0].FirstName,"ALBERT");
    strcpy(commarea->Customers[0].PhoneNumber,"3048");
    strcpy(commarea->Customers[1].LastName,"NEWTON");
    strcpy(commarea->Customers[1].FirstName,"ISAAC");
    strcpy(commarea->Customers[1].PhoneNumber,"4041");
    strcpy(commarea->Customers[2].LastName,"HAWKING");
    strcpy(commarea->Customers[2].FirstName,"STEVEN");
    strcpy(commarea->Customers[2].PhoneNumber,"5154");
    for (int i=0; i<3; i++) {
        cout << "Customer LastName : "
            << commarea->Customers[i].LastName << endl;
        cout << "Customer FirstName : "
            << commarea->Customers[i].FirstName << endl;
        cout << "Customer PhoneNumber : "
            << commarea->Customers[i].PhoneNumber << endl;
    }
    // Increase salary by 8%
    commarea->Salary *= 1.08;
    cout << "New Salary: " << commarea->Salary << endl;
    EXEC CICS RETURN;
}

```

CICS COBOL Application: This COBOL application under CICS does the same thing as “CICS C++ Application” on page 192 (displays the data, creates new customers and increases the salary by 8 percent). It corresponds to the forward/backward mapping example in “Forward/backward mapping definition” on page 166.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "SAMPCBL".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 PRINT-SALARY PIC Z(5)9.9(2).
01 PRINT-TAX PIC Z9.99.
LINKAGE SECTION.
01 DFHCOMMAREA.

*
*   AccountRepStructure
*
02 LASTNAME PIC X(50).
02 FIRSTNAME PIC X(50).
02 ZIP PIC 9999 COMP-4.
02 SALARY PIC 9(6)V9(2) COMP-3.
02 TAX PIC V99 COMP-3.

*
*   CustomerStructure
*
02 CUSTOMERS OCCURS 3 TIMES
    INDEXED BY CUSTOMER-INDEX.
03 LAST-NAME PIC X(50).
03 FIRST-NAME PIC X(50).
03 PHONE-NUMBER PIC X(10).

```

Program mapping

```
PROCEDURE DIVISION.  
  IF EIBCALEN <= 0  
    DISPLAY "???" EMPTY COMMAREA ???"  
    EXEC CICS RETURN  
    END-EXEC  
  END-IF  
  
  *  
  *   Display all data  
  *  
    DISPLAY "Lastname: " LASTNAME  
    DISPLAY "Firstname: " FIRSTNAME  
    DISPLAY "Zip:      " ZIP  
    MOVE TAX TO PRINT-TAX  
    MOVE SALARY TO PRINT-SALARY  
    DISPLAY "Salary:   " PRINT-SALARY  
    DISPLAY "Tax:      " PRINT-TAX  
  
  *  
  *   Create some customers  
  *  
    MOVE "EINSTEIN" TO LAST-NAME(1)  
    MOVE "ALBERT" TO FIRST-NAME(1)  
    MOVE "3048" TO PHONE-NUMBER(1)  
    MOVE "NEWTON" TO LAST-NAME(2)  
    MOVE "ISAAC" TO FIRST-NAME(2)  
    MOVE "4041" TO PHONE-NUMBER(2)  
    MOVE "HAWKING" TO LAST-NAME(3)  
    MOVE "STEPHEN" TO FIRST-NAME(3)  
    MOVE "5154" TO PHONE-NUMBER(3)  
    PERFORM  
      VARYING CUSTOMER-INDEX FROM 1 BY 1  
        UNTIL CUSTOMER-INDEX > 3  
        DISPLAY "Customer LastName : "  
          LAST-NAME(CUSTOMER-INDEX)  
        DISPLAY "Customer FirstName: "  
          FIRST-NAME(CUSTOMER-INDEX)  
        DISPLAY "Customer PhoneNumber: "  
          PHONE-NUMBER(CUSTOMER-INDEX)  
    END-PERFORM  
  
  *   Increase salary by 8%  
    COMPUTE SALARY = SALARY * 1.08  
    MOVE SALARY TO PRINT-SALARY  
    DISPLAY "New Salary: " PRINT-SALARY  
    EXEC CICS RETURN  
    END-EXEC  
  GOBACK.
```

MDL examples

This section illustrates some examples of how to use the mapper. There are examples coded in C, COBOL and for simple and complex data structures.

Simple data structure with default name mapping: In this example the mapping is defined for a simple data structure and the mapping used is the default one (which means that each element of a container is mapped to the element with the **same name** in the other container).

```
STRUCTURE SimpleDataStructure  
  element1: STRING;  
  element2: STRING;  
  element3: LONG;  
  element4: FLOAT;  
  element5: BINARY;  
  element6: BINARY;  
  element7: LONG(20);  
END SimpleDataStructure  
INTERFACE SimpleDataInterface  
  DESCRIPTION 'This is an example of a simple interface mapping'
```

```

element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
element2: CHAR(20) JUSTIFY LEFT PAD "<h00>";
element3: SIGNED INTEGER 16;
element4: FLOAT IBM 32;
element5: CHAR(500);
element6: CHAR(200);
element7: ARRAY (10) SIGNED INTEGER 8;
END SimpleDataInterface
BACKWARDMAPPING SimpleMapping
FROM SimpleDataInterface
TO SimpleDataStructure
END SimpleMapping
FORWARDMAPPING SimpleMapping
FROM SimpleDataStructure
TO SimpleDataInterface
END SimpleMapping

```

Complex data structure with default name mapping: In this example, the mapping is defined for a complex data structure and the mapping used is the default one (which means that each element of a container is mapped to the element with the **same name** in the other container).

```

STRUCTURE ComplexDataStructure1
  element1: STRING (10);
  element2: FLOAT;
END ComplexDataStructure1
STRUCTURE ComplexDataStructure2
  element1: STRING (20);
  element2: ComplexDataStructure (5);
END ComplexDataStructure2
INTERFACE ComplexDataInterface1
  element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
  element2: FLOAT IBM 32";
END ComplexDataInterface1
INTERFACE ComplexDataInterface2
  element1: CHAR(20) JUSTIFY RIGHT PAD ' ';
  element2: ARRAY(5) ComplexDataInterface1;
END ComplexDataInterface2
BACKWARDMAPPING ComplexMapping
FROM ComplexDataInterface2
TO ComplexDataStructure2
/*
  * Element1 and element2 are mapped implicitly
  */
END ComplexMapping
FORWARDMAPPING ComplexMapping
FROM ComplexDataStructure2
TO ComplexDataInterface2
/*
  * element1 and element2 are mapped implicitly
  */
END ComplexMapping

```

Complex data structure with non-default name mapping: In this example the mapping is defined for a complex data structure and the mapping used is not the default one (which means that the structure elements of ComplexDataStructure1 do not have identical names in the interface ComplexStructure1 and are mapped explicitly).

```

STRUCTURE ComplexDataStructure1
  str: STRING (10);
  flt: FLOAT;
END ComplexDataStructure1
INTERFACE ComplexDataInterface1
  stri: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
  flti: FLOAT IBM 32;

```

Program mapping

```
        END ComplexDataInterface1
BACKWARDMAPPING ComplexMapping
    FROM ComplexDataInterface1
    TO ComplexDataStructure1
    MAP stri TO str;
    MAP flti TO flt;
    END ComplexMapping
FORWARDMAPPING ComplexMapping
    FROM ComplexDataStructure1
    TO ComplexDataInterface1
    MAP str TO stri;
    MAP flt TO flti;
    END ComplexMapping
```

Complex data structure with non-default name mapping with arrays and structures: In this example, the mapping is defined for a complex data structure and the mapping used is not the default one (which means that the elements (in this case arrays) of structure ComplexDataStructure1 do not have identical names in the Interface ComplexStructure1 and are mapped explicitly).

```
STRUCTURE ComplexDataStructure1
    element1: STRING (10);
    element2: FLOAT;
    END ComplexDataStructure1
STRUCTURE ComplexDataStructure2
    element1: STRING (20);
    element2: ComplexDataStructure1 (5);
    special: FLOAT;
    END ComplexDataStructure2
STRUCTURE ComplexDataStructure3
    element1: STRING (5);
    element2: ComplexDataStructure2 (4);
    element3: ComplexDataStructure1 (4);
    END ComplexDataStructure3
INTERFACE ComplexDataInterface1
    element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
    element2: FLOAT IBM 32;
    END ComplexDataInterface1
INTERFACE ComplexDataInterface2
    element1: CHAR(20) TERMINATEDBY "<h00>" JUSTIFY RIGHT PAD '*';
    element2: ARRAY (5) ComplexDataInterface1;
    special: FLOAT IBM 32
    END ComplexDataInterface2
INTERFACE ComplexDataInterface3
    element1: CHAR(5) JUSTIFY RIGHT PAD '*';
    element2: ARRAY (4) ComplexDataInterface1;
    elementx: ARRAY (4) ComplexDataInterface2;
    END ComplexDataInterface3
BACKWARDMAPPING ComplexMapping
    FROM ComplexDataInterface3
    TO ComplexDataStructure3
    /* Interface element elementx is explicitly mapped to structure
    element element2. All structure and interface elements of this
    structure are mapped per default; interface element element2 is
    also explicitly mapped with all its subelements. */
    MAP 'elementx' to 'element2';
    MAP 'element2' to 'element3';
    /* element special is not mapped per default and an explicit rule
    for this element is required */
    MAP 'elementx.special' to 'element2.special';
    /* Per default mapping element1 with all subelements is mapped */
    END ComplexMapping
FORWARDMAPPING ComplexMapping
    FROM ComplexDataStructure1
    TO ComplexDataInterface1
    /* Structure element element2 is explicitly mapped to interface
    element elementx. All structure and interface elements of this
```



```

        structure are mapped per default; structure element element3 is
        also explicitly mapped with all its subelements. */
MAP 'element2' to 'elementx';
MAP 'element3' to 'element2';
/* element specials is not mapped per default and an explicit rule
   for this element is required */
MAP 'element2.specials' to 'elementx.speciali';
/* Per default mapping element1 with all subelements is mapped */
END ComplexMapping

```

Simple data structure with all interface types with CONSTANTS and usertypes:

In this example the mapping is defined for a simple data structure and the mapping used is not the default one (which means that the elements of structure ComplexDataStructure1 do not have identical names in the interface ComplexStructure1 and are mapped explicitly). Additionally, there is a usertype defined, which converts a 4 byte integer into a Workflow string, separates every three digits by a comma and prefixes the string with a currency symbol, for example \$1,234,567.

```

/* A usertype which converts a 4 byte integer into a Workflow string, separates
 * every three digits by a comma and prefixes the string with a currency
 * symbol, for example $1,234,567 */
USERTYPE user1 LENGTH(4)
        DLL "dlluser","user2Inbound"
        END user1
STRUCTURE SimpleDataStructure
        element1: LONG;
        element2: STRING;
        element3: LONG;
        element7: LONG(20);
        element8: STRING;
        END SimpleDataStructure
INTERFACE SimpleDataInterface
        DESCRIPTION 'This is an example of a simple interface mapping'
        /* The following integer constant is inserted in forward mapping
         * and removed in backward mapping */
        insert: SIGNED INTEGER 16
                CONSTANT 4711
        element3: SIGNED INTEGER 16;
        element7: ARRAY (100) SIGNED INTEGER 8;
        element1: USERTYPE user1 PARS 'DM'; /* three digits */
        element8: USERTYPE user1 PARS '$'; /* for example $12,345 */
        element9: CHAR(5) CONSTANT "This is a string constant with some
                hex chars <h47><h11>" JUSTIFY RIGHT PAD '*'
        element10: PACKED (10) CONSTANT p47.11
                SIGNED FIRST MINUS "<h0d>" PLUS "<h0c>" UNSIGNED "<h0c>"
                SCALE 5
        element11: ZONED (10) CONSTANT z47.11
                SIGNED FIRST MINUS "<h0d>" PLUS "<h0c>" SCALE 5
        element12: FLOAT IBM 8
                CONSTANT +47E11
        END SimpleDataInterface
BACKWARDMAPPING SimpleMapping
        FROM SimpleDataInterface
        TO SimpleDataStructure
        END SimpleMapping
FORWARDMAPPING SimpleMapping
        FROM SimpleDataStructure
        TO SimpleDataInterface
        END SimpleMapping

```

Program execution server exits

Introduction

For extensibility of MQSeries Workflow for OS/390, the PES uses exits in the following areas:

- Application invocation
- Legacy application program mapping

Exits provided by MQSeries Workflow for OS/390 for program mapping, program invocation, and journaling. Whenever new exits are needed, these exits can be used instead of the IBM supplied exits or in parallel.

The exits have an interface defined to which every user written exit must conform.

Each exit type must provide an Init() function, which is called from the PES when the exit is needed the first time. Later, the exit-specific functions are called (see the specific exit descriptions for more details). A shutdown request for the PES triggers a call to the Deinit() function of the exits.

Usually the Init() function does all the initialization needed for the exit. If information is needed further on in subsequent calls, a handle can be filled in the initialization call, which is then passed to all subsequent functions (for example, DB handles, connection information, states, etc.). Deinit(), which is called last, normally deallocates and frees all resources allocated during Init().

The exit DLL is loaded by the PES when the exit is needed the first time and unloaded when the PES terminates.

The main difference between the exit types is the following:

- **Mapping** exits do a data conversion between MQSeries Workflow for OS/390 containers and data acceptable by legacy applications.
- **Invocation** exits invoke applications on the application side.

Notes:

1. Whenever you modify an exit, you must shutdown and reboot the PES in order to make your changes effective.
2. All PES exits must be reentrant.

Return codes and error messages

All exits use a return code to signal availability of error information from the exit functions to the PES. If the return code is not OK, 4 parameters used in each function contain more detailed error information. The possibility exists to classify errors as recoverable or non-recoverable. The PES no longer distinguishes between these two types of errors. Both are handled the same way: by passing the error to the execution server for a Workflow request or in the case of an Execute Program request issued by a Workflow client.

Parameters:

char * errorIdBuffer

- 4-character buffer provided by the PES and used to pass an error number. Must be set accordingly if the ID is shorter than 4 characters.
- Input/output parameter

long * errorIdBufferLength

Program execution server exits

- Length of the message number in `errorIdBuffer`, which must be set by the exit. The maximum available number of characters is specified. Valid lengths are between 0 and the value passed in `errorIdBufferLength`.
- Input/output parameter

`char * errorDescriptionBuffer`

- Character buffer provided by the PES, which is 512 characters long and is used for an error message. Has to be set accordingly if the message is shorter than 512 characters.
- Input/output parameter

`long * errorDescriptionBufferLength`

- Length of the description in `errorDescriptionBuffer`, which must be set by the exit. The maximum available number of characters is passed. Valid lengths are between 0 and the value passed in `errorDescriptionBufferLength`.
- Input/output parameter

The error ID can consist of up to 4 digits. The PES prefixes it with a character identifying the exit type (I for invocation, M for mapping). In addition the `errorDescription` is prefixed by the PES with the DLL name of the exit so that each exit can use the message numbers and the DLL name is the identifier of the exit.

Return codes:

`FMC_EXIT_OK`

The function was successful.

`FMC_EXIT_RECOVERABLE_ERROR`

The function was unsuccessful but recoverable. The PES will return message `FMC32204` (see *MQSeries Workflow for OS/390: Messages and Codes*) with the passed error information. The PES continues processing. `errorIdBuffer`, `errorIdBufferLength`, `errorDescriptionBuffer`, and `errorDescriptionBufferLength` must be set accordingly .

`FMC_EXIT_NONRECOVERABLE_ERROR`

The function was unsuccessful and unrecoverable. The PES handles this error similar to recoverable errors.

Interfaces for all exits

Note: All interfaces must have C linkage!

`Init()`

Header files: `FMCXMIF.H` (program mapping exit) and `FMCXIEP.H` (invocation exit).

Function:

- Initialize exit.
- It is called once when the exit is used the first time.

Interface:

```
long Init
( void ** exitHandle,
  void *  initializationParameter,
  long   initializationParameterLength,
```

Program execution server exits

```
char *   errorIdBuffer,  
long *   errorIdBufferLength,  
char *   errorDescriptionBuffer,  
long *   errorDescriptionBufferLength)
```

Parameters:

void ** exitHandle

- Pointer passed to the Init() function, which is not used by the PES but passed to any function called at a later time. Used to pass exit environment data between all exit functions.
- Input/output parameter

void * initializationParameter

- Parameters defined in the PES directory entry for the exit, which can be used to customize the exit initialization. The parameter is terminated by zero.
- Input parameter

long initializationParameterLength

- Length of the initializationParameter in bytes.
- Input parameter

char * errorIdBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorIdBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

char * errorDescriptionBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorDescriptionBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

Return codes: See “Return codes and error messages” on page 198 for the return codes.

Deinit()

Header files: FMCXMIF.H (program mapping exit) and FMCXIEP.H (invocation exit).

Function:

- Deinitialize exit.
- It is called once when the PES terminates.

Interface:

```
long Deinit  
( void ** exitHandle,  
  char *   errorIdBuffer,  
  long *   errorIdBufferLength,  
  char *   errorDescriptionBuffer,  
  long *   errorDescriptionBufferLength)
```

Parameters:

void ** exitHandle

- Pointer passed to the Init() function, which is not used by the PES but passed to any function called later on. Used to pass exit environment data between all exit functions.
- Input/output parameter

char * errorIdBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorIdBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

char * errorDescriptionBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorDescriptionBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

Return codes: See “Return codes and error messages” on page 198 for the return codes.

Special considerations for exit programming

Use of RRS commit and rollback

PES exits are called in the context of an MQSeries Workflow transaction by the program execution server (PES). Changes made by the exits can be made persistent (RRS commit) or rescinded (RRS rollback). However, these actions also terminate the transaction itself. Since PES exits can be called at any point during the transaction, RRS commit or rollback would terminate the transaction prematurely and lead to unpredictable results. In particular, a “safe application” would no longer be “safe”.

Consequently, RRS commit and rollback calls **srrcmit** and **srrback** must never be issued in a PES exit. Furthermore, any functions performing these calls must *not* be used in PES exits; otherwise, units of work are terminated by these calls before the PES can complete them.

Buffer allocation

Both invocation and mapping exits use buffers to receive data from the PES and pass data to the PES. In some situations, the buffers are owned by the PES and must not be deleted by the exit. In other situations, the buffers must be created and deleted by the exit. For specific requirements, see the descriptions of the individual buffer parameters.

Program mapping exit

The IBM-supplied program mapping exit can be used to convert and translate data between legacy applications and MQSeries Workflow for OS/390. Any other mapping exit that conforms to the mapping exit interface can be used in parallel with, or in place of, the IBM supplied mapping exit.

Program execution server exits

The program mapping exit must be available in a DLL which is loaded by the PES and used to translate the MQSeries Workflow for OS/390 container data into data accepted by a legacy application. This DLL must have two exit functions: `Init()` and `Deinit()` (see above) and one exit specific function `Translate()`. It is used to do the actual conversion and translation for forward and backward mappings every time a legacy application is called. The raw buffer for the legacy application is available, and the container can be accessed via MQSeries Workflow for OS/390 Container API calls.

In forward mapping calls, the program mapping exit must extract the data from the container and translate the data into a raw buffer which is passed to the legacy application. In backward mapping calls the program mapping exit must translate the raw buffer and assign the data to the container.

Additional interfaces specific to the program mapping exit

A mapping exit must follow the general rules for PES exits as described in "Interfaces for all exits" on page 199.

Translate()

Header files: FMCXMIF.H (program mapping exit).

Function:

- **Translate()** is called each time by the PES in the direction `FMC_PROGRAMMAPPING_BACKWARDMAPPING` when backward mapping is to be done and in the direction `FMC_PROGRAMMAPPING_FORWARDMAPPING` when forward mapping is to be done.

Interface:

```
long Translate
(
    void * exitHandle,
    short direction,
    char * mappingName,
    long mappingNameLength,
    char * buildTimeParameter,
    long buildTimeParameterLength,
    void * containerHandle,
    char** buffer,
    long * bufferLength,
    char * errorIdBuffer,
    long * errorIdBufferLength,
    char * errorDescriptionBuffer,
    long * errorDescriptionBufferLength)

```

Parameters:

void * exitHandle

- Pointer passed to the `Init()` function, which is not used by the PES but passed to any function called at a later time. Used to pass exit environment data between all exit functions.
- Input/output parameter

short direction

- Used to identify the translation direction. Either `FMC_PROGRAMMAPPING_BACKWARDMAPPING` or `FMC_PROGRAMMAPPING_FORWARDMAPPING`
- Input parameter

char * mappingName

- Forward mapping format or backward mapping format defined in the OS/390 program properties. The name is zero terminated.
- Input parameter

long mappingNameLength

- Length of the mappingNameLength in bytes.
- Input parameter

char * buildTimeParameter

- Forward mapping parameter or backward mapping parameter defined in the OS/390 program properties. Can be used to customize the translation process. The parameter is zero terminated.
- Input parameter

long buildTimeParameterLength

- Length of the buildTimeParameter in bytes.
- Input parameter

void * containerHandle

- Handle to the MQSeries Workflow for OS/390 container used for translation. Argument in MQSeries Workflow for OS/390 container APIs to access the container.
- Input parameter

char ** buffer

- A valid buffer address for forward mapping calls. It must be set by the program mapping exit to a valid buffer address in backward mapping calls. This buffer is passed by the PES to the legacy application in forward mapping calls and to the program mapping exit for data conversion and setting of container elements in backward mapping calls.
- Input parameter for forward mapping. The buffer is created and owned by the exit and passed to the PES.
- Output parameter for backward mapping. The buffer is created and owned by the PES and passed to the exit.

long * bufferLength

- Length of the buffer in bytes. This is already set in forward mapping calls and must be set in backward mapping calls by the program mapping exit.
- Input parameter for forward mapping
- Output parameter for backward mapping

char * errorIdBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorIdBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

char * errorDescriptionBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorDescriptionBufferLength

Program execution server exits

- See “Return codes and error messages” on page 198 for detailed information.

Return codes: See “Return codes and error messages” on page 198 for the return codes.

See the program mapping sample “Program mapping exit sample” for more details.

Enabling the PES to use a program mapping exit

In order to use a program mapping exit, the exit must reside in a DLL which must be available in a link library used by the PES. Customize the sample JCL FMCHJMEX in *InstHLQ.SFMCCNTL* and submit the job.

You must then notify the PES of the program mapping exit by defining it in the PES directory. See *MQSeries Workflow for OS/390: Customization and Administration* for more information.

Program mapping exit sample

A C sample program mapping exit (FMCHSMEX in *InstHLQ.SFMCSRC*) with a corresponding JCL to compile and link the exit (FMCHJMEX in *InstHLQ.SFMCCNTL*).

The sample exit gives an example how a program mapping exit should work in general. It also shows how the exit can use all the different parameters passed (exit initialization parameter, forward/backward mapping format and forward/backward mapping parameters), how error messages can be created and how the container elements can be accessed.

The exit can convert MQSeries Workflow for OS/390 container elements of type LONG and FLOAT into C types long and double. The elements are converted in the same sequence that the elements are defined in the container structure.

The type of the element is derived from the first character of the element name and must be defined in the PES directory entry for the mapping exit (exitParameters).

ExitParameter syntax: LONG=x FLOAT=y

where x is the character used to identify LONG container elements and y is the character to identify FLOAT container elements. All elements starting with an undefined character are ignored.

In addition TRACE=YES can be defined during initialization to enable trace printouts.

Mapping example:

```
STRUCTURE S
  LE1: LONG(2);
  FE1: FLOAT;
  LE2: LONG;
END;
```

will be converted so that the following structure is filled with conversion data:


```
struct S {  
    long LE1[2];  
    double FE1;  
    long LE2;  
};
```

The following three different mapping formats are available:

1. DEFAULT
2. INCREASE_INCOME
3. DECREASE_INCOME

The first format does normal conversion of the values. The second one increases all values by 8% (default), while the third format decreases all values by 8% (default). If other percentages are needed, they can be defined as forward mapping parameters or backward mapping parameters using the Buildtime tool when the program properties for OS/390 are defined.

To use the mapping sample, you must

1. Compile and link the mapping sample (See JCL FMCHJMEX in *InstHLQSFMCNTL*).
2. Update the PES directory with following definition (see "Importing a PES directory source file" in *MQSeries Workflow for OS/390: Customization and Administration*).
(KEYTOMAPPING2)
type =SAMPLE
exitName =SAMPEXT
exitParameters =LONG=L FLOAT=F
3. Define a program in BuildTime that uses the sample mapping type SAMPLE and optionally provides a different increase/decrease amount for forward/backward mapping parameters.

Program invocation exit

A program invocation exit is used by the program execution server to run a requested application on a service system like CICS or IMS. The corresponding invocation request is issued to the program execution server by an MQSeries Workflow for OS/390 execution server.

To handle an invocation request, the program execution server uses an exit containing the implementation of an invocation type such as *EXCI*. This allows application developers using MQSeries Workflow for OS/390 to attach their own invocation types to MQSeries Workflow for OS/390. An invocation exit must follow the general rules for program execution server exits as described in "Program execution server exits" on page 198 in this book. As for all program execution server exits, invocation exits are available as dynamic link libraries providing entry points.

Synchronous and asynchronous invocation exits

MQSeries Workflow for OS/390 supports synchronous and asynchronous invocation exits. They are characterized by the following. Asynchronous invocations must be based on MQSeries queues.

Synchronous invocation exit: A synchronous invocation exit

- establishes the connection to the service where the request should run.
- runs the requested application on that service by use of the invocation protocol.

Program execution server exits

- removes the connection to the service after the application has terminated.
- passes back the output data from the application, or an error message if the invocation failed, to the program execution server.
- if, depending on the invocation protocol, connections can be reused by subsequent calls, the connections are cached after being used a first time and removed at deinitialization of the invocation exit.

The program execution server "waits" for the output from a synchronous invocation exit.

Asynchronous invocation exit: When processing a request, the program execution server calls an asynchronous invocation exit twice. First to handle a request, and second to handle a reply. If an asynchronous invocation exit is called by the program execution server with parameters describing a request, it does not execute a request from the program execution server directly on the target service system. It only creates a message consisting of an invocation specific header followed by the data for the requested application and passes it back to the program execution server. It also creates a message descriptor (which is an MQSeries message descriptor MQMD since only MQSeries based invocations are supported) and passes it back also.

The program execution server then uses the message descriptor and message together with the connection parameters to put the message to the input queue as specified by the connection parameters. The program execution server does not wait until the requested application has finished but rather continues with the next request.

When the program execution server gets the reply message consisting of message descriptor and message data, it calls the respective invocation exit to recognize the reply message. It is recommended that only asynchronous invocation protocols be used for which the message contains protocol-specific data at the beginning, for example CICS bridge header MQCIH or IMS bridge header MQIIH, so that the reply can be correctly handled by the invocation exit. Also, it is recommended to reflect that format in the message descriptor (by the Format field in the MQMD structure).

If an exit recognizes the reply, it is called to handle it and passes the application output data, or an error message, back to the program execution server.

Note: An asynchronous invocation exit does **not** connect itself to a service. The MQSeries queues to and from the service are always served by the PES.

Additional interfaces specific to the invocation exit

Each asynchronous and synchronous invocation exit must provide the following additional entry points beside the interfaces provided by all program execution server exits:

HdlRequ()

Header files: FMCXIEP.H (invocation exit).

Function:

- For a synchronous invocation, this method executes an application program passed as **executableName** on a service to which it connects as defined by *connectionParameters*. If no error occurs, this function returns the output from the application in a buffer passed back by the **programOutput** parameter.

Program execution server exits

- For an asynchronous invocation this method either creates a request message or treats the passed parameters as a reply message. The parameters represent a request message if an application name is passed as **executableName**. If zero is passed as **executableName**, the parameters represent a reply message whose data is passed as **executableParameters** and **executableParametersLength** together with a message descriptor MQMD as **messageDescriptor** and **messageDescriptorLength**.

Interface:

```
long Hd1Requ
(
    void * exitHandle,
    void * invocationContext,
    char * serviceName,
    long serviceNameLength,
    char * connectionParameters,
    long connectionParametersLength,
    char * executableName,
    long executableNameLength,
    char * executableType,
    long executableTypeLength,
    char * executionParameters,
    long executionParametersLength,
    char ** programOutput,
    long * programOutputLength,
    char ** messageDescriptor,
    char * messageDescriptorLength,
    char * errorIdBuffer,
    long * errorIdBufferLength,
    char * errorDescriptionBuffer,
    long * errorDescriptionBufferLength)

```

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit
- Input parameter

void * invocationContext

- Address of data describing the context in which the invocation of the request should be done.
- For the context information provided by MQSeries Workflow for OS/390, see “Invocation context” on page 210
- input parameter

char * serviceName

- Name of service as known to MQSeries Workflow for OS/390 where the requested application is to be performed.
- Input parameter

long serviceNameLength

- Length of serviceName character string
- Input parameter

char * connectionParameters

- Character string providing the parameters needed by the invocation to connect to the service where the requested program should run. These parameters are defined for the respective service in the PES directory.
- Input parameter
- See also “Connection parameters” on page 212

Program execution server exits

long connectionParametersLength

- Length of connectionParameters
- Input parameter

char * executableName

- Address of buffer containing the name of the executable of the request
- For asynchronous invocations only: the program execution server passes zero to indicate that this function is called to handle a reply message from an asynchronously invoked program execution request.
- Input parameter

long executableNameLength

- Length of executableName
- Input parameter

char * executableType

- Type of the program specified by the executableName
- Input parameter

long executableTypeLength

- Length of executableType
- Input parameter

char * executionParameters

- Parameters for the program specified by the executable name
- Reply from an asynchronously performed execution request
- Input parameter

long executionParametersLength

- Length of executionParameters or the asynchronous reply
- Input parameter

char ** programOutput

- Address of return buffer containing the output of the executable
- Asynchronous invocations use this parameter also to return the protocol-conforming message to the program execution server corresponding to the execution request passed in as **executableName** and **executionParameters**.
- This buffer is created and owned by the exit and passed to the PES.
- Input/output parameter

long * programOutputLength

- Address to length of output data or the protocol-conforming message
- Input/output parameter

char ** messageDescriptor

- Address to pass/return a buffer containing a message descriptor
- For asynchronous invocations only
- If the executable name is passed (request message has been constructed), the buffer is created and owned by the exit and passed to the PES. If a reply message is being handled, the buffer is created and owned by the PES and passed to the exit.
- Input/output parameter

long * messageDescriptorLength

- Address to buffer containing the length of the message descriptor
- For asynchronous invocations only
- Input/output parameter

char * errorIdBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorIdBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

char * errorDescriptionBuffer

- See “Return codes and error messages” on page 198 for detailed information.

long * errorDescriptionBufferLength

- See “Return codes and error messages” on page 198 for detailed information.

Return codes:

FMC_EXIT_OK

- See “Return codes and error messages” on page 198 for detailed information.

FMC_EXIT_RECOVERABLE_ERROR

- See “Return codes and error messages” on page 198 for detailed information.

FMC_EXIT_NONRECOVERABLE_ERROR

- See “Return codes and error messages” on page 198 for detailed information.

Recogn()

Header files: FMCXIEP.H (invocation exit).

Function:

- Checks whether the reply message passed as message descriptor and message data is recognized by the invocation type this exit represents.
- This method applies to exits for asynchronous invocations. However, this entry point must also be provided by synchronous invocation exits. For these exits, FMC_INV_NOT_RECOGNIZED must always be returned.
- This function is called for all asynchronous invocation exits when an invocation reply message is received. It is assumed that there are no ambiguities, so that the first invocation exit where this function recognizes this message is the correct one to handle it.

Interface:

```
long Recogn  
(void * exitHandle  
 char * messageDescriptor,  
 long messageDescriptorLength,  
 char * messageData,  
 long messageDataLength)
```

Program execution server exits

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit
- Input parameter

char * messageDescriptor

- Descriptor of the reply message
- Input parameter

long * messageDescriptorLength

- Length of messageDescriptor
- Input parameter

char * messageData

- The data of the reply message
- Input parameter

long messageDataLength

- Length of messageData
- Input parameter

Return codes:

FMC_INV_NOT_RECOGNIZED

The message is not recognized

FMC_INV_RECOGNIZED

The message is recognized

IsAsync()

Header files: FMCXIEP.H (invocation exit).

Function:

- Returns whether the exit represents an asynchronous invocation.

Interface:

```
long IsAsync  
    (void* exitHandle)
```

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit
- Input parameter

Return codes:

FMC_INV_SYNCHRONOUS

The invocation is synchronous.

FMC_INV_ASYNCHRONOUS

The invocation is asynchronous.

Invocation context

The PES passes an invocation context to the invocation exit for HdlRequ(). It contains information about the context in which the request should be executed. There are four different context types: Workflow, Security, Transaction, and

Program execution server exits

Performance. The context is created by the PES. The content depends on server and program settings. The workflow context is passed for internal reasons and is not intended to be used by an invocation exit. The security context is either set to the PES user ID or the execution user ID resolved for the request. The transaction context is set to a nonzero value if the request is to be executed as a safe application. The performance context is set to the WLM enclave token if the PES is running WLM-managed.

An invocation accesses the context by using a C-type interface defined in the included file `fmcxiinv.h`.

GetContext()

Header files: `FMCXIINV.H` (invocation context).

Function:

- Extracts the value of the context type as specified by a passed context name.

Interface:

```
int GetContext
(void * invContext,
 char * invContextName,
 char ** invContextValue,
 long * invContextValueLength)
```

Parameters:

void * invContext

- Address of the complete invocation context containing all context types
- Input parameter

char * invContextName

- Name of the requested context type
- Can be one of the character string constants as listed in the *Name* column in the *Context types* table below.
- Input parameter

char ** invContextValue

- Address to which this function puts the pointer to the value of the requested context type
- Input/output parameter

long * invContextValueLength

- Address to which this function puts the length of the value of the requested context type
- Input/output parameter

Return codes:

FMC_INV_CTX_ON

Context successfully retrieved

FMC_INV_CTX_NOT_SET

Context has not been set

FMC_INV_CTX_NOT_DEFINED

Context not found

Program execution server exits

Table 16. Context types

| Type | Meaning | Used by | Name |
|-------------|---|-------------------|------------------|
| Workflow | Contains the Workflow context | Internal use only | FMC_WORKFLOW_CTX |
| Security | Contains the user identification the request should be executed for | Invocation exit | FMC_SECURITY_CTX |
| Transaction | Contains an indication that the request should be executed in transactional context | Invocation exit | FMC_TRANSACT_CTX |
| Performance | Contains the WLM enclave token, if the PES is running WLM managed | Invocation exit | FMC_PERFMGMT_CTX |

To link your exit correctly, you must include definition side deck FMCH0XIC from *InstHLQ.SFMCDS*.

Connection parameters

The parameter *connectionParameters* of the entry point **HdlRequ()** represents a character string of up to 254 printable characters. This string contains the parameters needed by the invocation to connect to the service system in order to execute an application there.

Connection parameters for synchronous invocations: For synchronous invocations, these parameters and the syntax how to specify them must be defined by the developer of an invocation exit. In most cases it is recommended to use the following syntax:

```
<keyword_1>=<value_1>;<keyword_2>=<value_2>;...;<keyword_n>=<value_n>
```

where the parameters are represented by key-value pairs separated by semicolons. Nevertheless, you can define the syntax of the connection parameters for your invocation exit. Connection parameters and the syntax how to specify them must be part of the documentation of your invocation exit, since an MQSeries Workflow for OS/390 administrator must specify this string when defining this exit to the PES directory.

Connection parameters for asynchronous invocation: Because only asynchronous invocations based on MQSeries queues are supported, the connection parameters for asynchronous invocations and the syntax how to specify them must be

```
[QUEUEMANAGER=<queuemanagename>;]INPUTQUEUE=<inputqueuename>
```

where *<queuemanagename>* and *<inputqueuename>* represent the MQSeries queue manager or MQSeries queue, respectively, and must follow the corresponding MQSeries naming rules. These parameters define the MQSeries queue to which the request message created by an asynchronous invocation exit must be put. QUEUEMANAGER is optional and should be omitted when the target queue manager is part of a queue manager cluster. When work distribution is required, INPUTQUEUE should specify a cluster queue defined with the attribute DEFBIND=<NOTFIXED>. At least two queues with this name should exist within the cluster.

Enabling MQSeries Workflow for OS/390 to use an invocation exit

In order to use a program invocation exit, the exit must reside in a DLL which must be available in a link library used by the PES.

Note:

If more than one asynchronous invocation exit recognizes the same kind of reply messages, it is unpredictable which of the exits will handle a reply message. This may lead to incorrect results, the cause of which may be difficult to determine.

See also "Program execution" in *MQSeries Workflow for OS/390: Customization and Administration*.

Invocation exit coding example

Since an invocation exit assumes there is a special kind of service system available where an application should run, there are no compiling sources provided as samples but rather a skeleton for synchronous invocations FMCHSIVS and one for asynchronous invocations FMCHSIVA, both in *InstHLQ.SFMCSRC* and written in C.

Chapter 4. API classes and objects

This chapter lists the individual classes and objects in the API and their component API calls.

Summary

An alphabetical list of API classes and objects follows. Unless otherwise stated, indicated names are valid Java classes. To become valid C++ classes, a prefix of "Fmcj" must be added. To become valid C or COBOL calls, the class name must be prefixed with "Fmcj" and extended by the actual function name. For example, if the C++ class is "FmcjWorkitem", all functions in C or COBOL start with "FmcjWorkitem"; "FmcjWorkitemStart", for example, is a valid C function and COBOL subprogram name.

Note: In the following discussions, use of the listed class names should be understood as a generic designation that also applies to C, C++, and COBOL.

To adhere to language requirements, the paragraph names in copybook fmcperf.cpy are usually abbreviated versions of the C function names. For more information concerning COBOL naming conventions, see "COBOL interface" on page 133.

| Class/Object | Description |
|------------------------------------|---|
| ActivityInstance | An instance of a workflow process template activity. |
| ActivityInstanceNotification | A notification associated with an activity instance. |
| ActivityInstanceNotificationVector | The C or COBOL result of a query for activity instance notifications. |
| ActivityInstanceVector | The C or COBOL result of a query for activity instances. |
| Agent | An agent in the Java API to access an MQSeries Workflow domain. |
| BlockInstanceMonitor | The monitor for an activity instance of kind Block. |
| Calendar | The date/time object for Java. See "Date and Time". |
| Container | The data container of a work item or a process instance. |
| ContainerElement | An element of a data container. |
| ContainerElementVector | The C or COBOL result of a query for container elements. |
| ControlConnectorInstance | The instance of a control connector between two activity instances. |
| ControlConnectorInstanceVector | The C or COBOL result of a query for control connector instances. |

API classes and objects

| Class/Object | Description |
|-----------------------------------|--|
| Date and Time | The corresponding C++ object is FmcjDateTime. FmcjCDateTime is the equivalent structure in C and COBOL. Note: Java uses the Calendar object. |
| DllOptions | The program implementation definitions for a dynamic link library. |
| ExecutionData | Information pushed by an MQSeries Workflow execution server or the response to an asynchronous request. |
| ExecutionAgent | The Java representation of an MQSeries Workflow program execution agent. |
| ExecutionService | The representation of a session between a user and an MQSeries Workflow execution server so that services can be requested. |
| ExeOptions | The program implementation definitions for an executable. |
| ExternalOptions | The program implementation definitions for an external service. |
| FmcError | Describes the cause of a state InError in Java. The C++ class is called FmcjError. |
| FmcException | The Java representation of an exception. |
| Global | A means to group API calls which are global API calls in C, C++, and COBOL. |
| ImplementationData | The program implementation definitions. |
| Item | An item associated with a user; can be a work item or notification. |
| ItemVector | The C or COBOL result of a query for items. |
| Message | A means to request a formatted message (in the local language, if applicable) for a known message ID; only C, COBOL, and C++. |
| PersistentList | A list definition stored persistently. |
| Person | User-specific settings for the user logged on to an MQSeries Workflow execution server. |
| Point | Describes the bend points of a control connector instance. |
| PointVector | The C or COBOL result of a query for bend points. |
| ProcessInstance | An instance of a workflow process template. |
| ProcessInstanceList | A list to group process instances. |
| ProcessInstanceListVector | The C or COBOL result of a query for process instance lists. |
| ProcessInstanceMonitor | The monitor for a process instance. |
| ProcessInstanceNotification | A notification associated with a process instance. |
| ProcessInstanceNotificationVector | The C or COBOL result of a query for process instance notifications. |
| ProcessInstanceVector | The C or COBOL result of a query for process instances. |

| Class/Object | Description |
|---------------------------|---|
| ProcessTemplate | A workflow process template consisting of activities and containers and their control and data flow. |
| ProcessTemplateList | A list to group process templates. |
| ProcessTemplateListVector | The C or COBOL result of a query for process template lists. |
| ProcessTemplateVector | The C or COBOL result of a query for process templates. |
| ProgramData | The program definitions of an activity implementation. |
| ProgramTemplate | A program definition contained in a process template. |
| ReadOnlyContainer | A data container that can only be read. |
| ReadWriteContainer | A data container that can be read and written to. |
| Result | The detailed result of a request; only C, COBOL, and C++. |
| Service | Provides for common aspects of MQSeries Workflow services. |
| StringVector | The C or COBOL result of a query resulting in a list of strings or the C or COBOL means of providing a list of strings. |
| SymbolLayout | Describes the graphical layout of an activity instance. |
| Workitem | A user-assigned activity instance to be worked on. |
| WorkitemVector | The C or COBOL result of a query for work items. |
| Worklist | A list to group work items or notifications. |
| WorklistVector | The C or COBOL result of a query for worklists. |

API calls by class

Note: In the following descriptions, the basic methods listed are not differentiated by language. Not all of these methods are available in each language.

ActivityInstance

An activity instance represents an instance of a process template activity. It is part of a process instance.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs an activity instance object. | 71 |
| Copy() | Allocates and initializes the storage for an activity instance object by copying. | 74 |
| Deallocate() | Deallocates the storage for an activity instance object. | 75 |
| destructor() | Destructs an activity instance object. | 75 |
| Equal() | Compares two activity instances. | 73 |

API classes and objects

| Basic methods | Description | Page |
|---------------|---|------|
| IsComplete() | Indicates whether the complete activity instance information is available. | 75 |
| IsEmpty() | Indicates whether activity instance information is not available. | 76 |
| Kind() | States the kind of the activity instance, whether it is a program, a process, or a block. | 76 |
| operator=() | Assigns an activity instance to this one. | 73 |
| operator==() | Compares two activity instances. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance. Note that the activity instances returned by the (process or block) instance monitor contain both primary and secondary values.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|------------------------|--------------|--|------|
| ActivationTime() | P/D | Returns the activation time of the activity instance. | 87 |
| ActivationTimeIsNull() | P/B | Indicates whether an activation time is set. | 110 |
| Category() | P/C | Returns the process category of the activity instance. | 107 |
| CategoryIsNull() | P/B | Indicates whether a category is set. | 110 |
| Description() | P/C | Returns the description of the activity instance. | 107 |
| DescriptionIsNull() | P/B | Indicates whether a description is set. | 110 |
| Documentation() | S/C | Returns the documentation of the activity instance. | 107 |
| DocumentationIsNull() | S/B | Indicates whether a documentation is set. | 110 |
| EndTime() | S/D | Returns the ending time of the activity instance. | 87 |
| EndTimeIsNull() | S/B | Indicates whether an end time is set. | 110 |
| ErrorReason() | S/O | Returns an error object describing the reason why the activity instance is in the state InError. | 108 |
| ErrorReasonIsNull() | S/B | Indicates whether an error reason is set. | 110 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|-------------------------------|--------------|--|------|
| ExitCondition() | S/C | Returns the exit condition of the activity instance. | 107 |
| FirstNotificationTime() | S/D | Returns the time the first notification for the activity instance is to occur or has occurred. | 87 |
| FirstNotificationTimeIsNull() | S/B | Indicates whether a first notification time is set. | 110 |
| FirstNotifiedPersons() | S/M | Returns the persons who received a first notification for the activity instance. | 107 |
| FullName() | P/C | Returns the fully qualified name of the activity instance (dot notation). | 107 |
| Icon() | P/C | Returns the icon associated with the activity instance. | 107 |
| Implementation() | P/C | Returns the name of the implementing program of the activity instance. | 107 |
| ImplementationIsNull() | P/B | Indicates whether an implementation is set. | 110 |
| InContainerName() | S/C | Returns the name of the input container of the activity instance. | 107 |
| LastModificationTime() | P/D | Returns the last time a primary attribute of the activity instance was changed. | 87 |
| LastStateChangeTime() | P/D | Returns the last time the state of the activity instance changed. | 87 |
| ManualExitMode() | S/B | Returns whether the exit mode of the activity instance is manual. | 86 |
| ManualStartMode() | S/B | Returns whether the start mode of the activity instance is manual. | 86 |
| Name() | P/C | Returns the name of the activity instance. | 107 |
| OutContainerName() | S/C | Returns the name of the output container of the activity instance. | 107 |
| PersistentOid() | P/C | Returns a representation of the object identification of the activity instance. | 107 |
| Priority() | P/I | Returns the priority of the activity instance. | 106 |
| PriorityIsNull() | P/B | Indicates whether a priority is set. | 110 |
| ProcessAdmin() | S/C | Returns the process administrator of the activity instance. | 107 |
| ProcessAdminIsNull() | S/B | Indicates whether a process administrator is set. | 110 |
| ProcessInstanceName() | P/C | Returns the name of the process instance the activity instance is part of. | 107 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|----------------------------------|--------------|---|------|
| ProcessInstanceState() | P/E | Returns the state of the process instance the activity instance is part of. | 87 |
| ProcessInstanceSystemGroupName() | S/C | Returns the name of the system group of the process instance the item is part of. | 107 |
| ProcessInstanceSystemName() | S/C | Returns the name of the system of the process instance the activity instance is part of. | 107 |
| SecondNotificationTime() | S/D | Returns the time the second notification for the activity instance is to occur or has occurred. | 87 |
| SecondNotificationTimeIsNull() | S/B | Indicates whether a second notification time is set. | 110 |
| SecondNotifiedPersons() | S/M | Returns the persons who received a second notification for the activity instance. | 107 |
| Staff() | S/M | Returns all persons a work item for the activity instance has been assigned to. | 107 |
| StartCondition() | S/C | Returns the start condition of the activity instance. | 107 |
| Starter() | P/C | Returns the starter of the activity instance. | 107 |
| StarterIsNull() | P/B | Indicates whether a starter is set. | 110 |
| StartTime() | P/D | Returns the start time of the activity instance. | 87 |
| StartTimeIsNull() | P/B | Indicates whether a start time is set. | 110 |
| State | P/E | Returns the state of the activity instance. | 87 |
| StateOfNotification() | S/E | Returns the notification state of the activity instance. | 87 |
| SupportTools() | P/M | Returns the support tools associated with the activity instance. | 107 |
| SupportToolsIsNull() | P/B | Indicates whether support tools are set. | 110 |
| SymbolLayout() | S/O | Returns the symbol layout of the activity instance. | 108 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------------------|---|------|
| ObtainProcessInstanceMonitor() | Retrieves the process instance monitor for the process instance the activity instance is part of. | 273 |
| SubProcessInstance() | Retrieves the process instance implementing the activity instance of type Process. | 275 |

ActivityInstanceNotification

An activity instance notification represents a notification for an activity instance.

Note: All API calls of `FmcjItem` are also applicable to activity instance notifications.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------------------|--|------|
| <code>constructor()</code> | Constructs an activity instance notification object. | 71 |
| <code>Copy()</code> | Allocates and initializes the storage for an activity instance notification object by copying. | 74 |
| <code>Deallocate()</code> | Deallocates the storage for an activity instance notification object. | 75 |
| <code>destructor()</code> | Destructs an activity instance notification object. | 75 |
| <code>Kind()</code> | In C++, states that the object is an activity instance notification. | 76 |
| <code>operator=()</code> | Assigns an activity instance notification to this one. | 73 |
| <code>operator==()</code> | Compares two activity instance notifications. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instance notifications are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance notification.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|--------------------------------------|--------------|--|------|
| <code>ActivityKind()</code> | P/E | Returns the kind of the associated activity instance, whether it is a program or process and so on. | 87 |
| <code>ErrorReason()</code> | S/O | Returns an error object describing the reason why the associated activity instance is in state <code>InError</code> . | 108 |
| <code>ErrorReasonIsNull()</code> | S/B | Indicates whether an error reason is set. | 110 |
| <code>ExitCondition()</code> | S/C | Returns the exit condition of the associated activity instance. | 107 |
| <code>Expired()</code> | P/B | Returns whether the associated activity instance has been started and is expired now. | 86 |
| <code>FirstNotificationTime()</code> | S/D | Returns the first notification time of the activity instance, that is, the time when this notification has been created. | 87 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--------------------------------|--------------|---|------|
| Implementation() | P/C | Returns the implementing program or process name of the associated activity instance. | 107 |
| ImplementationIsNull() | P/B | Indicates whether an implementation is set. | 110 |
| ManualExitMode() | S/B | Returns whether the exit mode of the associated activity instance is manual. | 86 |
| ManualStartMode() | S/B | Returns whether the start mode of the associated activity instance is manual. | 86 |
| Priority() | P/I | Returns the priority of the associated activity instance. | 106 |
| SecondNotificationTime() | S/D | Returns the second notification time of the associated activity instance. | 87 |
| SecondNotificationTimeIsNull() | S/B | Indicates whether a second notification time is set. | 110 |
| Staff() | S/M | Returns all persons owning a work item for the associated activity instance. | 107 |
| StartCondition() | S/C | Returns the start condition of the associated activity instance. | 107 |
| StartOverdue() | P/B | Returns whether the start of the associated activity instance is overdue. | 86 |
| State | P/E | Returns the state of the associated activity instance. | 87 |
| StateOfNotification() | S/E | Returns the notification state of the associated activity instance. | 87 |
| SupportTools() | P/M | Returns the support tools associated with the activity instance. | 107 |
| SupportToolsIsNull() | P/B | Indicates whether support tools are set. | 110 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------|---|------|
| PersistentObject() | Retrieves the specified activity instance notification. | 278 |
| StartTool() | Starts the specified support tool. | 280 |

ActivityInstanceNotificationVector

An activity instance notification vector represents the result of a query for activity instance notifications in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector functions.

| Vector methods | Description |
|----------------|--|
| Deallocate() | Deallocates an activity instance notification vector object. |
| FirstElement() | Returns the first element of the activity instance notification vector. |
| NextElement() | Returns the next element of the activity instance notification vector. |
| Size() | Returns the number of elements in the activity instance notification vector. |

ActivityInstanceVector

An activity instance vector represents the result of a query for activity instances in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates an activity instance vector object. |
| FirstElement() | Returns the first element of the activity instance vector. |
| NextElement() | Returns the next element of the activity instance vector. |
| Size() | Returns the number of elements in the activity instance vector. |

Agent

An agentobject represents an MQSeries Workflow instance in Java.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs an agent object. Initially an agent has no context, locator policy, or name. | 71 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-----------------------------|------|--|------|
| addPropertyChangeListener() | O | Adds the specified listener to the set of listeners to be notified of property changes. | 112 |
| addVetoableChangeListener() | O | Adds the specified listener to the set of listeners to be notified of vetoable property changes. | 112 |
| getConfigurationID() | C | Returns the configuration to be used for profile accesses. | 107 |

API classes and objects

| Accessor methods | Type | Description | Page |
|---|------|--|------|
| <code>getExecutionAgent()</code> | O | Returns a program execution agent to the calling activity implementation provided that the LOC_LOCATOR policy was used. Otherwise, null is returned. | 108 |
| <code>getLocator()</code> | I | Returns the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR. | 106 |
| <code>getName()</code> | C | Returns the name of the Java Agent. If the agent is not bound, an empty string is returned. | 107 |
| <code>isBound()</code> | B | Indicates whether the agent bean is bound to a Java CORBA agent. | 86 |
| <code>locate()</code> | O | Locates the execution service in the provided system group and system. | 108 |
| <code>removePropertyChangeListener()</code> | O | Removes the specified listener from the set of listeners. | 112 |
| <code>removeVetoableChangeListener()</code> | O | Removes the specified listener from the set of listeners. | 112 |
| <code>setConfigurationID()</code> | C | Sets the configuration ID to be used for profile access. A locator policy of LOC_LOCATOR is automatically assumed. | 107 |
| <code>setContext()</code> | O | Sets the context of the agent. An applet must set the context by issuing a <i>agent.setContext(this,null)</i> ; | 108 |
| <code>setLocator()</code> | I | Sets the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR. This call must precede the <i>Agent.setName()</i> . If LOC_LOCATOR is set, the default configuration ID for profile access is automatically used. Note: Java RMI agents should only be used for prototyping. They are currently not suited for production purposes. | 111 |
| <code>setName()</code> | C | Sets the name of the Java Agent. | 107 |
| <code>toString()</code> | C | Returns the name of the agent. | 107 |

BlockInstanceMonitor

A block instance monitor object represents a monitor of an activity instance of type *Block*.

Note: All API calls of a block instance monitor are also applicable to process instance monitors.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| destructor() | Destructs a block instance monitor object, that is, the transient representation in the C++ interface. The internal block instance monitor object is, however, not deallocated since it is part of the process instance monitor. It is deallocated when the process instance monitor is destructed/deallocated. | 75 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary because a block instance monitor is a part of a process instance monitor.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-----------------------------|------|---|------|
| ActivityInstances() | M | Returns the activity instances which are represented by the block instance monitor, that is, which are part of the activity instance of type <i>Block</i> . The activity instances contain both primary and secondary values. | 107 |
| ControlConnectorInstances() | M | Returns the control connector instances which are represented by the block instance monitor, that is, which are part of the activity instance of type <i>Block</i> . | 107 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------------------|--|------|
| ObtainBlockInstanceMonitor() | Returns the block instance monitor for an activity instance of type <i>Block</i> . The activity instance is part of the set of activity instances represented by the block instance monitor. | 282 |
| ObtainProcessInstanceMonitor() | Returns the process instance monitor for an activity instance of type <i>Process</i> . The activity instance is part of the set of activity instances represented by the block instance monitor. | 284 |
| Refresh() | Refreshes the block instance monitor from the MQSeries Workflow execution server. | 287 |

Container

A container represents an input or output data container of a process instance or work item.

Note: All API calls of a container are applicable to read-only and read/write containers.

API classes and objects

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| IsEmpty() | Indicates whether container information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|---------------------|------|---|------|
| AllLeafCount() | I | Returns the number of leaf elements of the container including the MQSeries Workflow predefined members. | 37 |
| AllLeaves() | M | Returns all leaf elements of the container including the MQSeries Workflow predefined members. | 37 |
| ArrayBinaryLength() | I | Returns the length of the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type BINARY. | 47 |
| ArrayBinaryValue() | C | Returns the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type BINARY. | 47 |
| ArrayFloatValue() | F | Returns the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type FLOAT. | 47 |
| ArrayLongValue() | I | Returns the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type LONG. | 47 |
| ArrayStringLength() | I | Returns the length of the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type STRING. | 47 |
| ArrayStringValue() | C | Returns the value of the specified container leaf element in C or COBOL. The leaf is part of an array and of type STRING. | 47 |
| BinaryLength() | I | Returns the length of the value of the specified container leaf element. The leaf is of type BINARY. | 47 |
| BinaryValue() | C | Returns the value of the specified container leaf element in C or COBOL. The leaf is of type BINARY. | 47 |
| FloatValue() | F | Returns the value of the specified container leaf element in C or COBOL. The leaf is of type FLOAT. | 47 |
| getBuffer() | C | Returns the value of the specified container leaf element in Java. The leaf is of type BINARY. | 47 |

API classes and objects

| Accessor methods | Type | Description | Page |
|------------------|---------|---|------|
| getBuffer2() | C | Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type BINARY. | 47 |
| getDouble() | F | Returns the value of the specified container leaf element in Java. The leaf is of type FLOAT. | 47 |
| getDouble2() | F | Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type FLOAT. | 47 |
| GetElement() | O | Provides access to a container element. | 47 |
| getLong() | I | Returns the value of the specified container leaf element in Java. The leaf is of type LONG. | 47 |
| getLong2() | C | Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type LONG. | 47 |
| getString() | C | Returns the value of the specified container leaf element in Java. The leaf is of type STRING. | 47 |
| getString2() | C | Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type STRING. | 47 |
| LeafCount() | I | Returns the number of user-defined leaf elements of the container. | 47 |
| Leaves() | M | Returns all user-defined leaf elements of the container. | 47 |
| LongValue() | I | Returns the value of the specified container leaf element in C or COBOL. The leaf is of type LONG. | 47 |
| MemberCount() | I | Returns the number of structural members in the container. | 47 |
| StringLength() | I | Returns the length of the value of the specified container leaf element in C or COBOL. The leaf is of type STRING. | 47 |
| StringValue() | C | Returns the value of the specified container leaf element in C or COBOL. The leaf is of type STRING. | 47 |
| StructMembers() | M | Returns the structural members of the container. | 37 |
| Type() | C | Returns the type of the container, that is, the data structure name. | 37 |
| Value() | C/I/F/N | Returns the value of the specified container leaf element in C++. | 47 |

Refer to “Activity implementation API calls” on page 121 for detailed descriptions of activity implementation API calls.

| Activity implementation methods | Description | Page |
|---------------------------------|--|------|
| InContainer() | Accesses the input container from within an activity implementation; for Java, see ExecutionAgent. | 289 |

API classes and objects

| Activity implementation methods | Description | Page |
|---------------------------------|--|------|
| OutContainer() | Accesses the output container from within an activity implementation; for Java, see ExecutionAgent. | 291 |
| RemoteInContainer() | Accesses the input container from within a program started by an activity implementation; for Java, see ExecutionAgent. | 293 |
| RemoteOutContainer() | Accesses the output container from within a program started by an activity implementation; for Java, see ExecutionAgent. | 294 |
| SetOutContainer() | Sets the output container from within an activity implementation; for Java, see ExecutionAgent. | 296 |
| SetRemoteOutContainer() | Sets the output container from within a program started by an activity implementation; for Java, see ExecutionAgent. | 298 |

ContainerElement

A container element represents an arbitrary element of a container.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a container element object. | 71 |
| Copy() | Allocates and initializes the storage for a container element object by copying. | 74 |
| Deallocate() | Deallocates the storage for a container element object. | 75 |
| destructor() | Destructs a container element object. | 75 |
| Equal() | Compares two container elements. | 73 |
| operator=() | Assigns a container element to another one. | 73 |
| operator==(()) | Compares two container elements. | 73 |
| IsEmpty() | Indicates whether container element information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties because a container element describes a part of a container.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|---------------------|------|---|------|
| ArrayBinaryLength() | I | Returns the length of the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type BINARY. | 54 |

| Accessor methods | Type | Description | Page |
|---------------------|------|---|------|
| ArrayBinaryValue() | C | Returns the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type BINARY. | 54 |
| ArrayElements() | M | Returns the array elements of the container element. | 41 |
| ArrayFloatValue() | F | Returns the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type FLOAT. | 54 |
| ArrayLongValue() | I | Returns the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type LONG. | 54 |
| ArrayStringLength() | I | Returns the length of the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type STRING. | 54 |
| ArrayStringValue() | C | Returns the value of the specified container element leaf element in C or COBOL. The leaf is part of an array and of type STRING. | 54 |
| BinaryLength() | I | Returns the length of the value of the specified container element leaf element. The leaf is of type BINARY. | 54 |
| BinaryValue() | C | Returns the value of the specified container element leaf element in C or COBOL. The leaf is of type BINARY. | 54 |
| Cardinality() | I | Returns the number of array elements of the container element. | 41 |
| FloatValue() | ? | Returns the value of the specified container element leaf element in C or COBOL. The leaf is of type FLOAT. | 54 |
| FullName() | C | Returns the fully-qualified dotted name of the container element. | 41 |
| getBuffer() | C | Returns the value of the specified container element leaf element in Java. The leaf is of type BINARY. | 47 |
| getBuffer2() | C | Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type BINARY. | 47 |
| getDouble() | F | Returns the value of the specified container element leaf element in Java. The leaf is of type FLOAT. | 47 |
| getDouble2() | F | Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type FLOAT. | 47 |
| GetElement() | O | Provides access to an element of the container element. | 47 |
| getLong() | I | Returns the value of the specified container element leaf element in Java. The leaf is of type LONG. | 47 |
| getLong2() | I | Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type LONG. | 47 |

API classes and objects

| Accessor methods | Type | Description | Page |
|------------------|---------|---|------|
| getString() | C | Returns the value of the specified container element leaf element in Java. The leaf is of type STRING. | 47 |
| getString2() | C | Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type STRING. | 47 |
| isArray() | B | Indicates whether the container element is an array. | 41 |
| isLeaf() | B | Indicates whether the container element is a leaf. | 41 |
| isStruct() | B | Indicates whether the container element is a structure itself. | 41 |
| leafCount() | I | Returns the number of leaf elements of the container element. | 41 |
| leaves() | M | Returns all leaf elements of the container element. | 41 |
| longValue() | I | Returns the value of the specified container element leaf element in C or COBOL. The leaf is of type LONG. | 54 |
| memberCount() | I | Returns the number of structural members in the container element. | 41 |
| name() | C | Returns the name of the container element. | 41 |
| stringLength() | I | Returns the length of the value of the specified container element leaf element in C or COBOL. The leaf is of type STRING. | 54 |
| stringValue() | C | Returns the value of the specified container element leaf element in C or COBOL. The leaf is of type STRING.. | 54 |
| structMembers() | M | Returns the structural members of the container element. | 41 |
| type() | C | Returns the type of the container element, that is, the data structure name. | 41 |
| value() | C/I/F/N | Returns the value of the specified container element leaf element in C++. | 54 |

ContainerElementVector

A container element vector represents the result of a query for container elements in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector functions.

| Vector methods | Description |
|----------------|---|
| Deallocate() | Deallocates a container element vector object. |
| FirstElement() | Returns the first element of the container element vector. |
| NextElement() | Returns the next element of the container element vector. |
| Size() | Returns the number of elements in the container element vector. |

ControlConnectorInstance

A control connector instance object represents a control connector between two activity instances and its state.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs a control connector instance object. | 71 |
| Copy() | Allocates and initializes the storage for a control connector instance object by copying. | 74 |
| Deallocate() | Deallocates the storage for a control connector instance object. | 75 |
| destructor() | Destructs a control connector instance object. | 75 |
| Equal() | Compares two control connector instance objects on the basis of their source and target activity instances. | 73 |
| IsEmpty() | Indicates whether control connector instance information is not available. | 76 |
| Kind() | States the kind of the control connector instance, whether it is a transition condition or the "otherwise" connector. | 76 |
| operator=() | Assigns a control connector instance object to this one. | 73 |
| operator==(()) | Compares two control connector instance objects on the basis of their source and target activity instances. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|---------------------------------|------|---|------|
| BendPoints() | M | Returns the bend points of the control connector instance. | 107 |
| Name() | C | Returns the name associated with the control connector instance. | 107 |
| NameIsNull() | B | Indicates whether a name is set. | 110 |
| PersistentOidOfSourceActivity() | C | Returns the object ID of the activity instance which is the source of this control connector instance. | 107 |
| PersistentOidOfTargetActivity() | C | Returns the object ID of the activity instance which is the target of this control connector instance. | 107 |
| State() | E | Returns the state of the control connector instance, whether it is evaluated, and the result of evaluation. | 87 |
| TransitionCondition() | C | Returns the transition condition of the control connector instance. | 107 |
| TransitionConditionIsNull() | B | Indicates whether a transition condition is set. | 110 |

API classes and objects

ControlConnectorInstanceVector

A control connector instance vector represents the result of a query for control connector instances in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|--|
| Deallocate() | Deallocates a control connector instance vector object. |
| FirstElement() | Returns the first element of the control connector instance vector. |
| NextElement() | Returns the next element of the control connector instance vector. |
| Size() | Returns the number of elements in the control connector instance vector. |

Date and Time (FmcDateTime/FmcjCDateTime/Calendar)

An FmcjDateTime object represents date and time values in C++. An FmcjCDateTime structure represents date and time values in C or COBOL. Java uses a Calendar object.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls. The following methods are only available in C++.

| Basic methods | Description | Page |
|-------------------|--|------|
| constructor() | Constructs a date/time object. | 71 |
| destructor() | Destructs a date/time object. | 75 |
| operator=() | Assigns a date/time object to another one. | 73 |
| operator==(()) | Compares two date/time objects. | 73 |
| operator string() | Returns the string representation of the date/time object. | 107 |
| IsEmpty() | Indicates whether date/time information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. Because a date/time object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

The following methods are only available in C++.

| Accessor methods | Type | Description | Page |
|------------------|------|--|------|
| Day() | I | Returns the day of the date/time object. | 106 |
| Hour() | I | Returns the hours of the date/time object. | 106 |
| Minute() | I | Returns the minutes of the date/time object. | 106 |
| Month() | I | Returns the month of the date/time object. | 106 |
| Second() | I | Returns the seconds of the date/time object. | 106 |

| Accessor methods | Type | Description | Page |
|------------------|------|---|------|
| Year() | I | Returns the year of the date/time object. | 106 |

The following functions are only available in C and COBOL.

| Accessor functions | Type | Description | Page |
|---------------------------|------|--|------|
| FmcjDateTimeAsString() | C | Returns the string representation of the date/time structure. | 107 |
| FmcjDateTimeCurrentTime() | D | Returns the current date/time. | 87 |
| FmcjDateTimeIsValid() | B | Indicates whether the passed date/time is a valid date/time; must be greater than or equal to 1970-1-1 12:00 (yyyy-mm-dd hh:mm). | 86 |

DllOptions

A DllOptions object represents the program implementation definitions for a dynamic link library.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| constructor() | Constructs a DLL options object. | 71 |
| Copy() | Allocates and initializes the storage for a DLL options object by copying. | 74 |
| Deallocate() | Deallocates the storage for a DLL options object. | 75 |
| destructor() | Destructs a DLL options object. | 75 |
| IsEmpty() | Indicates whether DLL options information is not available. | 76 |
| operator=() | Assigns a DLL options object to this one. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-----------------------|------|--|------|
| EntryPointName() | C | Returns the name of the entry point of the DLL. | 107 |
| ExecuteFenced() | B | States whether the DLL should run in a separate address space. | 86 |
| ExecuteFencedIsNull() | B | Indicates whether execute fenced is set. | 110 |
| KeepLoaded() | B | States whether the DLL should stay loaded. | 86 |
| KeepLoadedIsNull() | B | Indicates whether keep loaded is set. | 110 |
| PathAndFileName() | C | Returns the path and file name of the DLL. | 107 |

ExecutionAgent

An ExecutionAgent object (Java) represents an MQSeries Workflow program execution agent.

API classes and objects

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. Because the following information is retrieved from the program execution agent, a distinction between primary and secondary properties is not applicable.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-------------------------|-------------|--|-------------|
| ProgramID() | C | Returns the program identification by which the invoked activity implementation is known to the program execution agent. | 107 |
| RemoteUserID() | C | Returns the user identification on whose behalf the activity implementation which started this program was originally started. | 107 |
| UserID() | C | Returns the user identification on whose behalf the activity implementation was started. | 107 |

Refer to “Activity implementation API calls” on page 121 for detailed descriptions of activity implementation API calls.

| Activity implementation methods | Description | Page |
|--|--|-------------|
| InContainer() | Accesses the input container from within an activity implementation in Java. | 289 |
| OutContainer() | Accesses the output container from within an activity implementation in Java. | 291 |
| RemoteInContainer() | Accesses the input container from within a program started by an activity implementation in Java. | 293 |
| RemoteOutContainer() | Accesses the output container from within a program started by an activity implementation in Java. | 294 |
| SetOutContainer() | Sets the output container from within an activity implementation in Java. | 296 |
| SetRemoteOutContainer() | Sets the output container from within a program started by an activity implementation in Java. | 298 |

ExecutionData

An Execution data object represents data sent from an MQSeries Workflow execution server.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------------|--|-------------|
| constructor() | Constructs an Execution data object. | 71 |
| Copy() | Allocates and initializes the storage for an Execution data object by copying. | 74 |
| Deallocate() | Deallocates the storage for an execution data object. | 75 |
| destructor() | Destructs an execution data object. | 75 |

| Basic methods | Description | Page |
|---------------|---|------|
| IsEmpty() | Indicates whether execution data information is not available. | 76 |
| Kind() | Returns the kind of the data, whether it is describing a work item creation, deletion, and so on. | 76 |
| operator=() | Assigns an execution data object to this one. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|--|------|--|------|
| ActivityInstanceNotificationFromData() | P | Creates an activity instance notification from the execution data. | 110 |
| ErrorFromData() | P | Creates an error description object from the execution data. | 110 |
| IsError() | B | States whether the execution data describes an erroneous situation. | 86 |
| PersistentOid() | C | Returns a representation of the object ID of the object described by the execution data. | 107 |
| ProcessInstanceNotificationFromData() | P | Creates a process instance notification from the execution data. | 110 |
| ReadOnlyContainerFromData() | P | Creates a container object from the execution data. | 110 |
| WorkitemFromData() | P | Creates a work item from the execution data. | 110 |
| UserContext() | C | Returns the user context. | 107 |
| UserContextIsNull() | B | States whether a user context had been specified. | 110 |

ExecutionService

An ExecutionService object represents a user session to an execution server.

Note: All API calls provided by FmcjService are also applicable.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| Allocate() | Allocates the storage for an execution service object. The execution server to connect to is taken from the MQSeries Workflow user’s or configuration profile in the currently set configuration. | 71 |

API classes and objects

| Basic methods | Description | Page |
|---------------------|--|------|
| AllocateForSystem() | Allocates the storage for the specified execution service object. The execution server to connect to is taken from the specified parameters in the currently set configuration. | 71 |
| AllocateForGroup() | Allocates the storage for the specified execution service object. The execution server to connect to can be any system within the specified system group in the currently set configuration. | 71 |
| constructor() | Constructs an execution service object. | 71 |
| Copy() | Allocates and initializes the storage for an execution service object by copying. | 74 |
| Deallocate() | Deallocates the storage for an execution service object. | 75 |
| destructor() | Destructs an execution service object. | 75 |
| Equal() | Compares two execution service objects if they represent the same session. | 73 |
| operator=() | Assigns an execution service object to this one. | 73 |
| operator==(()) | Compares two execution service objects if they represent the same session. | 73 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|---|---|------|
| CreateProcessInstanceList() | Creates a new process instance list on the execution server. | 301 |
| CreateProcessTemplateList() | Creates a new process template list on the execution server. | 306 |
| CreateWorklist() | Creates a new worklist on the execution server. | 312 |
| Logoff() | Logs off from the connected execution server. | 319 |
| Logon() | Logs on to the execution server. | 320 |
| Logon2() | Logs on to the execution server in Java and provides additional parameters. | 320 |
| persistentActivityInstance Notification() | Retrieves the activity instance notification specified by the passed object identification in the Java API. | 278 |
| persistentProcessInstance() | Retrieves the process instance specified by the passed object identification in the Java API. | 412 |
| persistentProcessInstance Notification() | Retrieves the process instance notification specified by the passed object identification in the Java API. | 432 |
| persistentProcessTemplate() | Retrieves the process template specified by the passed object identification in the Java API. | 453 |
| persistentWorkItem() | Retrieves the work item specified by the passed object identification in the Java API. | 490 |

| Action methods | Description | Page |
|---------------------------------------|---|------|
| QueryActivityInstance Notifications() | Retrieves the activity instance notifications the logged-on user has access to. | 327 |
| QueryItems() | Retrieves the work items or notifications the logged-on user has access to. | 333 |
| QueryProcessInstanceLists() | Retrieves the process instance lists the logged-on user has access to. | 339 |
| QueryProcessInstance Notifications() | Retrieves the process instance notifications the logged-on user has access to. | 341 |
| QueryProcessInstances() | Retrieves the process instances the logged-on user has access to. | 347 |
| QueryProcessTemplateLists() | Retrieves the process template lists the logged-on user has access to. | 352 |
| QueryProcessTemplates() | Retrieves the process templates the logged-on user has access to. | 354 |
| QueryWorkitems() | Retrieves the work items the logged-on user has access to. | 358 |
| QueryWorklists() | Retrieves the worklists the logged-on user has access to. | 365 |
| Receive() | Receives execution data sent by an MQSeries Workflow execution server. | 367 |
| TerminateReceive() | Places information in the client input queue to indicate that receiving execution data sent by an MQSeries Workflow execution server can end. | 372 |

Refer to “Activity implementation API calls” on page 121 for detailed descriptions of activity implementation API calls.

| Activity implementation methods | Description | Page |
|---------------------------------|--|------|
| Passthrough() | Establishes a session between an activity implementation and an execution server. | 325 |
| RemotePassthrough() | Establishes a session between a program started by an activity implementation and an execution server. | 370 |

ExeOptions

An ExeOptions object represents the program implementation definitions for an executable.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs an EXE options object. | 71 |
| Copy() | Allocates and initializes the storage for an EXE options object by copying. | 74 |
| Deallocate() | Deallocates the storage for an EXE options object. | 75 |
| destructor() | Destructs an EXE options object. | 75 |
| operator=() | Assigns an EXE options object to this one. | 73 |

API classes and objects

| Basic methods | Description | Page |
|---------------|---|------|
| IsEmpty() | Indicates whether EXE options information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|------------------------------|------|--|------|
| AutomaticClose() | B | States whether the window in which the EXE starts should close when the EXE ends. | 86 |
| AutomaticCloseIsNull() | B | Indicates whether automatic close is set. | 110 |
| Environment() | C | States the environment settings for the EXE. | 107 |
| EnvironmentIsNull() | B | Indicates whether an environment is set. | 110 |
| InheritEnvironment() | B | States whether the environment settings should be merged with the operating system environment settings. | 86 |
| PathAndFileName() | C | Returns the path and file name of the EXE. | 107 |
| RunInXTerm() | B | States whether the EXE should start in a separate xterm. | 86 |
| RunInXTermIsNull() | B | Indicates whether run in xterm is set. | 110 |
| StartInForeground() | B | States whether the EXE should start in the foreground. | 86 |
| StartInForegroundIsNull() | B | Indicates whether start in foreground is set. | 110 |
| WindowStyle() | O | States the initial window style. | 108 |
| WindowStyleIsNull() | B | Indicates whether a window style is set. | 110 |
| WorkingDirectoryName() | C | States the working directory for the EXE. | 107 |
| WorkingDirectoryNameIsNull() | B | Indicates whether a working directory is set. | 110 |

ExternalOptions

An ExternalOptions object represents the program implementation definitions for an external service.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| constructor() | Constructs an External options object. | 71 |

| Basic methods | Description | Page |
|---------------|--|------|
| Copy() | Allocates and initializes the storage for an External options object by copying. | 74 |
| Deallocate() | Deallocates the storage for an External options object. | 75 |
| destructor() | Destructs an External options object. | 75 |
| operator=() | Assigns an External options object to this one. | 73 |
| IsEmpty() | Indicates whether External options information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-----------------------------------|------|--|------|
| BackwardMappingFormat() | C | Specifies the format of the mapping from the structure the executable uses to an MQSeries Workflow container. | 107 |
| BackwardMappingFormatIsNull() | B | Indicates whether a backward mapping format is set. | 110 |
| BackwardMappingParameters() | M | Returns backward mapping parameters, if any. | 107 |
| BackwardMappingParametersIsNull() | B | Indicates whether backward mapping parameters are set. | 110 |
| CodePage() | I | Specifies the code page of the service. | 106 |
| CodePageIsNull() | B | Indicates whether a code page is set. | 110 |
| ExecutableName() | C | Specifies the executable to be invoked by the invocation type and service. | 107 |
| ExecutableType() | C | Identifies the type of the executable. | 107 |
| ForwardMappingFormat() | C | Specifies the format for the mapping from an MQSeries Workflow container to the structure the executable uses. | 107 |
| ForwardMappingFormatIsNull() | B | Indicates whether a forward mapping format is set. | 110 |
| ForwardMappingParameters() | M | Returns forward mapping parameters, if any. | 107 |
| ForwardMappingParametersIsNull() | B | Indicates whether forward mapping parameters are set. | 110 |
| InvocationType() | C | Specifies the invocation mechanism to invoke the executable on the service. | 107 |

API classes and objects

| Accessor methods | Type | Description | Page |
|-------------------------|------|--|------|
| IsLocalUser() | B | Returns whether a local user is to be resolved instead of using the MQSeries Workflow user ID. | 86 |
| IsMappingRoutineCall() | B | Specifies whether forward and backward mapping routines are to be called. | 86 |
| IsSecurityRoutineCall() | B | Specifies whether a security routine is to be called. | 86 |
| MappingType() | C | Identifies the type of mapping that should occur. | 107 |
| MappingTypeIsNull() | B | Indicates whether a mapping type is set. | 110 |
| ServiceName() | C | Identifies the service that is to be called. | 107 |
| ServiceType() | C | Identifies the type of service to be called, for example, CICS(R) or IMS(TM). | 107 |
| TimeoutPeriod() | E | Specifies how long the program execution server should wait for a response from the started service: forever, a time period, or never. | 87 |
| TimeoutInterval() | I | Specifies the timeout interval. | 106 |
| TimeoutIntervalIsNull() | B | Indicates whether a timeout interval is set. | 110 |

FmcError/FmcjError

An FmcError or FmcjError object represents a description of the reason why a work item or activity instance is in state InError. It also describes an error returned as an asynchronous response.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs an Error object. | 71 |
| Copy() | Allocates and initializes the storage for an Error object by copying. | 74 |
| Deallocate() | Deallocates the storage for an Error object. | 75 |
| destructor() | Destructs an Error object. | 75 |
| Equal() | Compares two Error objects on the basis of their return codes and parameters. | 73 |
| IsEmpty() | Indicates whether Error information is not available. | 76 |
| operator=() | Assigns an Error object to this one. | 73 |
| operator==(()) | Compares two Error objects on the basis of their return codes and parameters. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|------------------|------|--|------|
| MessageText() | C | Returns the error as a formatted message (in the local language if applicable). | 107 |
| Parameters() | M | Returns the parameters of the error; these are to be incorporated into the message text. | 107 |
| Rc() | I | Returns the return code remembered in the error object. | 106 |

FmcException

An FmcException object represents a description of an exception thrown by Java.

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-------------------|------|--|------|
| MessageText() | C | Returns the exception as a formatted message (in the local language if applicable). | 107 |
| nestedException() | - | Returns an exception thrown by the communication layer. Note: The nested exception can be inspected by (down-)casting to either <code>org.omg.CORBA.SystemException</code> or to <code>java.rmi.RemoteException</code> depending on the communication protocol used. However, doing so will make the client code protocol-dependent (unless it deals with both cases). When using local bindings the nested exception will always be null. | 107 |
| origin() | C | Returns the module that threw the exception. | 107 |
| Parameters() | M | Returns the parameters of the error; these are already incorporated into the message text. | 107 |
| Rc() | I | Returns the return code remembered in the error object. | 106 |

Global

A Global object serves to group global MQSeries Workflow API calls.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description |
|---------------|--|
| Connect() | Initializes the API in the current thread. |

API classes and objects

| Basic methods | Description |
|---------------|--|
| Disconnect() | Deinitializes the API in the current thread. |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|----------------------|------|--|------|
| ConfigurationID() | C | Returns the configuration ID to be used for profile access. | 107 |
| SetConfigurationID() | C | Sets the configuration ID to be used for profile access. Can only be set before the first profile usage. | 107 |

ImplementationData

An ImplementationData object represents the program implementation definitions.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs an implementation data object. | 71 |
| Copy() | Allocates and initializes the storage for an implementation data object by copying. | 74 |
| Deallocate() | Deallocates the storage for an implementation data object. | 75 |
| destructor() | Destructs an implementation data object. | 75 |
| operator=() | Assigns an implementation data object to this one. | 73 |
| IsEmpty() | Indicates whether implementation data information is not available. | 76 |
| Kind() | States the actual kind of the implementation data, whether it is a DLL or an EXE. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-------------------------------|------|--|------|
| CommandLineParameters() | M | Returns the command line parameters to be passed to the invoked program. | 107 |
| CommandLineParametersIsNull() | B | Indicates whether command line parameters are set. | 110 |

| Accessor methods | Type | Description | Page |
|-------------------|------|--|------|
| DllOptions() | P | Returns the description of a DLL, if the implementation is a DLL. | 110 |
| ExeOptions() | P | Returns the description of an EXE, if the implementation is an EXE. | 110 |
| ExternalOptions() | P | Returns the description of external options, if the implementation is an external service. | 110 |
| options() | P | Returns the description of an EXE, a DLL, or an external service in Java. | 110 |
| Platform() | E | Returns the operating system platform this implementation data describes. | 87 |

Item

An item represents a work item, an activity instance notification, or a process instance notification.

Note: This means that all API calls of an item are also applicable to work items, activity instance notifications, and process instance notifications.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs an item object. | 71 |
| Copy() | Allocates and initializes the storage for an item object by copying. | 74 |
| Deallocate() | Deallocates the storage for an item object. | 75 |
| destructor() | Destructs an item object. | 75 |
| Equal() | Compares two items. | 73 |
| IsComplete() | Indicates whether the complete item information is available. | 75 |
| IsEmpty() | Indicates whether item information is not available. | 76 |
| Kind() | States the actual kind of the item, whether it is a work item or some kind of notification. | 76 |
| operator=() | Assigns an item to this one. | 73 |
| operator==(()) | Compares two items. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor and mutator API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific item.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|----------------------------------|--------------|---|------|
| Category() | P/C | Returns the process category of the item. | 107 |
| CategoryIsNull() | P/B | Indicates whether a category is set. | 110 |
| CreationTime() | P/D | Returns the creation time of the item. | 87 |
| Description() | P/C | Returns the description of the item. | 107 |
| DescriptionIsNull() | P/B | Indicates whether a description is set. | 110 |
| Documentation() | S/C | Returns the documentation of the item. | 107 |
| DocumentationIsNull() | S/B | Indicates whether a documentation is set. | 110 |
| EndTime() | S/D | Returns the ending time of the item. | 87 |
| EndTimeIsNull() | S/B | Indicates whether an end time is set. | 110 |
| Icon() | P/C | Returns the icon associated with the item. | 107 |
| InContainerName() | S/C | Returns the name of the input container of the item. | 107 |
| LastModificationTime() | P/D | Returns the last time a primary attribute of the item was changed. | 87 |
| Name() | P/C | Returns the name of the item. in C or COBOL, a work item or activity instance notification requires a buffer of at least 33 bytes, a process instance notification a buffer of at least 64 bytes. | 107 |
| OutContainerName() | S/C | Returns the name of the output container of the item. | 107 |
| Owner() | P/C | Returns the user ID of the owner of the item. | 107 |
| PersistentOid() | P/C | Returns a representation of the object identification of the item. | 107 |
| ProcessAdmin() | S/C | Returns the user ID of the process administrator of the item. | 107 |
| ProcessInstanceName() | P/C | Returns the name of the process instance the item is part of. | 107 |
| ProcessInstanceState() | P/E | Returns the state of the process instance the item is part of. | 87 |
| ProcessInstanceSystemGroupName() | S/C | Returns the name of the system group of the process instance the item is part of. | 107 |
| ProcessInstanceSystemName() | S/C | Returns the name of the system of the process instance the item is part of. | 107 |
| ReceivedAs() | P/E | Returns the reason why the item was received. | 87 |

| Accessor methods | Set/ Type | Description | Page |
|-------------------|--------------|--|------|
| ReceivedTime() | P/D | Returns the time when the item was received. | 87 |
| StartTime() | P/D | Returns the start time of the item. | 87 |
| StartTimeIsNull() | P/B | Indicates whether a start time is set. | 110 |

| Mutator methods | Description | Page |
|-----------------|--|------|
| Update() | Updates the item with the execution data sent by an MQSeries Workflow execution server. The object IDs of the item and of the object described by the execution data must match. | 113 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------------------|--|------|
| Delete() | Deletes an item. | 374 |
| ObtainProcessInstanceMonitor() | Retrieves the process instance monitor for the process instance the item is part of. | 376 |
| ProcessInstance() | Retrieves the process instance the item is part of. | 378 |
| Refresh() | Retrieves the complete information of the item. | 380 |
| SetDescription() | Sets the description of the item. | 382 |
| SetName() | Sets the name of the item. | 384 |
| Transfer() | Transfers an item to the specified user. | 386 |

ItemVector

An item vector represents the result of a query for items in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|--|
| Deallocate() | Deallocates an item vector object. |
| FirstElement() | Returns the first element of the item vector. |
| NextElement() | Returns the next element of the item vector. |
| Size() | Returns the number of elements in the item vector. |

Message

A message object serves to access the MQSeries Workflow provided message catalog.

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself. The API call declaration can be found in a general format at the

API classes and objects

indicated page.

| Accessor methods | Type | Description | Page |
|------------------|------|--|------|
| MessageText() | C | Returns a formatted message (in the local language if applicable) based on the message ID. Any parameters passed will be incorporated. | 107 |

PersistentList

A persistent list object represents a persistent list definition.

Note: All API calls of a persistent list are also applicable to process instance lists, process template lists, and worklists.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| IsEmpty() | Indicates whether persistent list information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|----------------------|------|--|------|
| Description() | C | Returns the description of the persistent list. | 107 |
| DescriptionIsNull() | B | Indicates whether a description is set. | 110 |
| Filter() | C | Returns the filter of the persistent list. | 107 |
| FilterIsNull() | B | Indicates whether a filter is set. | 110 |
| Name() | C | Returns the name of the persistent list. | 107 |
| OwnerOfList() | C | Returns the user ID of the owner of the persistent list. | 107 |
| OwnerOfListIsNull() | B | Indicates whether an owner is set; a public list does not have an owner. | 110 |
| SortCriteria() | C | Returns the sort criteria of the persistent list. | 107 |
| SortCriteriaIsNull() | B | Indicates whether sort criteria are set. | 110 |
| Threshold() | I | Returns the threshold of the persistent list. | 106 |
| ThresholdIsNull() | B | Indicates whether a threshold is set. | 110 |
| Type() | C | Returns the type of the persistent list, whether it is a public or private list. | 107 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|----------------|--------------------------------|------|
| Delete() | Deletes the persistent list. | 389 |
| Refresh() | Refreshes the persistent list. | 390 |

| Action methods | Description | Page |
|-------------------|--|------|
| setDescription() | Sets the description of the persistent list. | 392 |
| setFilter() | Sets the filter of the persistent list. | 394 |
| setSortCriteria() | Sets the sort criteria of the persistent list. | 396 |
| setThreshold() | Sets the threshold of the persistent list. | 398 |

Person

A person object represents the settings of the logged-on user.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs a person object. | 71 |
| copy() | Allocates and initializes the storage for a person object by copying. | 74 |
| deallocate() | Deallocates the storage for a person object. | 75 |
| destructor() | Destructs a person object. | 75 |
| equal() | Compares two persons. | 73 |
| operator=() | Assigns a person to this one. | 73 |
| operator==() | Compares two persons. | 73 |
| isComplete() | Indicates whether the complete person information is available. | 75 |
| isEmpty() | Indicates whether person information is not available. | 76 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when persons are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific person.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|---------------------------|--------------|--|------|
| CategoriesAuthorizedFor() | P/M | Returns the categories the person is authorized for with basic or with administration rights. If the person is authorized for all categories as administrator, no category is returned here. If the person is authorized for all categories with basic rights, categories authorized with administration rights are returned here. | 107 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--|--------------|---|------|
| CategoriesAuthorizedForAsAdmin() | P/M | Returns the categories the person is authorized for with administration rights. If the person is authorized for all categories with administration rights, no category is returned here. | 107 |
| Description() | P/C | Returns the description of the person. | 107 |
| DescriptionIsNull() | P/B | Indicates whether a description is set. | 110 |
| FirstName() | P/C | Returns the first name of the person. | 107 |
| FirstNameIsNull() | P/B | Indicates whether a first name is set. | 110 |
| IsAbsent() | P/B | Indicates whether the person is absent. | 86 |
| IsAdminForCategory() | P/B | Indicates whether the person has administrator rights for the specified category. Returns false if the category does not exist. If the person is authorized for all categories as administrator, then true is returned independent of the category existence. | 86 |
| IsAdministrator() | S/B | Indicates whether the person is an administrator. | 86 |
| IsAuthorizedForAllCategories() | P/B | Indicates whether the person is said to be authorized for all categories either with basic and/or administration rights. | 86 |
| IsAuthorizedForAllCategoriesAsAdmin() | P/B | Indicates whether the person is said to be authorized for all categories as administrator. | 86 |
| IsAuthorizedForAllPersons() | P/B | Indicates whether the person is authorized to see the items of all persons. | 86 |
| IsAuthorizedForAuthorizationDefinition() | P/B | Indicates whether the person is authorized to define authorizations. | 86 |
| IsAuthorizedForOperationAdministration() | P/B | Indicates whether the person is authorized for operational administrations. | 86 |
| IsAuthorizedForProcessDefinition() | P/B | Indicates whether the person is authorized to define process models. | 86 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|-------------------------------------|--------------|---|------|
| IsAuthorizedForStaffDefinition() | P/B | Indicates whether the person is authorized to define persons. | 86 |
| IsAuthorizedForTopologyDefinition() | P/B | Indicates whether the person is authorized to define topological data. | 86 |
| IsManager() | S/B | Indicates whether the person is a manager. | 86 |
| IsResetAbsence() | P/B | Indicates whether the absence flag should be reset when the person logs on. | 86 |
| LastName() | P/C | Returns the last name of the person. | 107 |
| LastNameIsNull() | P/B | Indicates whether a last name is set. | 110 |
| Level() | P/I | Returns the level of the person. | 106 |
| Manager() | S/C | Returns the user identification of the person's manager. | 107 |
| ManagerIsNull() | S/B | Indicates whether the person's manager is set. | 110 |
| MiddleName() | P/C | Returns the middle name of the person. | 107 |
| MiddleNameIsNull() | P/B | Indicates whether a middle name is set. | 110 |
| NamesOfManagedOrganizations() | S/M | Returns the names of organizations the person manages. | 107 |
| NamesOfRoles() | P/M | Returns the names of roles the person belongs to. | 107 |
| NamesOfRolesToCoordinate() | S/M | Returns the names of roles the person can coordinate. | 107 |
| OrganizationName() | P/C | Returns the name of the organization the person belongs to. | 107 |
| OrganizationNameIsNull() | P/B | Indicates whether an organization name is set. | 110 |
| PersonID() | P/C | Returns the person ID of the person. | 107 |
| PersonIDIsNull() | P/B | Indicates whether a person ID is set. | 110 |
| PersonsAuthorizedFor() | P/M | Returns the persons for whom this person is authorized either explicitly or by being a substitute. If the person is authorized for all other persons, then no person is returned here. | 107 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--------------------------|--------------|---|------|
| PersonsAuthorizedForMe() | S/M | Returns the persons who are authorized for this person. | 107 |
| PersonsToStandInFor() | S/M | Returns the persons for whom this person stands in. | 107 |
| Phone() | P/C | Returns the phone number of the person. | 107 |
| PhoneIsNull() | P/B | Indicates whether a phone is set. | 110 |
| SecondPhone() | P/C | Returns the alternate phone number of the person. | 107 |
| SecondPhoneIsNull() | P/B | Indicates whether an alternate phone is set. | 110 |
| Substitute() | P/C | Returns the substitute of the person. | 107 |
| SubstituteIsNull() | P/B | Indicates whether a substitute is set. | 110 |
| SystemName() | P/C | Returns the home system of the person. | 107 |
| UserID() | P/C | Returns the user identification of the person. | 107 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|-----------------|--|------|
| Refresh() | Retrieves the complete person information from the server. | 400 |
| SetAbsence() | Sets the absent flag of the logged-on user to the specified value. | 402 |
| SetSubstitute() | Sets the substitute of the logged-on user to the specified value. | 403 |

Point

A point object represents a bend point of a control connector.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a point object. | 71 |
| Copy() | Allocates and initializes the storage for a point object by copying. | 74 |
| Deallocate() | Deallocates the storage for a point object. | 75 |
| destructor() | Destructs a point object. | 75 |
| Equal() | Compares two point objects on the basis of their contents. | 73 |
| IsEmpty() | Indicates whether point information is not available. | 76 |
| operator=() | Assigns a point object to this one. | 73 |
| operator==(()) | Compares two point objects on the basis of their contents. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|------------------|------|--|------|
| XPosition() | I | Returns the x-coordinate of the point. | 106 |
| YPosition() | I | Returns the y-coordinate of the point. | 106 |

PointVector

A point vector represents the result of a query for points in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates a point vector object. |
| FirstElement() | Returns the first element of the point vector. |
| NextElement() | Returns the next element of the point vector. |
| Size() | Returns the number of elements in the point vector. |

ProcessInstance

A process instance object represents an instance of a workflow process template.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs a process instance object. | 71 |
| Copy() | Allocates and initializes the storage for a process instance object by copying. | 74 |
| Deallocate() | Deallocates the storage for a process instance object. | 75 |
| destructor() | Destructs a process instance object. | 75 |
| Equal() | Compares two process instances. | 73 |
| IsComplete() | Indicates whether the complete process instance information is available. | 75 |
| IsEmpty() | Indicates whether process instance information is not available. | 76 |
| operator=() | Assigns a process instance to this one. | 73 |
| operator==(()) | Compares two process instances. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific process instance.

API classes and objects

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|------------------------|--------------|--|------|
| AuditMode() | S/E | Returns the audit mode of the process instance. | 87 |
| Category() | P/C | Returns the category of the process instance. | 107 |
| CategoryIsNull() | P/B | Indicates whether a category is set. | 110 |
| CreationTime() | S/D | Returns the creation time of the process instance. | 87 |
| Creator() | S/C | Returns the creator of the process instance. | 107 |
| Description() | P/C | Returns the description of the process instance. | 107 |
| DescriptionIsNull() | P/B | Indicates whether a description is set. | 110 |
| Documentation() | S/C | Returns the documentation of the process instance. | 107 |
| DocumentationIsNull() | S/B | Indicates whether a documentation is set. | 110 |
| EndTime() | S/D | Returns the end time of the process instance. | 87 |
| EndTimeIsNull() | S/B | Indicates whether an end time is set. | 110 |
| Icon() | P/C | Returns the icon associated with the process instance. | 107 |
| InContainerName() | S/C | Returns the name of the input container of the process instance. | 107 |
| InContainerNeeded() | P/B | Indicates whether an input container is needed to start the process instance. An input container is needed when <ul style="list-style-type: none"> • There is a mapping to some other container. • Staff assignment data is taken from it. • Notification related data is taken from it. • A transaction or exit condition refers to a container element. • A description refers to a container element. • <i>Prompt for data at process start</i> is set for the process model. | 86 |
| LastModificationTime() | P/D | Returns the last time a primary attribute of the process instance was changed. | 87 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--------------------------|--------------|--|------|
| LastStateChangeTime() | P/D | Returns the last time the state of the process instance was changed. | 87 |
| Name() | P/C | Returns the name of the process instance. | 107 |
| NotificationTime() | S/D | Returns the notification time of the process instance. | 87 |
| NotificationTimeIsNull() | S/B | Indicates whether a notification time is set. | 110 |
| NotifiedPerson() | S/C | Returns the person who received the notification. | 107 |
| NotifiedPersonIsNull() | S/B | Indicates whether a notified person is set. | 110 |
| OrganizationName() | S/C | Returns the name of the organization of the process instance. | 107 |
| OrganizationNameIsNull() | S/B | Indicates whether an organization name is set. | 110 |
| OutContainerName() | S/C | Returns the name of the output container of the process instance. | 107 |
| ParentName() | P/C | Returns the name of the parent process instance of this process instance. | 107 |
| ParentNameIsNull() | P/B | Indicates whether a parent name is set. | 110 |
| PersistentOid() | P/C | Returns a representation of the object identification of the process instance. | 107 |
| ProcessAdmin() | S/C | Returns the user ID of the process administrator of the process instance. | 107 |
| ProcessAdminIsNull() | S/B | Indicates whether a process administrator is set. | 110 |
| ProcessTemplateName() | P/C | Returns the name of the process template the process instance is derived from. | 107 |
| RoleName() | S/C | Returns the name of the role of the process instance. | 107 |
| RoleNameIsNull() | S/B | Indicates whether a role is set. | 110 |
| Starter() | S/C | Returns the starter of the process instance. | 107 |
| StarterIsNull() | S/B | Indicates whether a starter is set. | 110 |
| StartTime() | S/D | Returns the start time of the process instance. | 87 |
| StartTimeIsNull() | S/B | Indicates whether a start time is set. | 110 |
| State() | P/E | Returns the state of the process instance. | 87 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|----------------------------------|--------------|--|------|
| StateOfNotification() | S/E | Returns the notification state of the process instance. | 87 |
| SuspensionExpirationTime() | P/D | Returns the suspension expiration time of the process instance. | 87 |
| SuspensionExpirationTimeIsNull() | P/B | Indicates whether the suspension expiration time is set. | 110 |
| SuspensionTime() | P/D | Returns the time the process instance was suspended. | 87 |
| SuspensionTimeIsNull() | P/B | Indicates whether the suspension time is set. | 110 |
| SystemGroupName() | P/C | Returns the name of the system group where the process instance runs. | 107 |
| SystemName() | P/C | Returns the name of the system where the process instance runs. | 107 |
| TopLevelName() | P/C | Returns the name of the top level process instance of this process instance. | 107 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------|---|------|
| Delete() | Deletes the process instance. | 406 |
| InContainer() | Retrieves the input container of the process instance. | 407 |
| ObtainMonitor() | Retrieves the process instance monitor for the process instance. | 410 |
| PersistentObject() | Retrieves the process instance specified by the passed object identification. | 412 |
| Refresh() | Retrieves the complete information of the process instance. | 414 |
| Restart() | Restarts the process instance. | 416 |
| Resume() | Resumes the execution of a suspended process instance. | 417 |
| SetDescription() | Sets the description of the process instance. | 419 |
| SetName() | Sets the name of the process instance. | 421 |
| Start() | Starts the process instance. | 423 |
| Start2() | Starts the process instance in Java and provides an input container. | 423 |
| Suspend() | Suspends the process instance. | 425 |
| Suspend2() | Suspends the process instance in Java until the specified calendar date. | 425 |
| SuspendUntil() | Suspends the process instance until the specified time. | 425 |
| Terminate() | Terminates the process instance. | 427 |

ProcessInstanceList

A process instance list represents a group of process instances.

Note: All API calls of a persistent list are also applicable to process instance lists.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a process instance list object. | 71 |
| Copy() | Allocates and initializes the storage for a process instance list object by copying. | 74 |
| Deallocate() | Deallocates the storage for a process instance list object. | 75 |
| destructor() | Destructs a process instance list object. | 75 |
| Equal() | Compares two process instance lists. | 73 |
| operator=() | Assigns a process instance list to this one. | 73 |
| operator==(()) | Compares two process instance lists. | 73 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|-------------------------|---|------|
| QueryProcessInstances() | Retrieves the process instances qualifying via the process instance list. | 429 |

ProcessInstanceListVector

A process instance list vector represents the result of a query for process instance lists in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates a process instance list vector object. |
| FirstElement() | Returns the first element of the process instance list vector. |
| NextElement() | Returns the next element of the process instance list vector. |
| Size() | Returns the number of elements in the process instance list vector. |

ProcessInstanceMonitor

A process instance monitor object represents a monitor of a process instance.

Note: All API calls of FmcjBlockInstanceMonitor are also applicable to process instance monitors.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| Deallocate() | Deallocates the storage for a process instance monitor object. All block instance monitors contained are also deallocated. | 75 |

API classes and objects

| Basic methods | Description | Page |
|---------------|---|------|
| destructor() | Destructs a process instance monitor object. All block instance monitors contained are also destructed. | 75 |

ProcessInstanceNotification

A process instance notification represents a notification raised for a process instance.

Note: All API calls for FmcjItem are also applicable to process instance notifications.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| constructor() | Constructs a process instance notification object. | 71 |
| Copy() | Allocates and initializes the storage for a process instance notification object by copying. | 74 |
| Deallocate() | Deallocates the storage for a process instance notification object. | 75 |
| destructor() | Destructs a process instance notification object. | 75 |
| Kind() | In C++, states that the object is a process instance notification. | 76 |
| operator=() | Assigns a process instance notification to this one. | 73 |
| operator==() | Compares two process instance notifications. | 73 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------|--|------|
| PersistentObject() | Retrieves the specified process instance notification. | 432 |

ProcessInstanceNotificationVector

A process instance notification vector represents the result of a query for process instance notifications in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates a process instance notification vector object. |
| FirstElement() | Returns the first element of the process instance notification vector. |
| NextElement() | Returns the next element of the process instance notification vector. |
| Size() | Returns the number of elements in the process instance notification vector. |

ProcessInstanceVector

A process instance vector represents the result of a query for process instances in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|--|
| Deallocate() | Deallocates the storage for a process instance vector object. |
| FirstElement() | Returns the first element of the process instance vector. |
| NextElement() | Returns the next element of the process instance vector. |
| Size() | Returns the number of elements in the process instance vector. |

ProcessTemplate

A process template object represents the Runtime equivalent of a Buildtime workflow process model.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs a process template object. | 71 |
| Copy() | Allocates and initializes the storage for a process template object by copying. | 74 |
| Deallocate() | Deallocates the storage for a process template object. | 75 |
| destructor() | Destructs a process template object. | 75 |
| Equal() | Compares two process templates. | 73 |
| IsComplete() | Indicates whether the complete process template information is available. | 75 |
| IsEmpty() | Indicates whether process template information is not available. | 76 |
| operator=() | Assigns a process template to this one. | 73 |
| operator==(()) | Compares two process templates. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process templates are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific process template.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|------------------|--------------|---|------|
| AuditMode() | S/E | Returns the audit mode of the process template. | 87 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--------------------------|--------------|---|------|
| Category() | P/C | Returns the category of the process template. | 107 |
| CategoryIsNull() | P/B | Indicates whether a category is set. | 110 |
| CreationTime() | P/D | Returns the creation time of the process template. | 87 |
| Description() | P/C | Returns the description of the process template. | 107 |
| DescriptionIsNull() | P/B | Indicates whether a description is set. | 110 |
| Documentation() | S/C | Returns the documentation of the process template. | 107 |
| DocumentationIsNull() | S/B | Indicates whether a documentation is set. | 110 |
| Icon() | P/C | Returns the icon associated with the process template. | 107 |
| InContainerName() | S/C | Returns the name of the input container of the process template. | 107 |
| InContainerNeeded() | P/B | Indicates whether an input container is needed to start an instance of the process template. An input container is needed when <ul style="list-style-type: none"> • There is a mapping to some other container. • Staff assignment data is taken from it. • Notification related data is taken from it. • A transaction or exit condition refers to a container element. • A description refers to a container element. • <i>Prompt for data at process start</i> is set for the process model. | 86 |
| LastModificationTime() | P/D | Returns the last time a primary attribute of the process template was changed. | 87 |
| Name() | P/C | Returns the name of the process template. | 107 |
| OrganizationName() | S/C | Returns the name of the organization of the process template. | 107 |
| OrganizationNameIsNull() | S/B | Indicates whether an organization name is set. | 110 |
| OutContainerName() | S/C | Returns the name of the output container of the process template. | 107 |
| PersistentOid() | P/C | Returns a representation of the object identification of the process template. | 107 |
| ProcessAdmin() | S/C | Returns the user ID of the process administrator of an instance of the process template. | 107 |
| ProcessAdminIsNull() | S/B | Indicates whether a process administrator is set. | 110 |
| RoleName() | S/C | Returns the name of the role of the process template. | 107 |
| RoleNameIsNull() | S/B | Indicates whether a role is set. | 110 |

| Accessor methods | Set/ Type | Description | Page |
|------------------|--------------|---|------|
| ValidFromTime() | P/D | Returns the time when the process template becomes valid. | 87 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|-------------------------------|---|------|
| CreateAndStartInstance() | Creates and starts an instance of the process template. | 434 |
| CreateAndStartInstance2() | Creates and starts an instance of the process template in Java and provides an input container. | 434 |
| CreateInstance() | Creates an instance of the process template. | 439 |
| Delete() | Deletes the specified process template. | 442 |
| Delete2() | Deletes the specified process template versions in Java. | 442 |
| ExecuteProcessInstance() | Creates and executes an instance from the specified process template. | 444 |
| ExecuteProcessInstanceAsync() | Creates and executes an instance from the specified process template without waiting for an answer. | 444 |
| InitialInContainer() | Retrieves the initially defined input container of the process template. | 451 |
| PersistentObject() | Retrieves the process template specified by the passed object identification. | 453 |
| ProgramTemplate() | Retrieves the program template specified by the passed name. | 455 |
| Refresh() | Retrieves the complete information of the process template. | 457 |

ProcessTemplateList

A process template list represents a group of process templates.

Note: All API calls of a persistent list are also applicable to process template lists.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a process template list object. | 71 |
| Copy() | Allocates and initializes the storage for a process template list object by copying. | 74 |
| Deallocate() | Deallocates the storage for a process template list object. | 75 |
| Equal() | Compares two process template lists. | 73 |
| destructor() | Destructs a process template list object. | 75 |
| operator=() | Assigns a process template list to this one. | 73 |
| operator==(()) | Compares two process template lists. | 73 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

API classes and objects

| Action methods | Description | Page |
|-------------------------|---|------|
| QueryProcessTemplates() | Retrieves the process templates qualifying via the process template list. | 459 |

ProcessTemplateListVector

A process template list vector represents the result of a query for process template lists in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates a process template list vector object. |
| FirstElement() | Returns the first element of the process template list vector. |
| NextElement() | Returns the next element of the process template list vector. |
| Size() | Returns the number of elements in the process template list vector. |

ProcessTemplateVector

A process template vector represents the result of a query for process templates in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|--|
| Deallocate() | Deallocates the storage for a process template vector object. |
| FirstElement() | Returns the first element of the process template vector. |
| NextElement() | Returns the next element of the process template vector. |
| Size() | Returns the number of elements in the process template vector. |

ProgramData

A program data object represents the program implementation definitions. In C++, it privately inherits from ProgramTemplate.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs a program data object. | 71 |
| Copy() | Allocates and initializes the storage for a program data object by copying. | 74 |
| Deallocate() | Deallocates the storage for a program data object. | 75 |
| destructor() | Destructs a program data object. | 75 |
| Equal() | Compares two program data objects if they belong to the same work item. | 73 |

| Basic methods | Description | Page |
|----------------|---|------|
| IsEmpty() | Indicates whether program data information is not yet available. | 76 |
| operator=() | Assigns a program data object to this one. | 73 |
| operator==(()) | Compares two program data objects if they belong to the same work item. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|---------------------|------|---|------|
| Description() | C | Returns the description of the implementing program. | 107 |
| DescriptionIsNull() | B | Indicates whether a description is set. | 110 |
| ExecutionMode() | E | States whether the program can participate in global transactions or not. | 87 |
| ExecutionUser() | C | Returns the user on whose behalf the program is to be executed. | 107 |
| Icon() | C | Returns the icon associated with the implementing program. | 107 |
| Implementations() | M | Returns the implementation definitions of the program. | 107 |
| InContainer() | P | Returns the input container of the program. | 110 |
| IsUnattended() | B | States whether the program can run unattended. | 86 |
| OutContainer() | P | Returns the output container of the program. | 110 |
| ProgramTrusted() | B | States whether the program can be trusted. Only a trusted program can receive its program ID. | 86 |

ProgramTemplate

A program template object represents the program implementation definitions.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| constructor() | Constructs a program template object. | 71 |
| Copy() | Allocates and initializes the storage for a program template object by copying. | 74 |
| Deallocate() | Deallocates the storage for a program template object. | 75 |
| destructor() | Destructs a program template object. | 75 |
| Equal() | Compares two program template objects on the basis of their names and the process template they belong to. | 73 |

API classes and objects

| Basic methods | Description | Page |
|----------------|--|------|
| IsEmpty() | Indicates whether program template information is not yet available. | 76 |
| operator=() | Assigns a program template object to this one. | 73 |
| operator==(()) | Compares two program template objects on the basis of their names and the process template they belong to. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|--------------------------|------|---|------|
| Description() | C | Returns the description of the implementing program. | 107 |
| DescriptionIsNull() | B | Indicates whether a description is set. | 110 |
| ExecutionMode() | E | States whether the program can participate in global transactions or not. | 87 |
| ExecutionUser() | C | Returns the user on whose behalf the program is to be executed. | 107 |
| Icon() | C | Returns the icon associated with the implementing program. | 107 |
| Implementations() | M | Returns the implementation definitions of the program. | 107 |
| InitialInContainer() | P | Returns the initially defined input container of the program. | 110 |
| InContainerAccess() | B | States whether the input container is accessed by the program. | 86 |
| IsUnattended() | B | States whether the program can run unattended. | 86 |
| InitialOutContainer() | P | Returns the initially defined output container of the program. | 110 |
| OutContainerAccess() | B | States whether the output container is accessed by the program. | 86 |
| ProgramTrusted() | B | States whether the program can be trusted. Only a trusted program can receive its program ID. | 86 |
| StructuresFromActivity() | B | States whether the program can handle any container passed to it. | 86 |
| ValidFromTime() | P/D | Returns the time when the process template and thus the program template becomes valid. | 87 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|---------------------------|---|------|
| Execute() | Requests the execution of the program by the system's program execution server. | 462 |
| Execute2() | Requests the execution of the program by the system's program execution server. | 462 |
| ExecuteWithOptions() | Requests the execution of the program by the system's program execution server. The priority of the program can be specified. | 462 |
| ExecuteAsync() | Requests the execution of the program by the system's program execution server without waiting for an answer. | 462 |
| ExecuteAsync2() | Requests the execution of the program by the system's program execution server without waiting for an answer. | 462 |
| ExecuteAsyncWithOptions() | Requests the execution of the program by the system's program execution server without waiting for an answer. The priority of the program can be specified. | 462 |

ReadOnlyContainer

A read-only container represents an input data container of a work item.

Note: All API calls of a container are applicable to read-only containers.

Refer to "Basic API calls" on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a read-only container object. | 71 |
| Copy() | Allocates and initializes the storage for a read-only container object by copying. | 74 |
| Deallocate() | Deallocates the storage for a read-only container object. | 75 |
| Equal() | Compares two read-only containers. | 73 |
| destructor() | Destructs a read-only container object. | 75 |
| operator=() | Assigns a read-only container to this one. | 73 |
| operator==(()) | Compares two read-only containers. | 73 |

ReadWriteContainer

A read/write container represents an input container of a process instance or an output container of a work item.

Note: All API calls of a container are applicable to read/write containers.

Refer to "Basic API calls" on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|---|------|
| constructor() | Constructs a read/write container object. | 71 |
| Copy() | Allocates and initializes the storage for a read/write container object by copying. | 74 |
| Deallocate() | Deallocates the storage for a read/write container object. | 75 |
| Equal() | Compares two read/write containers. | 73 |

API classes and objects

| Basic methods | Description | Page |
|---------------|--|------|
| destructor() | Destructs a read/write container object. | 75 |
| operator=() | Assigns a read/write container to another one. | 73 |
| operator==() | Compares two read/write containers. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

The value in the **Type** column states the type of the property set, whether it is a binary (N), a character string (C), a float (F), or an integer (I). The API call declaration can be found at the indicated page.

| Accessor methods | Type | Description | Page |
|-----------------------|------|--|------|
| SetArrayBinaryValue() | N | Sets the value of the specified container leaf element in C or COBOL. The leaf element is part of an array and of type BINARY. | 60 |
| SetArrayFloatValue() | F | Sets the value of the specified container leaf element in C or COBOL. The leaf element is part of an array and of type FLOAT. | 60 |
| SetArrayLongValue() | I | Sets the value of the specified container leaf element in C or COBOL. The leaf element is part of an array and of type LONG. | 60 |
| SetArrayStringValue() | C | Sets the value of the specified container leaf element in C or COBOL. The leaf element is part of an array and of type STRING. | 60 |
| SetBinaryValue() | N | Sets the value of the specified container leaf element in C or COBOL. The leaf element is of type BINARY. | 60 |
| SetBuffer() | N | Sets the value of the specified container leaf element in Java. The leaf element is of type BINARY. | 60 |
| SetBuffer2() | N | Sets the value of the specified container leaf element in Java. The leaf element is part of an array and of type BINARY. | 60 |
| SetDouble() | F | Sets the value of the specified container leaf element in Java. The leaf element is of type FLOAT. | 60 |
| SetDouble2() | F | Sets the value of the specified container leaf element in Java. The leaf element is part of an array and of type FLOAT. | 60 |
| SetFloatValue() | F | Sets the value of the specified container leaf element in C or COBOL. The leaf element is of type FLOAT. | 60 |
| SetLong() | I | Sets the value of the specified container leaf element in Java. The leaf element is of type LONG. | 60 |
| SetLong2() | I | Sets the value of the specified container leaf element in Java. The leaf element is part of an array and of type LONG. | 60 |
| SetLongValue() | I | Sets the value of the specified container leaf element in C or COBOL. The leaf element is of type LONG. | 60 |

| Accessor methods | Type | Description | Page |
|------------------|---------|--|------|
| SetString() | N | Sets the value of the specified container leaf element in Java. The leaf element is of type STRING. | 60 |
| SetString2() | N | Sets the value of the specified container leaf element in Java. The leaf element is part of an array and of type STRING. | 60 |
| SetStringValue() | C | Sets the value of the specified container leaf element in C or COBOL. The leaf element is of type STRING. | 60 |
| SetValue() | N/F/C/I | Sets the value of the specified container leaf element in C++. | 60 |

Result

A result object represents the result of an API call in C++, C, or COBOL.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| destructor | Destructs the C++ representation of the result object. | 75 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. Because a result object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-------------------------|------|---|------|
| MessageText() | C | Returns the result as a formatted message (in the local language if applicable). | 107 |
| ObjectOfCurrentThread() | P | Returns the result object associated with the thread from where this API call is called. | |
| Origin() | C | Returns the origin of the result, that is, file, line, function. | 107 |
| Parameters() | M | Returns the parameters of the result; these are already incorporated in the message text. | 107 |
| Rc() | I | Returns the return code remembered in the result object. | 106 |

Service

A service object represents common aspects of MQSeries Workflow service objects.

Note: All API calls of a service object are also applicable to execution services.

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. Because a service object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

API classes and objects

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|-------------------------|-------------|---|-------------|
| IsLoggedIn() | B | Indicates whether a user is logged on via this service object. | 86 |
| SetTimeout() | I | Sets the time the client will wait for a server to answer. The time is to be specified in milliseconds. | 111 |
| SystemGroupName() | C | Returns the name of the system group where the server resides. | 107 |
| SystemName() | C | Returns the name of the system where the server resides. | 107 |
| Timeout() | I | Returns the time the client will wait for a server to answer. | 106 |
| UserID() | C | Returns the user identification of the logged-on user. | 107 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|-----------------------|---|-------------|
| Refresh() | Refreshes information from the server, especially the logged-on status. | 466 |
| SetPassword() | Sets the password of the logged-on user. | 467 |
| UserSettings() | Retrieves the user settings of the logged-on user. | 469 |

StringVector

in C or COBOL, a string vector serves to represent a set of string information. For example, a string vector is returned to show the categories the logged-on user is authorized for. Or, a string vector must be used to specify the persons to stand in for.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|--------------------------|---|
| AddElement() | Adds a string to the string vector. |
| Allocate() | Allocates the storage for a string vector. |
| Deallocate() | Deallocates the storage for a string vector. |
| FirstElement() | Returns the first element of the string vector. |
| FirstResultParmElement() | Returns the first element of a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read. |

| Accessor methods | Description |
|-------------------------|--|
| NextElement() | Returns the next element of the string vector. |
| NextResultParmElement() | Returns the next element of a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read. |
| RemoveElement() | Removes a string from the string vector. |
| ResultParmDeallocate() | Deallocates the storage for a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read. |
| ResultParmSize() | Returns the number of elements in a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read. |
| Size() | Returns the number of elements in the string vector. |

SymbolLayout

A symbol layout object represents graphical information of a named icon.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|---------------|--|------|
| constructor() | Constructs a symbol layout object. | 71 |
| Copy() | Allocates and initializes the storage for a symbol layout object by copying. | 74 |
| Deallocate() | Deallocates the storage for a symbol layout object. | 75 |
| destructor() | Destructs a symbol layout object. | 75 |
| Equal() | Compares two symbol layout objects on the basis of their contents. | 73 |
| IsEmpty() | Indicates whether symbol layout information is not available. | 76 |
| operator=() | Assigns a symbol layout object to this one. | 73 |
| operator==() | Compares two symbol layout objects on the basis of their contents. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|------------------|------|---|------|
| XPosition() | I | Returns the x-coordinate of the named icon. | 106 |

API classes and objects

| Accessor methods | Type | Description | Page |
|-------------------|------|--|------|
| XPositionOfName() | I | Returns the x-coordinate of the name associated with the icon. | 106 |
| YPosition() | I | Returns the y-coordinate of the named icon. | 106 |
| YPositionOfName() | I | Returns the y-coordinate of the name associated with the icon. | 106 |

WorkItem

A work item represents an activity instance assigned to a user in order to be worked on.

Note: All API calls of an Item are also applicable to work items.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|--|------|
| constructor() | Constructs a work item object. | 71 |
| Copy() | Allocates and initializes the storage for a work item object by copying. | 74 |
| Deallocate() | Deallocates the storage for a work item object. | 75 |
| destructor() | Destructs a work item object. | 75 |
| Kind() | In C++, states that the object is a work item. | 76 |
| operator=() | Assigns a work item to this one. | 73 |
| operator==(()) | Compares two work items. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when work items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific work item.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), multi-valued property (M), a pointer to some object (P), or an object itself (O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Set/ Type | Description | Page |
|---------------------|--------------|---|------|
| ActivityKind() | P/E | Returns the kind of the associated activity instance, whether it is a program or process and so on. | 87 |
| ErrorReason() | S/O | Returns an error object describing the reason why the associated activity instance is in state InError. | 108 |
| ErrorReasonIsNull() | S/B | Indicates whether an error reason is set. | 110 |
| ExitCondition() | S/C | Returns the exit condition of the work item. | 107 |

API classes and objects

| Accessor methods | Set/ Type | Description | Page |
|--------------------------------|--------------|---|------|
| FirstNotificationTime() | S/D | Returns the time the first notification for the work item is to occur or has occurred. | 87 |
| FirstNotificationTimeIsNull() | S/B | Indicates whether a first notification time is set. | 110 |
| Implementation() | P/C | Returns the name of the implementing program of the associated activity instance. | 107 |
| ImplementationIsNull() | P/B | Indicates whether an implementation is set. | 110 |
| ManualExitMode() | S/B | Returns whether the exit mode of the work item is manual. | 86 |
| ManualStartMode() | S/B | Returns whether the start mode of the work item is manual. | 86 |
| Priority() | P/I | Returns the priority of the work item. | 106 |
| SecondNotificationTime() | S/D | Returns the time the second notification for the work item is to occur or has occurred. | 87 |
| SecondNotificationTimeIsNull() | S/B | Indicates whether a second notification time is set. | 110 |
| Staff() | S/M | Returns all persons owning a work item for the associated activity instance. | 107 |
| StartCondition() | S/C | Returns the start condition of the work item. | 107 |
| State | P/E | Returns the state of the work item. | 87 |
| StateOfNotification() | S/E | Returns the notification state of the work item. | 87 |
| SupportTools() | P/M | Returns the support tools associated with the work item. | 107 |
| SupportToolsIsNull() | P/B | Indicates whether support tools are set. | 110 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------|--|------|
| CancelCheckOut() | Cancels the checkout of the work item. | 473 |
| CheckIn() | Checks in the work item. | 475 |
| CheckOut() | Checks out the work item. | 477 |
| Finish() | Finishes a manual exit work item. | 481 |
| ForceFinish() | Force finishes the work item. | 483 |
| ForceRestart() | Force restarts the work item. | 485 |
| InContainer() | Retrieves the input container of the work item. | 486 |
| OutContainer() | Retrieves the output container of the work item. | 488 |
| PersistentObject() | Retrieves the specified work item. | 490 |

API classes and objects

| Action methods | Description | Page |
|----------------|------------------------------------|------|
| Restart() | Restarts the work item. | 492 |
| Start() | Starts the work item. | 494 |
| StartTool() | Starts the specified support tool. | 495 |
| Terminate() | Terminates the work item. | 497 |

WorkItemVector

A work item vector represents the result of a query for work items in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

| Accessor methods | Description |
|------------------|---|
| Deallocate() | Deallocates the storage for a work item vector object. |
| FirstElement() | Returns the first element of the work item vector. |
| NextElement() | Returns the next element of the work item vector. |
| Size() | Returns the number of elements in the work item vector. |

Worklist

A worklist represents a group of items.

Note: All API calls of a persistent list are also applicable to worklists.

Refer to “Basic API calls” on page 70 for detailed descriptions of basic API calls.

| Basic methods | Description | Page |
|----------------|---|------|
| constructor() | Constructs a worklist object. | 71 |
| Copy() | Allocates and initializes the storage for a worklist object by copying. | 74 |
| Deallocate() | Deallocates the storage for a worklist object. | 75 |
| destructor() | Destructs a worklist object. | 75 |
| Equal() | Compares two worklists. | 73 |
| operator=() | Assigns a worklist to another one. | 73 |
| operator==(()) | Compares two worklists. | 73 |

Refer to “Accessor API calls” on page 84 for detailed descriptions of accessor API calls. All properties are primary properties.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself. The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Type | Description | Page |
|------------------|------|--|------|
| BeepOption() | B | Indicates whether a beep should sound when the contents of the worklist changes. | 86 |

Refer to “Action API calls” on page 121 for detailed descriptions of action API calls.

| Action methods | Description | Page |
|--------------------------------------|--|------|
| QueryActivityInstanceNotifications() | Retrieves the activity instance notifications qualifying via the worklist. | 500 |
| QueryItems() | Retrieves all items qualifying via the worklist. | 502 |
| QueryProcessInstanceNotifications() | Retrieves the process instance notifications qualifying via the worklist. | 505 |
| QueryWorkitems() | Retrieves the work items qualifying via the worklist. | 507 |

WorklistVector

A worklist vector represents the result of a query for worklists in C or COBOL.

Refer to “C and COBOL vector accessor functions” on page 21 for detailed descriptions of vector access functions.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), or a multi-valued property (M), a pointer to some object (P), or an object itself(O). The API call declaration can be found in a general format at the indicated page.

| Accessor methods | Description |
|------------------|--|
| Deallocate() | Deallocates a worklist vector object. |
| FirstElement() | Returns the first element of the worklist vector. |
| NextElement() | Returns the next element of the worklist vector. |
| Size() | Returns the number of elements in the worklist vector. |

API classes and objects

Chapter 5. API action and activity implementation calls

The following chapter describes the MQSeries Workflow action and activity implementation API calls in alphabetical order by class.

Each entry contains a functional description of the API call followed by subsections:

Usage notes Points to general information about the nature of this call.

Authorization States the authority required to have the API call executed.

Required connection

States the MQSeries Workflow server a session must have been established with.

API interface declarations

Shows the required file declarations and calling sequences.

Parameters Describes each of the parameters together with an indication of whether the parameter is an input or output parameter.

Return type Describes the type of value returned by the call.

Return codes/ FmcException

Lists all possible return codes or exceptions which may issued or raised by this call.

Examples Points to an example of the call.

ActivityInstance actions

An ActivityInstance object represents an instance of an activity of a process instance. An activity instance is uniquely identified by its object identifier or by its fully qualified name within the process instance. The fully qualified name of an activity instance is a name in dot notation where the hierarchy of nested activities of type *Block* is presented from left to right, and their names are separated by a dot.

The following sections describe the actions which can be applied on an activity instance. See “ActivityInstance” on page 217 for a complete list of API calls.

ObtainProcessInstanceMonitor()

This API call retrieves the process instance monitor for the process instance the activity instance is part of from the MQSeries Workflow execution server (action call).

When the deep option is specified, all activity instances of type *Block* are resolved, that is, their block instance monitors are also fetched from the server.

Note: Deep is not yet supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes

- See “Action API calls” on page 121 for general information.

ActivityInstance

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process administrator
- Be the process creator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ActivityInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjActivityInstanceObtainProcessInstanceMonitor(  
    FmcjActivityInstanceHandle          hdlInstance,  
    bool                                deep,  
    FmcjProcessInstanceMonitorHandle *  monitor)
```

C++

```
APIRET ObtainProcessInstanceMonitor(  
    FmcjProcessInstanceMonitor &      monitor,  
    bool                                deep= false ) const
```

Java

```
public abstract  
ProcessInstanceMonitor obtainProcessInstanceMonitor( boolean deep )  
throws FmcException
```

COBOL

```
FmcjAINObtainProcInstMon.  
CALL "FmcjItemObtainProcessInstanceMonitor"  
USING  
    BY VALUE  
        hdlItem  
        deep  
    BY REFERENCE  
        monitor  
RETURNING  
    intReturnValue.
```

Parameters

| | |
|--------------------|--|
| <i>deep</i> | Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note: deep is not yet supported. |
| <i>hdlInstance</i> | Input. The activity instance whose process instance monitor is to be retrieved. |
| <i>monitor</i> | Input/Output. The address of the handle to the process instance monitor or the process instance monitor object to be set. |
| <i>returnCode</i> | Input/Output. The result of calling this method - see return codes below. |

Return type

APIRET The result of calling this method - see return codes below.

InstanceMonitor*/ProcessInstanceMonitor*/ ProcessInstanceMonitor

A pointer to the process instance monitor or the process instance monitor.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SubProcessInstance()

This API call retrieves the process instance which implements the activity instance from the MQSeries Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

In C++, when the process instance object to be initialized is not empty, then that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, no check is made whether the process instance handle already points to an object.

ActivityInstance

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ActivityInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjActivityInstanceSubProcessInstance(
    FmcjActivityInstanceHandle hdlInstance,
    FmcjProcessInstanceHandle * instance )
```

```
C++
APIRET SubProcessInstance( FmcjProcessInstance & instance ) const
```

```
Java
public abstract
ProcessInstance subProcessInstance() throws FmcException
```

```
COBOL
FmcjAISubProcInst.
CALL "FmcjActivityInstanceSubProcessInstance"
    USING
    BY VALUE
    hdlInstance
    BY REFERENCE
    instance
    RETURNING
    intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance object to be queried.
instance Input/Output. The subprocess instance object to be retrieved (initialized).
returnCode Input/Output. The result of calling this method - see return codes below.

Return type**APIRET**

The result of calling this method - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the subprocess instance or the subprocess instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ActivityInstanceNotification actions

An ActivityInstanceNotification object represents a notification on an activity instance assigned to a user.

Other items assigned to users are process instance notifications and work items. FmcJItem or Item represents the common properties of all items.

In C++, FmcjActivityInstanceNotification is thus a subclass of the FmcJItem class and inherits all properties and methods. In Java, ActivityInstanceNotification is thus a subclass of the Item class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcJItem. That is, common functions start with the prefix FmcJItem; they are also defined starting with the prefix FmcjActivityInstanceNotification.

An activity instance notification is uniquely identified by its object identifier.

ActivityInstanceNotification

The following sections describe the actions which can be applied on an activity instance notification. See “ActivityInstanceNotification” on page 221 for a complete list of API calls.

PersistentObject()

This API call retrieves the activity instance notification identified by the passed object identifier from the MQSeries Workflow execution server (action call).

The MQSeries Workflow execution server from which the activity instance notification is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information (primary and secondary) of the activity instance notification.

In C++, when the activity instance notification object to be initialized is not empty, that object is destructed before the new one is assigned. In C or COBOL, the application is completely responsible for the ownership of objects, that is, no check is made whether the activity instance notification handle already points to some object. In Java, an activity instance notification is newly created; the execution service acts as a factory.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Be the item owner
- Work item authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationPersistentObject(
    FmcjExecutionServiceHandle      service,
    char const *                      oid,
    FmcjActivityInstanceNotificationHandle * hdlItem )
```

```
C++
APIRET PersistentObject( FmcjExecutionService const & service,
    string const &          oid )
```

Java

```
public abstract
    ActivityInstanceNotification
    ExecutionService.persistentActivityInstanceNotification( String oid )
    throws FmcException
```

COBOL

```
FmcjAINPersistentObj.

    CALL      "FmcjActivityInstanceNotificationPersistentObject"
              USING
              BY VALUE
                serviceValue
                oid
              BY REFERENCE
                hdlItem
              RETURNING
                intReturnValue.
```

Parameters

hdlItem Input/Output. The address of the handle to the activity instance notification object to be set.

oid Input. The object identifier of the activity instance notification to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

ActivityInstanceNotification

The activity instance notification retrieved.

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance notification no longer exists.

FMC_ERROR_INVALID_OID(805)

The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

ActivityInstanceNotification

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

StartTool()

This API call starts the specified support tool (action call) for a user in a LAN environment.

The support tool must be one of the tools associated with the activity instance the notification has been created for. It is then started via the program execution agent associated with the logged-on user.

Note: A support tool can be started only via a program execution agent in the LAN environment; starting via a program execution server (in either environment) is currently not supported. Since there are only unattended processes under MQSeries Workflow for OS/390, it is not meaningful to start a support tool in this environment. The PES will simply ignore such an attempt.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the activity instance notification owner

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.ActivityInstanceNotification

COBOL fmcvars.cpy, fmcperf.cpy

```
C
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationStartTool(
    FmcjActivityInstanceNotificationHandle hdlItem,
    char const * toolName )
```

```
C++
APIRET StartTool( string const & toolName ) const
```

Java

```
public abstract
void startTool( String toolName ) throws FmcException
```

COBOL

```
FmcjAINstartTool.
      CALL      "FmcjActivityInstanceNotificationStartTool"
              USING
              BY VALUE
              hdItem
              toolName
              RETURNING
              intReturnValue.
```

Parameters

hdlItem Input. The handle of the activity instance notification to be dealt with.

toolName Input. The support tool to be started.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_INVALID_TOOL(129)

No tool name is provided or the specified tool is not defined for the activity instance notification.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

ActivityInstanceNotification

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

BlockInstanceMonitor actions

A BlockInstanceMonitor object represents a monitor for an activity instance of type *Block*.

Note: The ownership of a block instance monitor stays with the embracing process instance monitor. A block instance monitor is automatically deleted when the process instance monitor is deleted. After that action, using the block instance monitor handle or object is invalid.

An FmcjBlockInstanceMonitor or a BlockInstanceMonitor object represents the common aspects of monitors. In C++, FmcjBlockInstanceMonitor is thus the superclass of the FmcjProcessInstanceMonitor class and provides for all common properties and methods. In Java, BlockInstanceMonitor is thus a superclass of the ProcessInstanceMonitor class and provides for all common properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjBlockInstanceMonitor. That is, common functions start with the prefix FmcjBlockInstanceMonitor; they are also defined starting with the prefix FmcjProcessInstanceMonitor.

The following sections describe the actions which can be applied on a block instance monitor. See “BlockInstanceMonitor” on page 224 for a complete list of API calls.

ObtainBlockInstanceMonitor()

This API call retrieves the block instance monitor for the specified activity instance from the MQSeries Workflow execution server (action call). If the block instance monitor has already been retrieved, then that monitor is returned to the caller.

The specified activity instance must be of type *Block* and be part of this block instance monitor.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process administrator
- Be the process creator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx
Java com.ibm.workflow.api.BlockInstanceMonitor
COBOL fmcvars.cpy, fmcperf.cpy

C

```
FmcjBlockInstanceMonitorHandle
FMC_APIENTRY FmcjBlockInstanceMonitorObtainBlockInstanceMonitor(
    FmcjBlockInstanceMonitorHandle hdlMonitor,
    FmcjActivityInstanceHandle activity )
```

C++

```
FmcjBlockInstanceMonitor *
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity ) const

APIRET
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity,
                             FmcjBlockInstanceMonitor & monitor ) const
```

Java

```
public abstract
BlockInstanceMonitor obtainBlockInstanceMonitor(
    ActivityInstance activity ) throws FmcException
```

COBOL

```
FmcjBIMObtainBlockInstMon.

CALL
    "FmcjBlockInstanceMonitorObtainBlockInstanceMonitor"
    USING
    BY VALUE
        hdlMonitor
        activity
    RETURNING
        FmcjBIMHandleReturnValue.
```

Parameters

activity Input. The activity instance of type Block whose block instance monitor is to be retrieved.

hdlMonitor Input. The block instance monitor containing the activity instance of type Block.

monitor Input/Output. The block instance monitor retrieved.

Return type

APIRET The result returned by this API call - see return codes below.

FmcjBlockInstanceMonitor*/ Handle/ BlockInstanceMonitor
 The block instance monitor or a pointer or handle to the block instance monitor.

BlockInstanceMonitor

APIRET or the MQSeries Workflow **result object** can return the following codes or the following FmcExceptions can be thrown:

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The specified activity instance is not described by the block instance monitor.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_KIND(501)

The specified activity instance is not of type Block.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ObtainProcessInstanceMonitor()

This API call retrieves the process instance monitor for the specified activity instance from the MQSeries Workflow execution server (action call). If the process instance monitor has already been retrieved, then that monitor is returned to the caller.

The specified activity instance must be of type *Process* and be part of this block instance monitor.

When the *deep* option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also fetched from the server.

Note: Deep is not yet supported.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process administrator

- Be the process creator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.BlockInstanceMonitor |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
FmcjProcessInstanceMonitorHandle
FMC_APIENTRY FmcjBlockInstanceMonitorObtainProcessInstanceMonitor(
    FmcjBlockInstanceMonitorHandle hdlMonitor,
    FmcjActivityInstanceHandle activity,
    bool deep )
```

C++

```
FmcjProcessInstanceMonitor *
ObtainProcessInstanceMonitor(
    FmcjActivityInstance const & activity,
    bool deep= false ) const

APIRET ObtainProcessInstanceMonitor(
    FmcjActivityInstance const & activity,
    FmcjProcessInstanceMonitor & monitor,
    bool deep= false ) const
```

Java

```
public abstract
ProcessInstanceMonitor
obtainProcessBlockInstanceMonitor( ActivityInstance activity,
    boolean deep )
throws FmcException
```

BlockInstanceMonitor

COBOL

```
FmcjBIMObtainProcInstMon.  
  
    CALL  
        "FmcjBlockInstanceMonitorObtainProcessInstanceMonitor"  
        USING  
        BY VALUE  
        hdlMonitor  
        activity  
        deep  
        RETURNING  
        FmcjPIMHandleReturnValue.
```

Parameters

- activity* Input. The activity instance of type Process whose process instance monitor is to be retrieved.
- deep* Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is not yet supported.
- hdlMonitor* Input. The block instance monitor containing the activity instance of type Process.
- monitor* Output. The process instance monitor retrieved.

Return type

APIRET The result returned by this API call - see return codes below.

FmcjProcessInstanceMonitor*/ Handle/ ProcessInstanceMonitor

The process instance monitor or a pointer or handle to the process instance monitor.

APIRET or the MQSeries Workflow **result object** can return the following codes or the following FmcExceptions can be thrown:

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The specified activity instance is not described by the block instance monitor.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_KIND(501)

The specified activity instance is not of type Process.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Refresh()

This API call refreshes the block instance monitor from the MQSeries Workflow execution server (action call).

All information about the block instance monitor is retrieved.

When the deep option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also refreshed from the server.

Note: Deep is not yet supported.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process administrator
- Be the process creator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.BlockInstanceMonitor |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjBlockInstanceMonitorRefresh(
    FmcjBlockInstanceMonitorHandle hdlMonitor,
    bool deep )
```

C++

```
APIRET Refresh( bool deep= false )
```

BlockInstanceMonitor

Java

```
public abstract  
void refresh( boolean deep ) throws FmcException
```

COBOL

```
FmcjBIMRefresh.  
  
CALL "FmcjBlockInstanceMonitorRefresh"  
    USING  
    BY VALUE  
    hdlMonitor  
    deep  
    RETURNING  
    intReturnValue.
```

Parameters

deep Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is not yet supported.

hdlMonitor Input. The handle of the block instance monitor to be refreshed.

Return type

APIRET The result returned by this API call - see return codes below.

Return codes/ FmcException

| | |
|--------------------------------------|--|
| FMC_OK(0) | The API call completed successfully. |
| FMC_ERROR(1) | A parameter references an undefined location. For example, the address of a handle is 0. |
| FMC_ERROR_EMPTY(122) | The object has not yet been read from the database, that is, does not yet represent a persistent one. |
| FMC_ERROR_INVALID_HANDLE(130) | The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type. |
| FMC_ERROR_DOES_NOT_EXIST(118) | The process instance no longer exists. |
| FMC_ERROR_NOT_AUTHORIZED(119) | Not authorized to use the API call. |
| FMC_ERROR_NOT_LOGGED_ON(106) | Not logged on. |
| FMC_ERROR_COMMUNICATION(13) | The specified server cannot be reached; the server to which the connection should be established is not defined in your profile. |
| FMC_ERROR_INTERNAL(100) | An MQSeries Workflow internal error has occurred. Contact your IBM representative. |
| FMC_ERROR_MESSAGE_FORMAT(103) | An internal message format error. Contact your IBM representative. |
| FMC_ERROR_TIMEOUT(14) | Timeout has occurred. |

Container activity implementation calls

A Container object represents a data container of a process template, process instance, work item, or activity implementation . A container can be a read-only input container or a read/write input or output container.

The API calls defined for the container allow to access the values of data members of a basic type (container leaves), or to get a substructure of a container, a container element.

A Container object represents the common aspects of read-only or read/write containers. In C++, FmcjContainer is thus the superclass of the FmcjReadOnlyContainer and FmcjReadWriteContainer classes and provides for all common properties and methods. In Java, Container is thus a superclass of the ReadOnlyContainer and ReadWriteContainer classes and provides for all common properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjContainer. That is, common functions start with the prefix FmcjContainer; they are also defined starting with the prefixes FmcjReadOnlyContainer and FmcjReadWriteContainer.

The following sections describe the activity implementation functions which are used for communication between an activity implementation and a program execution server. See “Container” on page 225 for a complete list of API calls on containers.

InContainer()

This API call retrieves the input container from the CICS COMMAREA or IMS I/O Area (activity implementation call).

It can be used only from within an activity implementation.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer() or SetRemoteOutContainer().

Usage notes

- See “Activity implementation API calls” on page 121 for general information.

Authorization

Be an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|-------|---|
| C | fmcjcon.h or fmcjrun.h |
| C++ | fmcjpcon.hxx or fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionAgent |
| COBOL | fmcvars.cpy, fmcperfc.cpy (or fmcperfl.cpy) |

Container

C

```
APIRET FMC_APIENTRY FmcjContainerInContainer(  
    FmcjReadOnlyContainerHandle * input )
```

C++

```
static APIRET InContainer( FmcjReadOnlyContainer & input )
```

Java

```
public abstract  
    ReadOnlyContainer ExecutionAgent.inContainer()  
throws FmcException
```

COBOL

```
FmcjInCtnr.  
  
    CALL    "FmcjContainerInContainer"  
          USING  
          BY REFERENCE  
            inputValue  
          RETURNING  
            intReturnValue.
```

Parameters

input Input/Output. The address of the input container handle or the input container of the activity implementation to be set.

Return type

long/ APIRET

The return code from this API call - see return codes below.

ReadOnlyContainer

The input container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an input container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

Examples

- For a C example, see “Programming an activity implementation (C)” on page 559
- For a C++ example, see “Programming an activity implementation (C++)” on page 560
- For a COBOL example, see “Programming an activity implementation (COBOL)” on page 561

OutContainer()

This API call retrieves the output container from the CICS COMMAREA or IMS I/O Area (activity implementation call).

It can be used only from within an activity implementation.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer() or SetRemoteOutContainer().

Usage notes

- See “Activity implementation API calls” on page 121 for general information.

Authorization

Be an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|--------------|--|
| C | fmcjcon.h or fmcjcrun.h |
| C++ | fmcjpcon.hxx or fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionAgent |
| COBOL | fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy) |

```

C
APIRET FMC_APIENTRY FmcjContainerOutContainer(
    FmcjReadWriteContainerHandle * output )
    
```

```

C++
static APIRET OutContainer( FmcjReadWriteContainer & output )
    
```

Container

Java

```
public abstract  
    ReadWriteContainer ExecutionAgent.outContainer()  
throws FmcException
```

COBOL

```
FmcjCOutCtnr.  
  
    CALL    "FmcjContainerOutContainer"  
           USING  
           BY REFERENCE  
           outputValue  
           RETURNING  
           intReturnValue.
```

Parameters

output Input/Output. The address of the output container handle or the output container of the activity implementation to be set.

Return type

long/ APIRET The return code from this API call - see return codes below.

ReadWriteContainer

The output container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation, or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

Examples

- For a C example, see “Programming an activity implementation (C)” on page 559

- For a C++ example, see “Programming an activity implementation (C++)” on page 560
- For a COBOL example, see “Programming an activity implementation (COBOL)” on page 561

RemoteInContainer()

This API call retrieves the input container from the CICS COMMAREA or IMS I/O Area (activity implementation call).

It can be used only from within a program started by an activity implementation, if the COMMAREA or I/O Area was passed to the program.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer() or SetRemoteOutContainer().

Usage notes

- See “Activity implementation API calls” on page 121 for general information.

Authorization

Be a program started by an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|--------------|--|
| C | fmcjcon.h or fmcjcrun.h |
| C++ | fmcjpcon.hxx or fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionAgent |
| COBOL | fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy) |

```

C
APIRET FMC_APIENTRY FmcjContainerRemoteInContainer(
    char const *      programID,
    FmcjReadOnlyContainerHandle * input )
    
```

```

C++
static APIRET RemoteInContainer(
    string const &      programID,
    FmcjReadOnlyContainer & input )
    
```

```

Java
public abstract
    ReadOnlyContainer ExecutionAgent.remoteInContainer( String programID )
throws FmcException
    
```

```

COBOL

      FmcjCRemoteInCtrn.

          CALL      "FmcjContainerRemoteInContainer"
                USING
                BY REFERENCE
                programID
                inputValue
                RETURNING
                intReturnValue.
    
```

Parameters

input Input/Output. The address of the input container handle or the input container of the activity implementation to be set.

programID Input. The program identification by which the activity implementation is known to the program execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

ReadOnlyContainer

The input container of the activity implementation>.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)

The program identification is invalid.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an input container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation, or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

RemoteOutContainer()

This API call retrieves the output container from the CICS COMMAREA or IMS I/O Area (activity implementation call).

It can be used only from within a program started by an activity implementation if the COMMAREA or I/O Area was passed to the program.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer() or SetRemoteOutContainer().

Usage notes

- See “Activity implementation API calls” on page 121 for general information.

Authorization

Be a program started by an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|--------------|--|
| C | fmcjcon.h or fmcjrun.h |
| C++ | fmcjpcon.hxx or fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionAgent |
| COBOL | fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy) |

```

C
APIRET FMC_APIENTRY FmcjContainerRemoteOutContainer(
    char const *          programID,
    FmcjReadWriteContainerHandle * output )
    
```

```

C++
static APIRET RemoteOutContainer(
    string const &          programID,
    FmcjReadWriteContainer & output )
    
```

```

Java
public abstract
    ReadWriteContainer ExecutionAgent.remoteOutContainer( String programID )
    throws FmcException
    
```

COBOL

```
FmcjCRemoteOutCtnr.  
  
    CALL    "FmcjContainerRemoteOutContainer"  
           USING  
           BY REFERENCE  
           programID  
           outputValue  
           RETURNING  
           intReturnValue.
```

Parameters

output Input/Output. The address of the output container handle or the output container of the activity implementation to be set.

programID Input. The program identification by which the activity implementation is known to the program execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

ReadWriteContainer

The output container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)

The program identification is invalid.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation, or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

SetOutContainer()

This API call returns the output container to the MQSeries Workflow program execution server (activity implementation call).

It can be used from within an activity implementation as often as required. Note, however, that the output container is not returned to the MQSeries Workflow execution server until the activity implementation ends. It is kept transiently in the CICS COMMAREA or IMS I/O Area.

Note: The calls `InContainer()`, `OutContainer()`, `ServicePassthrough()`, and their "remote" counterparts will fail after this function is called, due to an altered COMMAREA or I/O Area.

Usage notes

- See "Activity implementation API calls" on page 121 for general information.

Authorization

Be an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|--------------|---|
| C | <code>fmcjcon.h</code> or <code>fmcjrun.h</code> |
| C++ | <code>fmcjpcon.hxx</code> or <code>fmcjprun.hxx</code> |
| Java | <code>com.ibm.workflow.api.ExecutionAgent</code> |
| COBOL | <code>fmcvars.cpy</code> , <code>fmcperf.cpy</code> (or <code>fmcperfl.cpy</code>) |

C

```
APIRET FMC_APIENTRY FmcjContainerSetOutContainer(
    FmcjReadWriteContainerHandle const output )
```

C++

```
static APIRET SetOutContainer( FmcjReadWriteContainer const & output )
```

Java

```
public abstract
void ExecutionAgent.setOutContainer( ReadWriteContainer output )
throws FmcException
```

COBOL

```
FmcjCSetOutCtnr.

CALL "FmcjContainerSetOutContainer"
    USING
    BY VALUE
    outputValue
    RETURNING
    intReturnValue.
```

Container

Parameters

output Input. The output container handle or the output container of the activity implementation to be passed.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation, or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

Examples

- For a C example, see "Programming an activity implementation (C)" on page 559
- For a C++ example, see "Programming an activity implementation (C++)" on page 560

SetRemoteOutContainer()

This API call returns the output container to the MQSeries Workflow program execution server (activity implementation call).

It can be used from within a program started by an activity implementation as often as required. Note, however, that the output container is not returned to the MQSeries Workflow execution server until the activity implementation ends. It is kept transiently in the CICS COMMAREA or IMS I/O Area.

Note: The calls `InContainer()`, `OutContainer()`, `ServicePassthrough()`, and their "remote" counterparts will fail after this function is called, due to an altered COMMAREA or I/O Area.

Usage notes

- See "Activity implementation API calls" on page 121 for general information.

Authorization

Be a program started by an activity implementation.

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|--------------|--|
| C | fmcjcon.h or fmcjrun.h |
| C++ | fmcjpcon.hxx or fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionAgent |
| COBOL | fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy) |

C

```
APIRET FMC_APIENTRY FmcjContainerSetRemoteOutContainer(
char const *          programID,
FmcjReadWriteContainerHandle const output )
```

C++

```
static APIRET SetRemoteOutContainer(
string const &          programID,
FmcjReadWriteContainer const & output )
```

Java

```
public abstract
void ExecutionAgent.setRemoteOutContainer( String          programID,
                                           ReadWriteContainer output )
throws FmcException
```

COBOL

```
FmcjCSetRemoteOutCtnr.
CALL "FmcjContainerSetRemoteOutContainer"
    USING
    BY REFERENCE
    programID
    BY VALUE
    outputValue
    RETURNING
    intReturnValue.
```

Parameters

| | |
|------------------|--|
| <i>output</i> | Input. The output container handle or the output container of the activity implementation to be passed. |
| <i>programID</i> | Input. The program identification by which the activity implementation is known to the program execution server. |

Return type

long/ APIRET The return code from this API call - see return codes below.

Container

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)

The program identification is invalid.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not issued from within an activity implementation, or the program execution server is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

ExecutionService actions

An ExecutionService object represents a session between a user and an MQSeries Workflow execution server so that Runtime services may be requested.

The execution service object essentially provides for the basic API calls to set up a communication path to the specified MQSeries Workflow execution server and to establish the user session (log on), and finish it (log off).

At FmcjExecutionService or ExecutionService construction or allocation time the name of the MQSeries Workflow system and system group where the execution server resides can be specified. Default values are taken from the current user's profile or from the configuration profile, in this sequence, when logging on. The configuration where to search for the profiles can also be specified.

When the session to an execution server has been established, you can query objects for which you are authorized; for example, you can query process templates, process instances, or work items. The attributes of the queried objects can then be read and further actions can be requested. For example, once a process template has been queried, creation of a process instance can be asked for.

When the execution service object is destructed or deallocated and still represents an active session, logoff is automatically called (provided that there is no other object referencing this session). It is, however, recommended that logon and logoff calls are paired before the execution service object is deallocated.

FmcjService or Service represents common properties of services.

In C++, `FmcjExecutionService` is thus a subclass of the `FmcjService` class and inherits all properties and methods. In the Java language, `ExecutionService` is thus a subclass of the `Service` class and inherits all properties and methods. Similarly, in C and COBOL, common implementations of functions are taken from `FmcjService`. That is, common functions start with the prefix `FmcjService`; they are also defined starting with the prefix `FmcjExecutionService`.

The following sections describe the actions which can be applied on an execution service. See “`ExecutionService`” on page 235 for a complete list of API calls.

CreateProcessInstanceList()

This API call creates a process instance list on the MQSeries Workflow execution server so that process instances can be grouped to one’s own taste or for a group of users (action call).

A process instance list is identified by:

- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the process instance list is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A process instance list groups a set of process instances which have the same characteristics. These characteristics are defined via search filters. The number of process instances in the list can be restricted via a threshold which specifies the maximum number of process instances to be returned to the client. That threshold is applied after the process instance list has been sorted according to sort criteria specified. Note that process instances are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a process instance list name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : .
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

A process instance list filter is specified as a character string containing a filter on process instances (refer to “How to read the syntax diagrams” on page xii).

Notes:

1. A *string* constant is to be enclosed in single quotes (`'`).

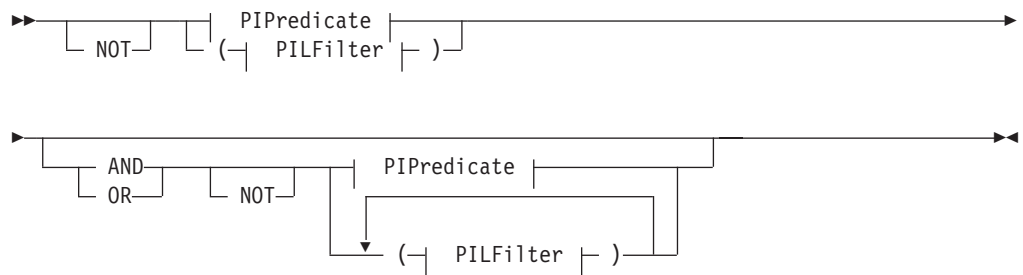
ExecutionService

A *pattern* is a string constant in which the asterisk and the question mark have special meanings.

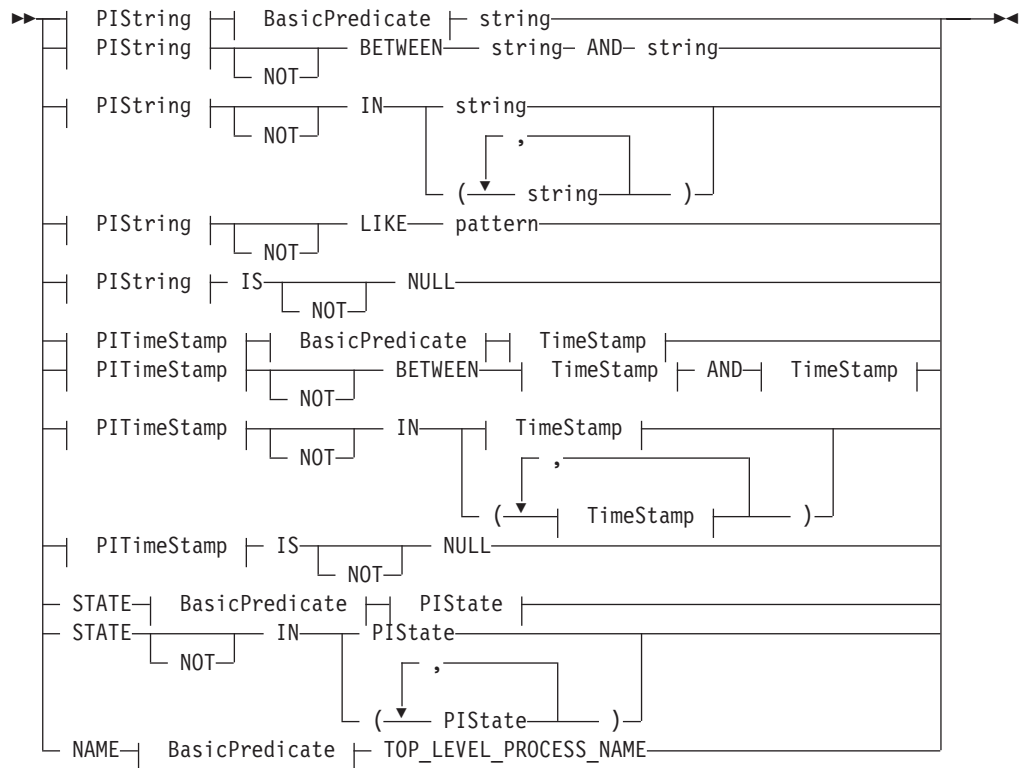
- The question mark (?) represents any single character.
- The asterisk (*) represents a string of zero or more characters.
- The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.

2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

PILFilter



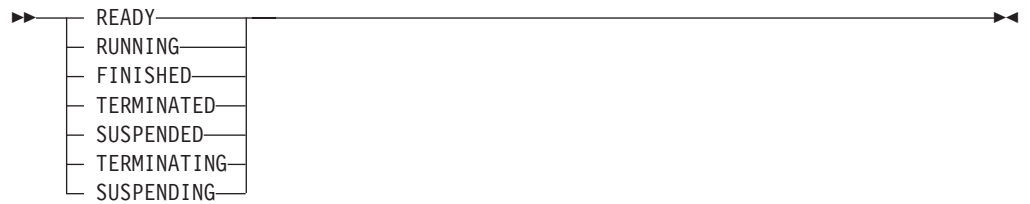
PIPredicate



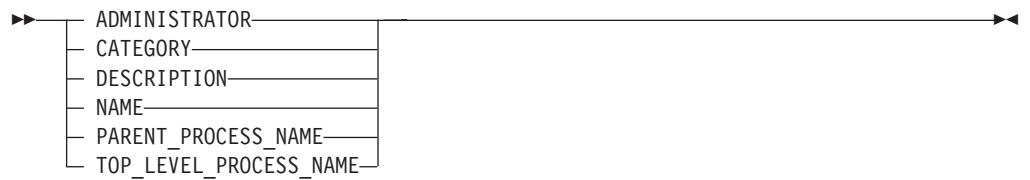
BasicPredicate



PIState



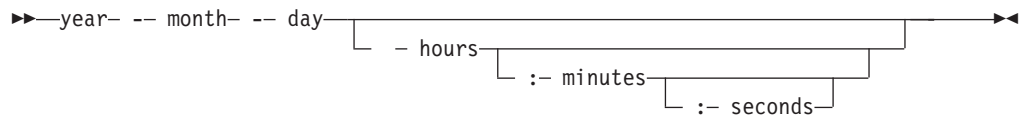
PIString



PITimeStamp



TimeStamp



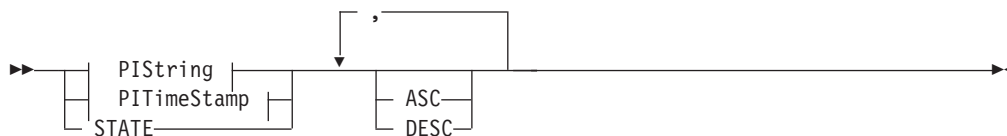
A process instance list sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PISState diagram.

PIOrderBy

ExecutionService



Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h
C++ fmcjprun.hxx
Java com.ibm.workflow.api.ExecutionService
COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceCreateProcessInstanceList(  
    FmcjExecutionServiceHandle        service,  
    char const *                        name,  
    enum FmcjPersistentListTypeOfList type,  
    char const *                        owner,  
    char const *                        description,  
    char const *                        filter,  
    char const *                        sortCriteria,  
    unsigned long *                     threshold,  
    FmcjProcessInstanceListHandle *    newList )
```

C++

```
APIRET CreateProcessInstanceList(  
    string const &                      name,  
    FmcjPersistentList::TypeOfList type,  
    string const *                      owner,  
    string const *                      description,  
    string const *                      filter,  
    string const *                      sortCriteria,  
    unsigned long const *                threshold,  
    FmcjProcessInstanceList &         newList ) const
```

Java

```

public abstract
ProcessInstanceList createProcessInstanceList(
    String          name,
    TypeOfList     type,
    String          owner,
    String          description,
    String          filter,
    String          sortCriteria,
    Integer         threshold ) throws FmcException

```

COBOL

```

FmcjESCreateProcInstList.
    CALL    "FmcjExecutionServiceCreateProcessInstanceList"
           USING
           BY VALUE
           serviceValue
           name
           typeValue
           ownerValue
           description
           filter
           sortCriteria
           threshold
           BY REFERENCE
           newList
           RETURNING
           intReturnValue.

```

Parameters

| | |
|---------------------|---|
| <i>description</i> | Input. A user-defined description of the process instance list. |
| <i>filter</i> | Input. The filter criteria which characterize the process instances to be contained in the process instance list. |
| <i>name</i> | Input. A user-defined name for the process instance list. |
| <i>newList</i> | Input/Output. The newly created process instance list. |
| <i>owner</i> | Input. The owner of the list when the type is private. Ignored for public lists. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the process instances in the process instance list. |
| <i>threshold</i> | Input. The threshold which defines the maximum number of process instances in the process instance list to be passed to the client. |
| <i>type</i> | Input. An indication whether a private or a public list is to be created. |

Return type

long/ APIRET The return code from this API call - see return codes below.

ProcessInstanceList

The newly created process instance list.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

ExecutionService

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_DESCRIPTION(810)

The specified description is invalid.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is invalid.

FMC_ERROR_INVALID_LIST_TYPE(813)

The specified list type is invalid.

FMC_ERROR_INVALID_NAME(134)

The specified process instance list name does not comply with the syntax rules.

FMC_ERROR_INVALID_USER(132)

The user ID specified for the owner of the list does not conform to the syntax rules.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are invalid.

FMC_ERROR_INVALID_THRESHOLD(807)

The specified threshold is invalid; exceeds the maximum possible value.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized.

FMC_ERROR_OWNER_NOT_FOUND(812)

The person to become the owner of the process instance list is not found.

FMC_ERROR_NOT_UNIQUE(121)

The name of the process instance list is not unique within the specified type.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Create a process instance list (C)” on page 512.
- For a C++ example, see “Create a process instance list (C++)” on page 513.
- For a Java example, see “Create a process instance list (Java)” on page 514.
- For a COBOL example, see “Create a process instance list (COBOL)” on page 517.

CreateProcessTemplateList()

This API call creates a process template list on the MQSeries Workflow execution server so that process templates can be grouped to one’s own taste or for a group of users (action call).

A process template list is identified by:

- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the process template list is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A process template list groups a set of process templates which have the same characteristics. These characteristics are defined via filters. The number of process templates in the list can be restricted via a threshold which specifies the maximum number of process templates to be returned to the client. That threshold is applied after the process template list has been sorted according to sort criteria specified. Process templates are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a process template list name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : .
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:

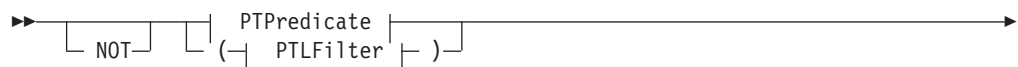
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

A process template list filter is specified as a character string containing a filter on process templates (refer to “How to read the syntax diagrams” on page xii).

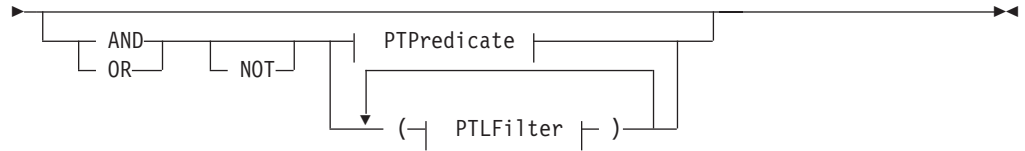
Notes:

1. A *string* constant is to be enclosed in single quotes (').
A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

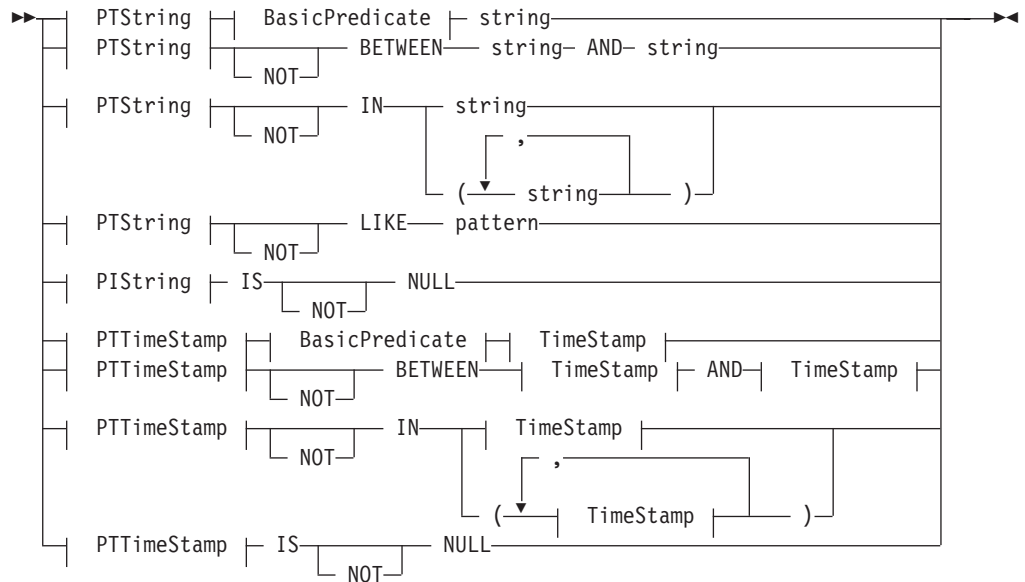
PTLFilter



ExecutionService



PTPredicate



BasicPredicate



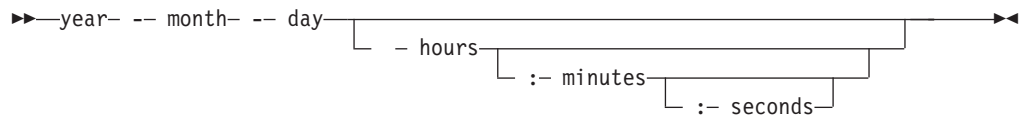
PTString



PTimeStamp



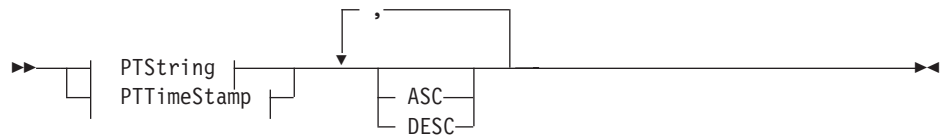
TimeStamp



A process template list sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTLOrderBy



Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

- C** fmcjcrun.h
- C++** fmcjprun.hxx
- Java** com.ibm.workflow.api.ExecutionService
- COBOL** fmcvars.cpy, fmcperf.cpy

```

C
APIRET FMC_APIENTRY FmcjExecutionServiceCreateProcessTemplateList(
    FmcjExecutionServiceHandle service,
    char const * name,
    enum FmcjPersistentListTypeOfList type,
    char const * owner,
    char const * description,
    char const * filter,
    char const * sortCriteria,
    unsigned long * threshold,
    FmcjProcessTemplateListHandle * newList )
    
```

ExecutionService

C++

```
APIRET CreateProcessTemplateList(  
    string const & name,  
    FmcjPersistentList::TypeOfList type,  
    string const * owner,  
    string const * description,  
    string const * filter,  
    string const * sortCriteria,  
    unsigned long const * threshold,  
    FmcjProcessTemplateList & newList ) const
```

Java

```
public abstract  
ProcessTemplateList createProcessTemplateList(  
    String name,  
    TypeOfList type,  
    String owner,  
    String description,  
    String filter,  
    String sortCriteria,  
    Integer threshold ) throws FmcException
```

COBOL

```
FmcjESCreateProcTemplList.  
  
    CALL "FmcjExecutionServiceCreateProcessTemplateList"  
        USING  
        BY VALUE  
            serviceValue  
            name  
            typeValue  
            ownerValue  
            description  
            filter  
            sortCriteria  
            threshold  
        BY REFERENCE  
            newList  
        RETURNING  
            intReturnValue.
```

Parameters

| | |
|---------------------|---|
| <i>description</i> | Input. A user-defined description of the process template list. |
| <i>filter</i> | Input. The filter criteria which characterize the process templates in the process template list. |
| <i>name</i> | Input. A user-defined name for the process template list. |
| <i>newList</i> | Input/Output. The newly created process template list. |
| <i>owner</i> | Input. The owner of the list when the type is private. Ignored for public lists. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the process templates in the process template list. |

threshold Input. The threshold which defines the maximum number of process templates in the process template list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code from this API call - see return codes below.

ProcessTemplateList

The newly created process template list.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_DESCRIPTION(810)

The specified description is invalid.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is invalid.

FMC_ERROR_INVALID_LIST_TYPE(813)

The specified list type is invalid.

FMC_ERROR_INVALID_NAME(134)

The specified process template list name does not comply with the syntax rules.

FMC_ERROR_INVALID_USER(132)

The user ID specified for the owner of the list does not conform to the syntax rules.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are invalid.

FMC_ERROR_INVALID_THRESHOLD(807)

The specified threshold is invalid; exceeds the maximum possible value.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized.

FMC_ERROR_OWNER_NOT_FOUND(812)

The person to become the owner of the process template list is not found.

FMC_ERROR_NOT_UNIQUE(121)

The name of the process template list is not unique within the specified type.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ExecutionService

Examples

- For a C example, see “Create a process instance list (C)” on page 512.
- For a C++ example, see “Create a process instance list (C++)” on page 513.
- For a Java example, see “Create a process instance list (Java)” on page 514.
- For a COBOL example, see “Create a process instance list (COBOL)” on page 517.

CreateWorklist()

This API call creates a worklist on the MQSeries Workflow execution server so that work items or notifications can be grouped to one’s own taste or for a group of users (action call).

A worklist is identified by:

- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the worklist is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A worklist groups a set of work items or notifications which have the same characteristics. These characteristics are defined via filters. The number of items in the worklist can be restricted via a threshold which specifies the maximum number of items to be returned to the client. That threshold is applied after the worklist has been sorted according to sort criteria specified. Items are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a worklist name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : .
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

A worklist filter is specified as a character string containing a filter on the items in the worklist (refer to “How to read the syntax diagrams” on page xii).

Notes:

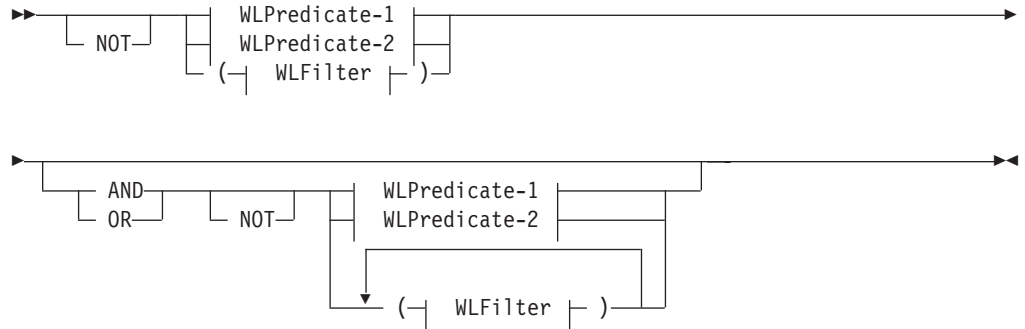
1. A *string* constant is to be enclosed in single quotes (').

A *pattern* is a string constant in which the asterisk and the question mark have special meanings.

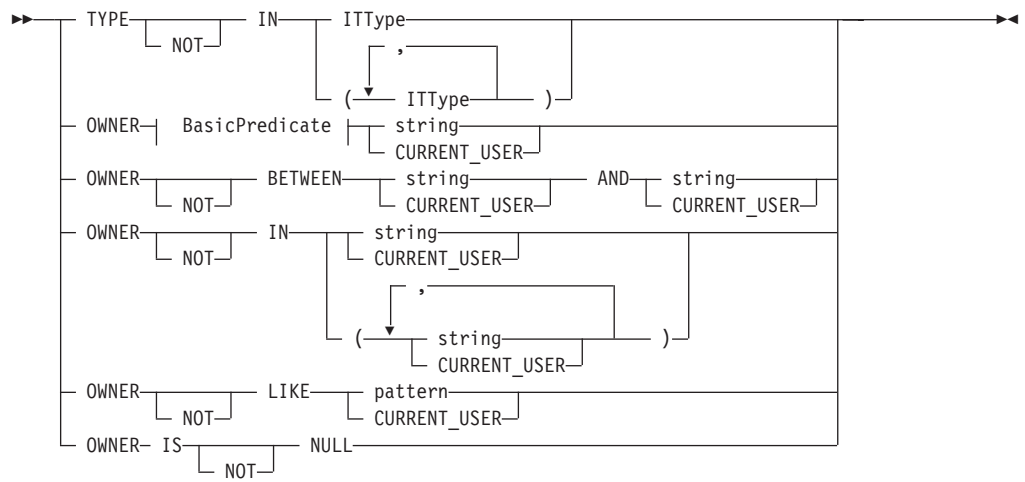
- The question mark (?) represents any single character.
- The asterisk (*) represents a string of zero or more characters.

- The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

WLFilter

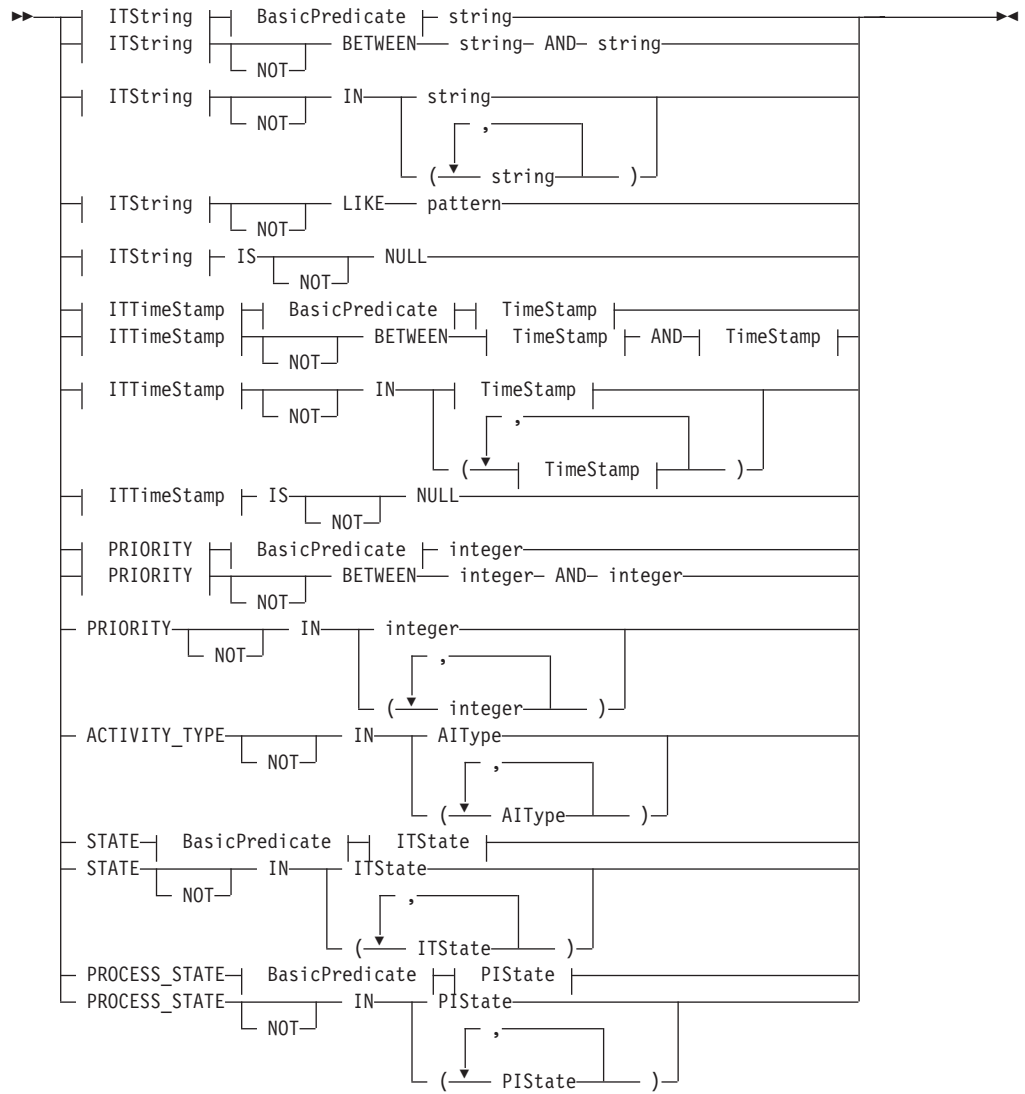


WLPredicate-1

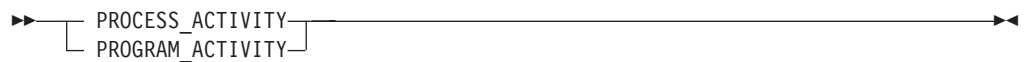


WLPredicate-2

ExecutionService



AIType



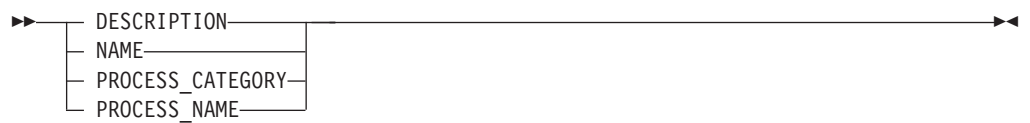
BasicPredicate



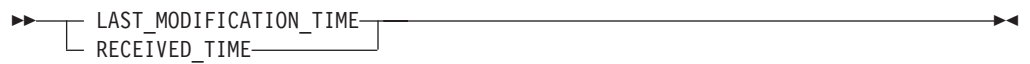
ITState



ITString



ITTimeStamp



ITType

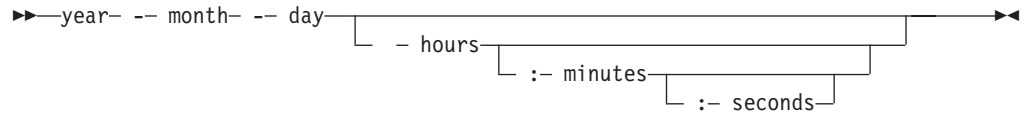


PIState



TimeStamp

ExecutionService



A worklist sort criterion is specified as a character string.

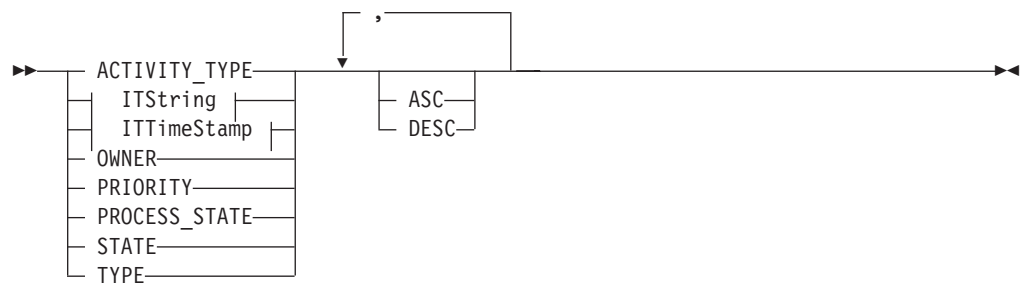
Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

Item types are sorted according to the sequence shown in the ITType diagram.

States are sorted according to the sequence shown in the ITState or the PISState diagram.

WLOrderBy



Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```

APIRET FMC_APIENTRY FmcjExecutionServiceCreateWorklist(
    FmcjExecutionServiceHandle    service,
    char const *                   name,
    enum FmcjPersistentListTypeOfList type,
    char const *                   owner,
    char const *                   description,
    char const *                   filter,
    char const *                   sortCriteria,
    unsigned long *                threshold,
    FmcjWorklistHandle *          newList )

```

C++

```

APIRET CreateWorklist(
    string const &                name,
    FmcjPersistentList::TypeOfList type,
    string const *                owner,
    string const *                description,
    string const *                filter,
    string const *                sortCriteria,
    unsigned long const *        threshold,
    FmcjWorklist &               newList ) const

```

Java

```

public abstract
WorkList createWorkList(
    String                name,
    TypeOfList           type,
    String                owner,
    String                description,
    String                filter,
    String                sortCriteria,
    Integer               threshold ) throws FmcException

```

COBOL

```

FmcjESCreateWorklist.

    CALL    "FmcjExecutionServiceCreateWorklist"
           USING
           BY VALUE
           serviceValue
           name
           typeValue
           ownerValue
           description
           filter
           sortCriteria
           threshold
           BY REFERENCE
           newList
           RETURNING
           intReturnValue.

```

ExecutionService

Parameters

| | |
|---------------------|---|
| <i>description</i> | Input. A user-defined description of the worklist. |
| <i>filter</i> | Input. The filter criteria which characterize the items in the worklist. |
| <i>name</i> | Input. A user-defined name for the worklist. |
| <i>newList</i> | Input/Output. The newly created worklist. |
| <i>owner</i> | Input. The owner of the list when the type is private. Ignored for public lists. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the items in the worklist. |
| <i>threshold</i> | Input. The threshold which defines the maximum number of items in the worklist. |
| <i>type</i> | Input. An indication whether a private or a public list is to be created. |

Return type

| | |
|---------------------|--|
| long/ APIRET | The return code from this API call - see return codes below. |
| WorkList | The newly created worklist. |

Return codes/ FmcException

| | |
|---|--|
| FMC_OK(0) | The API call completed successfully. |
| FMC_ERROR(1) | A parameter references an undefined location. For example, the address of a handle is 0. |
| FMC_ERROR_INVALID_HANDLE(130) | The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type. |
| FMC_ERROR_INVALID_DESCRIPTION(810) | The specified description is invalid. |
| FMC_ERROR_INVALID_FILTER(125) | The specified filter is invalid. |
| FMC_ERROR_INVALID_LIST_TYPE(813) | The specified list type is invalid. |
| FMC_ERROR_INVALID_NAME(134) | The specified worklist name does not comply with the syntax rules. |
| FMC_ERROR_INVALID_USER(132) | The user ID specified for the owner of the list does not conform to the syntax rules. |
| FMC_ERROR_INVALID_SORT(808) | The specified sort criteria are invalid. |
| FMC_ERROR_INVALID_THRESHOLD(807) | The specified threshold is invalid; exceeds the maximum possible value. |
| FMC_ERROR_NOT_LOGGED_ON(106) | Not logged on. |
| FMC_ERROR_NOT_AUTHORIZED(119) | Not authorized. |
| FMC_ERROR_OWNER_NOT_FOUND(812) | The person to become the owner of the worklist is not found. |
| FMC_ERROR_NOT_UNIQUE(121) | The name of the worklist is not unique within the specified type. |
| FMC_ERROR_COMMUNICATION(13) | The specified server cannot be reached; the server to which the connection should be established is not defined in your profile. |

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Create a process instance list (C)” on page 512.
- For a C++ example, see “Create a process instance list (C++)” on page 513.
- For a Java example, see “Create a process instance list (Java)” on page 514.
- For a COBOL example, see “Create a process instance list (COBOL)” on page 517.

Logoff()

This API call allows the application to finish the specified user session with an MQSeries Workflow execution server (action call).

When logoff has been successfully executed, no further client/server calls are accepted using this execution service object.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjExecutionServiceLogoff(
    FmcjExecutionServiceHandle service )
    
```

```

C++
APIRET Logoff()
    
```

ExecutionService

Java

```
public abstract  
void logoff() throws FmcException
```

COBOL

```
FmcjESLogoff.  
  
CALL "FmcjExecutionServiceLogoff"  
      USING  
      BY VALUE  
      serviceValue  
      RETURNING  
      intReturnValue.
```

Parameters

service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET

The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

For examples see "Chapter 6. Examples" on page 511.

Logon()

This API call allows an application to establish a user session with an MQSeries Workflow execution server (action call).

A successful Logon() is the prerequisite for using all other action and program execution management API calls of the MQSeries Workflow API.

The user ID to log on with must be a registered MQSeries Workflow user.

When the execution server supports *unified logon*, an empty password and user ID can be provided. The user ID to log on with is then retrieved from the operating system, that is, logon must have been performed at the client. The client is trusted and the execution server performs no password checking.

After a successful logon, the execution service object represents that single user session. A further request to log on with a different user ID will be rejected. You can, however, establish as many sessions as needed, even for the same user, using different execution service objects, one for each session.

At logon time, you can specify your mode of operation. When you are operating in the *present* session mode, the execution server can assume that you are able to react to requests from activity implementations which might ask, for example, for container data. Thus, activity instances that are started automatically may be scheduled on your behalf - provided that you also started a program execution agent.⁵

Furthermore, the *present* mode indicates to MQSeries Workflow that the session can handle unsolicited messages pushed by the execution server - see "The push data access model" on page 16 for additional prerequisites.

There can only be a single present session for one user. The *present here* option can be used, to force that other present session logoff and to newly establish a present session here.

When you are operating in a *default* session mode, the execution server does not assume that you are able to react. Activity instances are not automatically started on your behalf and messages are not pushed to you. There can be multiple sessions for one user with the *default* session mode.

The following enumeration types can be used to specify the session mode:

| | |
|-------------|---|
| C | FmcjServiceSessionMode |
| C++ | FmcjService::SessionMode |
| Java | com.ibm.workflow.api.ServicePackage.SessionMode |

The enumeration constants can take the following values; it is strongly advised to use the symbolic names instead of the associated integer values.

| | |
|----------------|--|
| Default | Indicates that you want to operate in a default, nonpresent, session mode. |
| C | Fmc_SM_Default |
| C++ | FmcjService::Default or FmcjExecutionService::Default |
| Java | SessionMode.DEFAULT |
| COBOL | Fmc-SM-Default |
| Present | Indicates that you want to operate in a present session mode. |

5. This is not an option under OS/390.

ExecutionService

| | | |
|--------------------|--------------|--|
| | C | Fmc_SM_Present |
| | C++ | FmcjService::Present or FmcjExecutionService::Present |
| | Java | SessionMode.PRESENT |
| | COBOL | Fmc-SM-Present |
| PresentHere | | Indicates that you want to operate in a present session mode. If a session with the present session mode already exists, then it should be logged off. |
| | C | Fmc_SM_PresentHere |
| | C++ | FmcjService::PresentHere or FmcjExecutionService::PresentHere |
| | Java | SessionMode.PRESENT_HERE |
| | COBOL | Fmc-SM-PresentHere |

At logon time, you can also specify whether you are back in case you are set to be absent. When you are not absent you participate in work assignment; otherwise no work items are assigned to you.

The following enumeration types can be used to deal with your absence:

| | |
|-------------|--|
| C | FmcjServiceAbsenceIndicator |
| C++ | FmcjService::AbsenceIndicator |
| Java | com.ibm.workflow.api.ServicePackage.AbsenceIndicator |

The enumeration constants can take the following values; it is strongly advised to use the symbolic names instead of the associated integer values.

| | | |
|---------------|--------------|---|
| NotSet | | Indicates that no value is specified. This means that the definition in your person record applies. Your absence is reset or not according to the definition found there. |
| | C | Fmc_SA_NotSet |
| | C++ | FmcjService::NotSet or FmcjExecutionService::NotSet |
| | Java | AbsenceIndicator.NOT_SET |
| | COBOL | Fmc-SA-NotSet |
| Reset | | Indicates that your absence setting is to be reset; you are back. |
| | C | Fmc_SA_Reset |
| | C++ | FmcjService::Reset or FmcjExecutionService::Reset |
| | Java | AbsenceIndicator.RESET |
| | COBOL | Fmc-SA-Reset |
| Leave | | Indicates that your absence setting should stay as is; you are either absent or not. |
| | C | Fmc_SA_Leave |
| | C++ | FmcjService::Leave or FmcjExecutionService::Leave |
| | Java | AbsenceIndicator.LEAVE |
| | COBOL | Fmc-SA-Leave |

For Java programs, logon2() allows for the specification of the session mode and absence setting.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be a registered MQSeries Workflow user

Required connection

None

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogon (
    FmcjExecutionServiceHandle service,
    char const * userID,
    char const * password,
    enum FmcjServiceSessionMode sessionMode,
    enum FmcjServiceAbsenceIndicator absenceIndicator )
```

C++

```
APIRET Logon( string const & userID, string const & password )

APIRET Logon(
    string const & userID,
    string const & password,
    SessionMode sessionMode = Present,
    AbsenceIndicator absenceIndicator = NotSet )
```

Java

```
public abstract
void logon ( String userID, String password )

public abstract
void logon2( String userID,
             String password,
             SessionMode sessionMode,
             AbsenceIndicator absenceIndicator ) throws FmcException
```

COBOL

```

FmcjESLogon.

CALL    "FmcjExecutionServiceLogon"
        USING
        BY VALUE
        serviceValue
        userID
        passwordValue
        sessionMode
        absenceIndicator
        RETURNING
        intReturnValue.
    
```

Parameters

absenceIndicator

Input. An indicator to state how to handle any absence set.

password

Input. The password of the user. Can be empty for unified logon.

service

Input. A handle to the service object representing the session to be established with the execution server.

sessionMode

Input. The mode of the session to be established.

userID

Input. The user ID of the user on whose behalf a logon is to be made. Can be empty for unified logon.

Return type

long/ APIRET

The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_ALREADY_LOGGED_ON(11)

The user is already logged on with present mode or the execution service object already represents a different user session.

FMC_ERROR_BACK_LEVEL_VERSION(504)

The version of the client is out-of-date, that is, not supported by this server.

FMC_ERROR_INVALID_ABSENCE_SPEC(905)

An unknown absence setting has been specified.

FMC_ERROR_INVALID_SESSION_MODE(901)

An unknown session mode has been specified.

FMC_ERROR_NEWER_VERSION(505)

The version of the client is newer than the server version, that is, not supported.

FMC_ERROR_PASSWORD(12)

Incorrect password.

FMC_ERROR_PROFILE(124)

Required user or workstation profile entries cannot be found.

- FMC_ERROR_USERID_UNKNOWN(10)**
No user ID registered with MQSeries Workflow has been provided.
- FMC_ERROR_COMMUNICATION(13)**
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
- FMC_ERROR_INTERNAL(100)**
An MQSeries Workflow internal error has occurred. Contact your IBM representative.
- FMC_ERROR_MESSAGE_FORMAT(103)**
An internal message format error. Contact your IBM representative.
- FMC_ERROR_TIMEOUT(14)**
Timeout has occurred.

Examples

For examples see “Chapter 6. Examples” on page 511.

Passthrough()

This API call can be used by an activity implementation to establish a user session with an MQSeries Workflow execution server from within this program (activity implementation).

When successfully executed, a session is set up to the same execution server from which the work item implemented by this program was started; the user on whose behalf the session is set up is the same one on whose behalf the work item was started.

Note: This call will fail after the COMMAREA or IMS I/O Area has been changed with SetOutContainer() or SetRemoteOutContainer().

Usage notes

- See “Activity implementation API calls” on page 121 for general information.

Authorization

Activity implementation started by MQSeries Workflow

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

- C** fmcjcrun.h
- C++** fmcjprun.hxx
- Java** com.ibm.workflow.api.ExecutionService
- COBOL** fmcvars.cpy, fmcperf.cpy

```

C
APIRET FMC_APIENTRY FmcjExecutionServicePassthrough(
    FmcjExecutionServiceHandle service )
    
```

ExecutionService

C++

```
APIRET Passthrough()
```

Java

```
public abstract  
void passthrough() throws FmcException
```

COBOL

```
FmcjESPassthrough.  
  
CALL "FmcjExecutionServicePassthrough"  
      USING  
      BY VALUE  
      serviceValue  
      RETURNING  
      intReturnValue.
```

Parameters

service Input. A handle to the service object which is to represent the session to be established with the execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_PROGRAM_EXECUTION(126)

Passthrough was not called from within an activity implementation, or the program execution server is not active.

FMC_ERROR_TOOL_FUNCTION(128)

Passthrough cannot be called from a program started by the program execution server.

FMC_ERROR_USERID_UNKNOWN(10)

The user who started the work item no longer exists.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Programming an activity implementation (C)” on page 559.
- For a C++ example, see “Programming an activity implementation (C++)” on page 560.
- For a COBOL example, see “Programming an activity implementation (COBOL)” on page 561.

QueryActivityInstanceNotifications()

This API call retrieves the activity instance notifications the user has access to from the MQSeries Workflow execution server (action call).

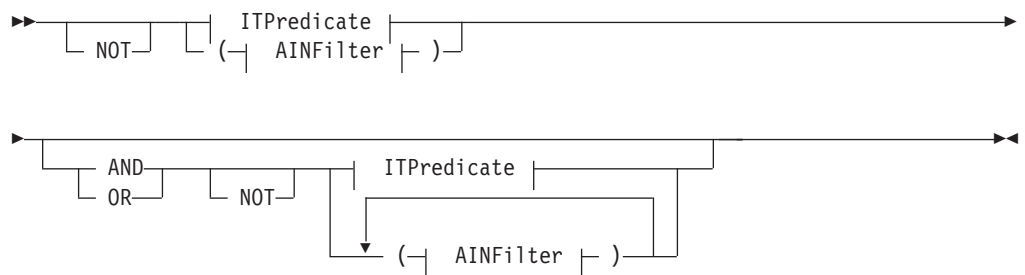
In C, C++, and COBOL, any activity instance notifications retrieved are appended to the supplied vector. If you want to read the current activity instance notifications only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

The activity instance notifications to be retrieved can be characterized by a filter. An activity instance notification filter is specified as a character string:

Notes:

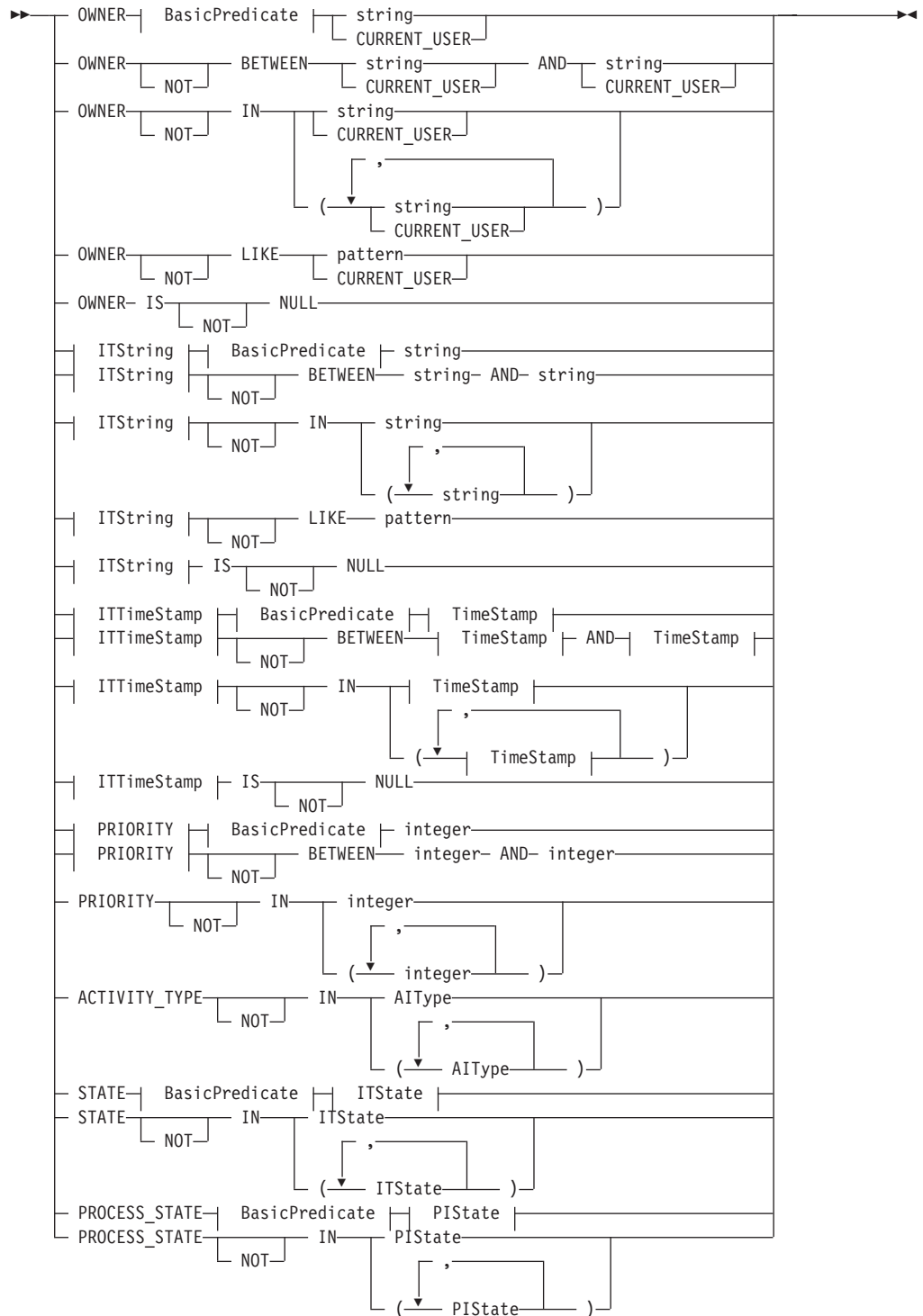
1. A *string* constant is to be enclosed in single quotes (').
A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

AINFilter



ITPredicate

ExecutionService



AType



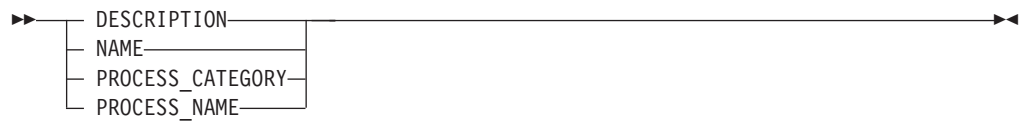
BasicPredicate



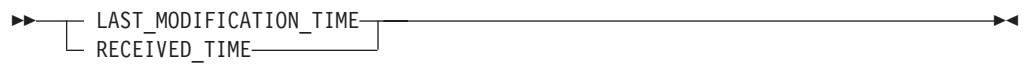
ITState



ITString



ITTimeStamp

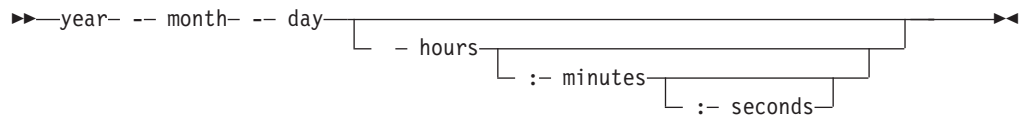


PIState



ExecutionService

TimeStamp



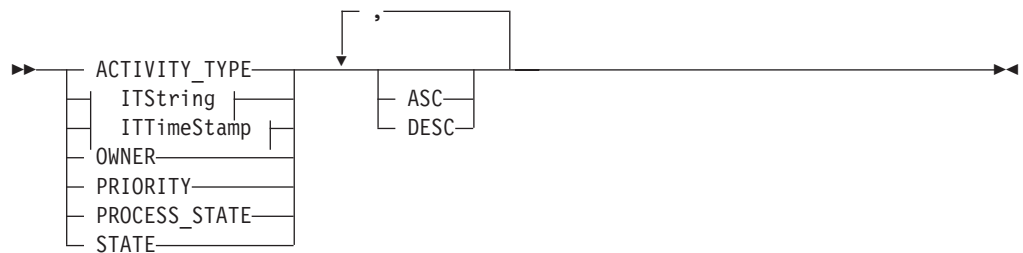
Activity instance notifications can be sorted. An activity instance notification sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState or the PISState diagram.

AINOrderBy



The number of activity instance notifications to be retrieved can be restricted via a threshold which specifies the maximum number of activity instance notifications to be returned to the client. That threshold is applied after the activity instance notifications have been sorted according to the sort criteria specified. Note that the activity instance notifications are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each activity instance notification is:

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- State

- SupportTools

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjExecutionServiceQueryActivityInstanceNotifications(
    FmcjExecutionServiceHandle          service,
    char const *                         filter,
    char const *                         sortCriteria,
    unsigned long const *                threshold,
    FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++

```
APIRET QueryActivityInstanceNotifications(
    string const *                       filter,
    string const *                       sortCriteria,
    unsigned long const *                threshold,
    vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications(
    String                               filter,
    String                               sortCriteria,
    Integer                               threshold )
throws FmcException
```

COBOL

```

FmcjESQueryActInstNotifs.

    CALL
        "FmcjExecutionServiceQueryActivityInstanceNotifications"
        USING
            BY VALUE
                serviceValue
                filter
                sortCriteria
                threshold
            BY REFERENCE
                notifications
        RETURNING
            intReturnValue.
    
```

Parameters

| | |
|----------------------|--|
| <i>filter</i> | Input. The filter criteria which characterize the activity instance notifications to be retrieved. |
| <i>notifications</i> | Input/Output. The qualifying vector of activity instance notifications. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the activity instance notifications found. |
| <i>threshold</i> | Input. The threshold which defines the maximum number of activity instance notifications to be returned to the client. |

Return type

APIRET The return code from this API call - see return codes below.

ActivityInstanceNotification[]

The qualifying activity instance notifications.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is invalid.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are invalid.

FMC_ERROR_INVALID_THRESHOLD(807)

The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of activity instance notifications to be returned exceeds the maximum size allowed for query results - see the **MAXIMUM_QUERY_MESSAGE_SIZE** definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query process instances (C)” on page 536.
- For a C++ example, see “Query process instances (C++)” on page 538.
- For a Java example, see “Query process instances (Java)” on page 539.
- For a COBOL example, see “Query process instances (COBOL)” on page 542.

QueryItems()

This API call retrieves the work items or notifications the user has access to from the MQSeries Workflow execution server (action call).

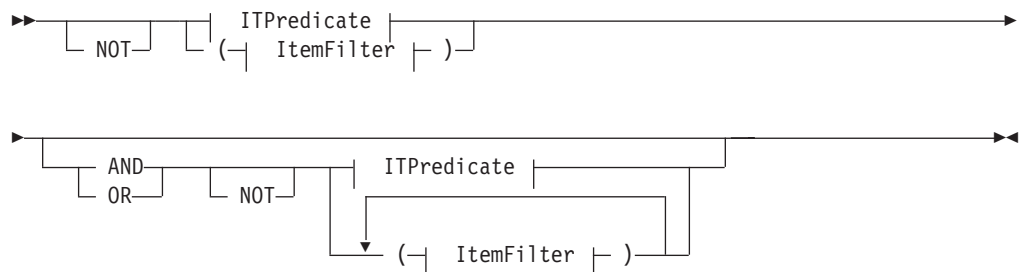
In C, C++, and COBOL, any items retrieved are appended to the supplied vector. If you want to read the current items only, you have to clear the vector before you issue this API call. This means that you should set the handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

The items to be retrieved can be characterized by a filter. An item filter is specified as a character string.

Notes:

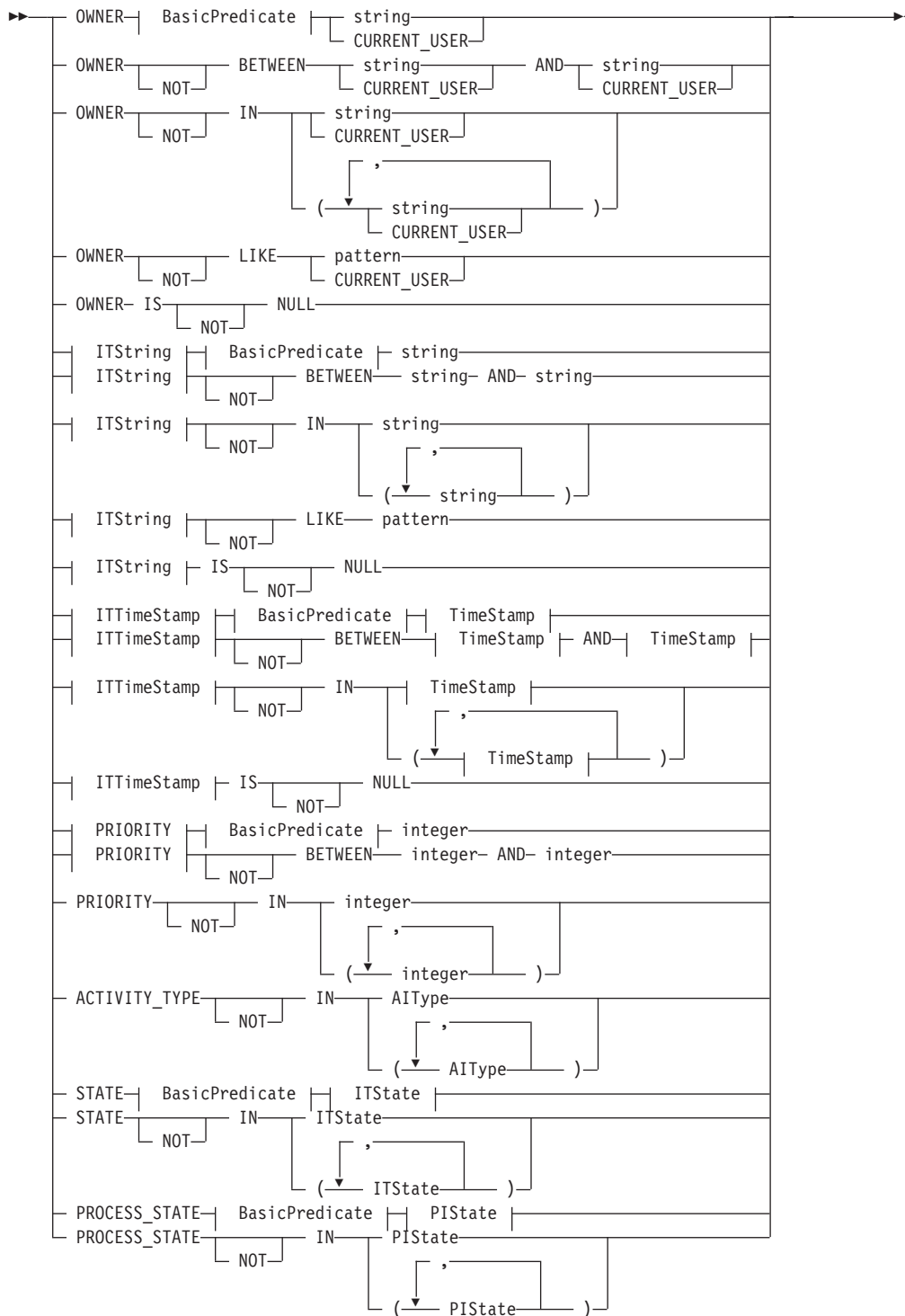
1. A *string* constant is to be enclosed in single quotes (').
 A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

ItemFilter

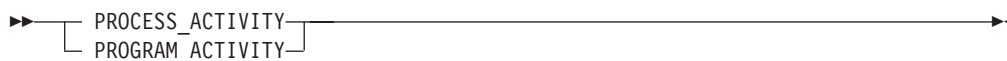


ITPredicate

ExecutionService



AType



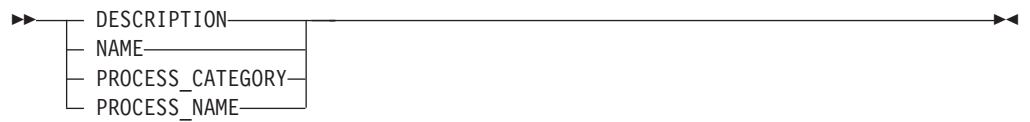
BasicPredicate



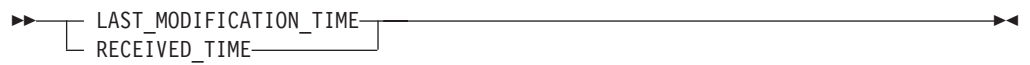
ITState



ITString



ITTimeStamp

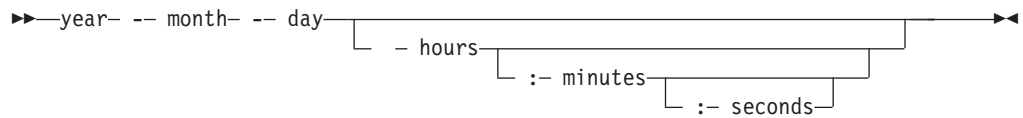


PIState



ExecutionService

TimeStamp



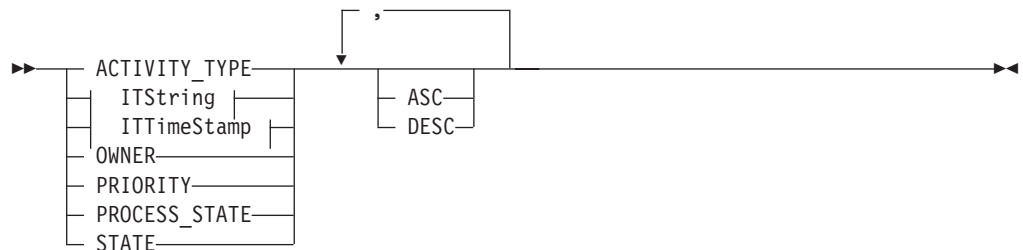
Items can be sorted. An item sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState or the PISState diagram.

ItemOrderBy



The number of items to be retrieved can be restricted via a threshold which specifies the maximum number of items to be returned to the client. That threshold is applied after the items have been sorted according to the sort criteria specified. Note that the items are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each item is:

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryItems(
    FmcjExecutionServiceHandle service,
    char const * filter,
    char const * sortCriteria,
    unsigned long const * threshold,
    FmcjItemHandle * items )
```

C++

```
APIRET QueryItems(
    string const * filter,
    string const * sortCriteria,
    unsigned long const * threshold,
    vector<FmcjItem> & items ) const
```

Java

```
public abstract
Item[] queryItems(
    String filter,
    String sortCriteria,
    Integer threshold ) throws FmcException
```

COBOL

```

FmcjESQueryItems.

    CALL    "FmcjExecutionServiceQueryItems"
           USING
           BY VALUE
           serviceValue
           filter
           sortCriteria
           threshold
           BY REFERENCE
           items
           RETURNING
           intReturnValue.
    
```

Parameters

- filter* Input. The filter criteria which characterize the items to be retrieved.
- items* Input/Output. The qualifying vector of items.
- service* Input. A handle to the service object representing the session with the execution server.
- sortCriteria* Input. The sort criteria to be applied to the items found.
- threshold* Input. The threshold which defines the maximum number of items to be returned to the client.

Return type

- APIRET** The return code from this API call - see return codes below.
- Item[]** The qualifying items.

Return codes/ FmcException

- FMC_OK(0)** The API call completed successfully.
- FMC_ERROR(1)** A parameter references an undefined location. For example, the address of a handle is 0.
- FMC_ERROR_INVALID_HANDLE(130)** The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
- FMC_ERROR_INVALID_FILTER(125)** The specified filter is invalid.
- FMC_ERROR_INVALID_SORT(808)** The specified sort criteria are invalid.
- FMC_ERROR_INVALID_THRESHOLD(807)** The specified threshold is invalid.
- FMC_ERROR_NOT_LOGGED_ON(106)** Not logged on.
- FMC_ERROR_QRY_RESULT_TOO_LARGE(817)** The number of items to be returned exceeds the maximum size allowed for query results - see the **MAXIMUM_QUERY_MESSAGE_SIZE** definition in your system, system group, or domain.
- FMC_ERROR_COMMUNICATION(13)** The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query process instances (C)” on page 536.
- For a C++ example, see “Query process instances (C++)” on page 538.
- For a Java example, see “Query process instances (Java)” on page 539.
- For a COBOL example, see “Query process instances (COBOL)” on page 542.

QueryProcessInstanceLists()

This API call retrieves the process instance lists the user has access to from the MQSeries Workflow execution server (action call).

In C, C++, and COBOL, any process instance lists retrieved are appended to the supplied vector. If you want to read the current process instance lists only, you have to clear the vector before you make this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceLists(
    FmcjExecutionServiceHandle      service,
    FmcjProcessInstanceListVectorHandle * lists )

```

```

C++
APIRET QueryProcessInstanceLists(
    vector<FmcjProcessInstanceList> & lists ) const

```

ExecutionService

Java

```
public abstract  
ProcessInstanceList[] queryProcessInstanceLists() throws FmcException
```

COBOL

```
FmcjESQueryProcInstLists.  
  
CALL      "FmcjExecutionServiceQueryProcessInstanceLists"  
          USING  
          BY VALUE  
          serviceValue  
          BY REFERENCE  
          lists  
          RETURNING  
          intReturnValue.
```

Parameters

lists Input/Output. The vector of process instance lists.
service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

ProcessInstanceList[]

The qualifying process instance lists.

Return codes/ **FmcException**

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of process instance lists to be returned exceeds the maximum size allowed for query results - see the **MAXIMUM_QUERY_MESSAGE_SIZE** definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query worklists (C)” on page 522.
- For a C++ example, see “Query worklists (C++)” on page 524.
- For a Java example, see “Query worklists (Java)” on page 525.
- For a COBOL example, see “Query worklists (COBOL)” on page 528.

QueryProcessInstanceNotifications()

This API call retrieves the process instance notifications the user has access to from the MQSeries Workflow execution server (action call).

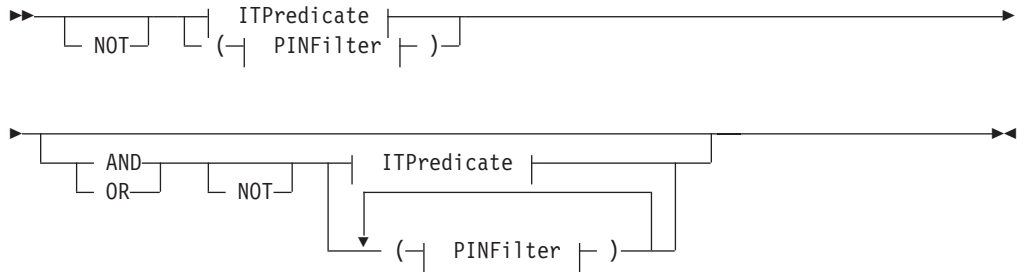
In C, C++, and COBOL, any process instance notifications retrieved are appended to the supplied vector. If you want to read the current process instance notifications only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

The process instance notifications to be retrieved can be characterized by a filter. A process instance notification filter is specified as a character string.

Notes:

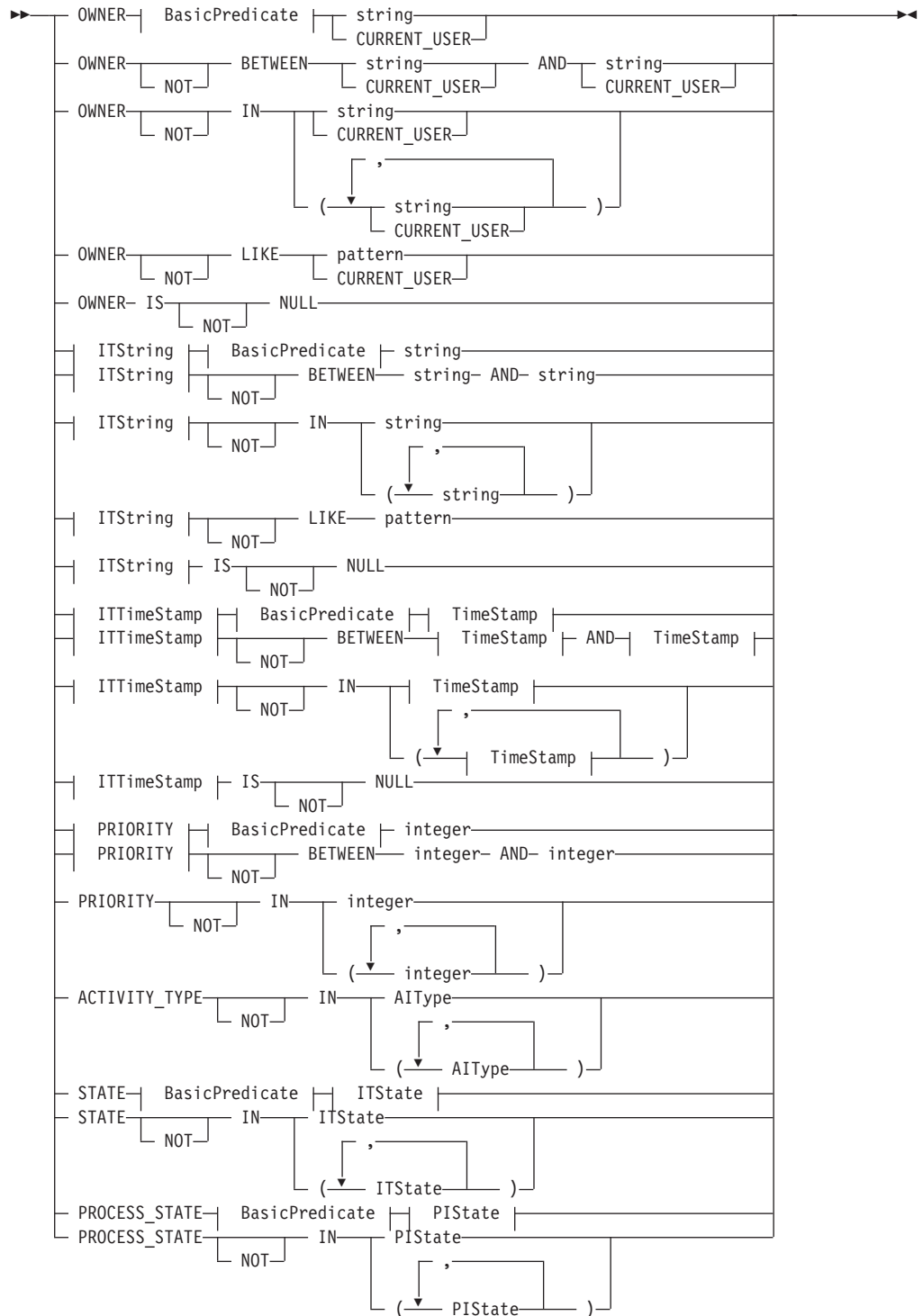
1. A *string* constant is to be enclosed in single quotes (').
 A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

PINFilter

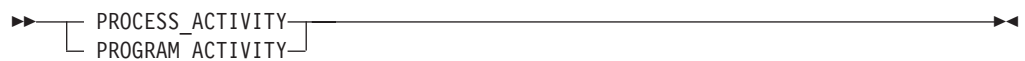


ITPredicate

ExecutionService



AType



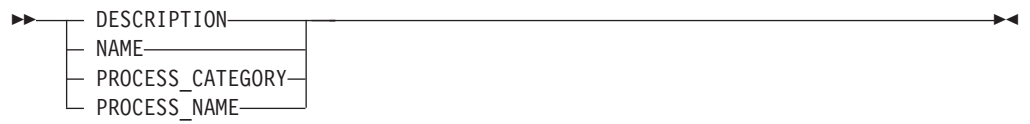
BasicPredicate



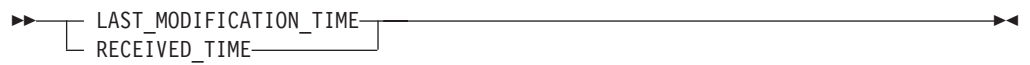
ITState



ITString



ITTimeStamp

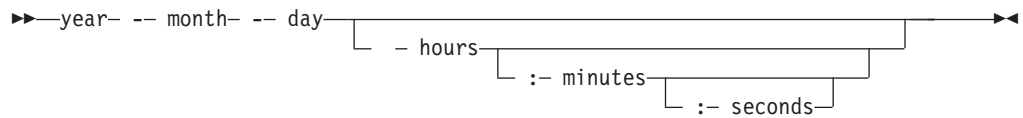


PIState



ExecutionService

TimeStamp



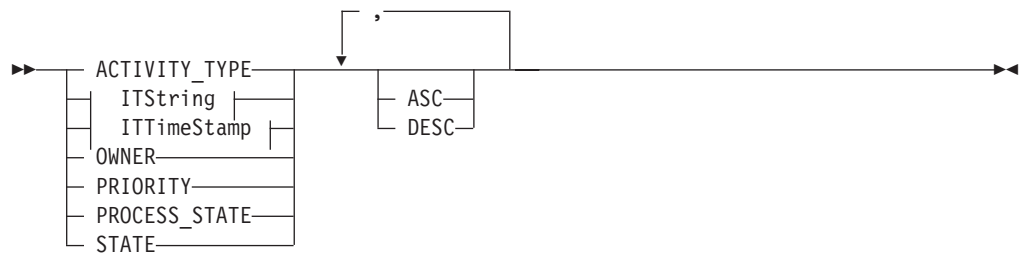
Process instance notifications can be sorted. A process instance notification sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState or the PISState diagram.

PINOrderBy



The number of process instance notifications to be retrieved can be restricted via a threshold which specifies the maximum number of process instance notifications to be returned to the client. That threshold is applied after the activity instance notifications have been sorted according to the sort criteria specified. Note that the process instance notifications are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process instance notification is:

- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceNotifications(
    FmcjExecutionServiceHandle          service,
    char const *                         filter,
    char const *                         sortCriteria,
    unsigned long const *                threshold,
    FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++

```
APIRET QueryProcessInstanceNotifications(
    string const *                       filter,
    string const *                       sortCriteria,
    unsigned long const *                threshold,
    vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java

```
public abstract
ProcessInstanceNotification[] queryProcessInstanceNotifications(
    String          filter,
    String          sortCriteria,
    Integer         threshold ) throws FmcException
```

```

COBOL

FmcjESQueryProcInstNotifs.

CALL
    "FmcjExecutionServiceQueryProcessInstanceNotifications"
    USING
    BY VALUE
        serviceValue
        filter
        sortCriteria
        threshold
    BY REFERENCE
        notifications
    RETURNING
        intReturnValue.
    
```

Parameters

- filter* Input. The filter criteria which characterize the process instance notifications to be retrieved.
- items* Input/Output. The qualifying vector of process instance notifications.
- service* Input. A handle to the service object representing the session with the execution server.
- sortCriteria* Input. The sort criteria to be applied to the process instance notifications found.
- threshold* Input. The threshold which defines the maximum number of process instance notifications to be returned to the client.

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstanceNotification[]
The qualifying process instance notifications.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is not applicable to process instance notifications.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are not applicable to process instance notifications.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instance notifications to be returned

exceeds the maximum size allowed for query results - see the `MAXIMUM_QUERY_MESSAGE_SIZE` definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query process instances (C)” on page 536.
- For a C++ example, see “Query process instances (C++)” on page 538.
- For a Java example, see “Query process instances (Java)” on page 539.
- For a COBOL example, see “Query process instances (COBOL)” on page 542.

QueryProcessInstances()

This API call retrieves the current process instances the user has access to from the MQSeries Workflow execution server (action call).

In C, C++, and COBOL, any process instances retrieved are appended to the supplied vector. If you want to read the current process instances only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

A filter on process instances is specified as a character string containing a filter predicate:

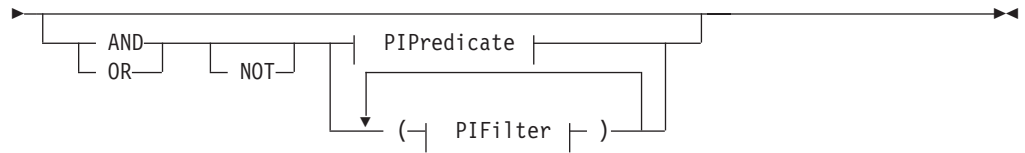
Notes:

1. A *string* constant is to be enclosed in single quotes (').
A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

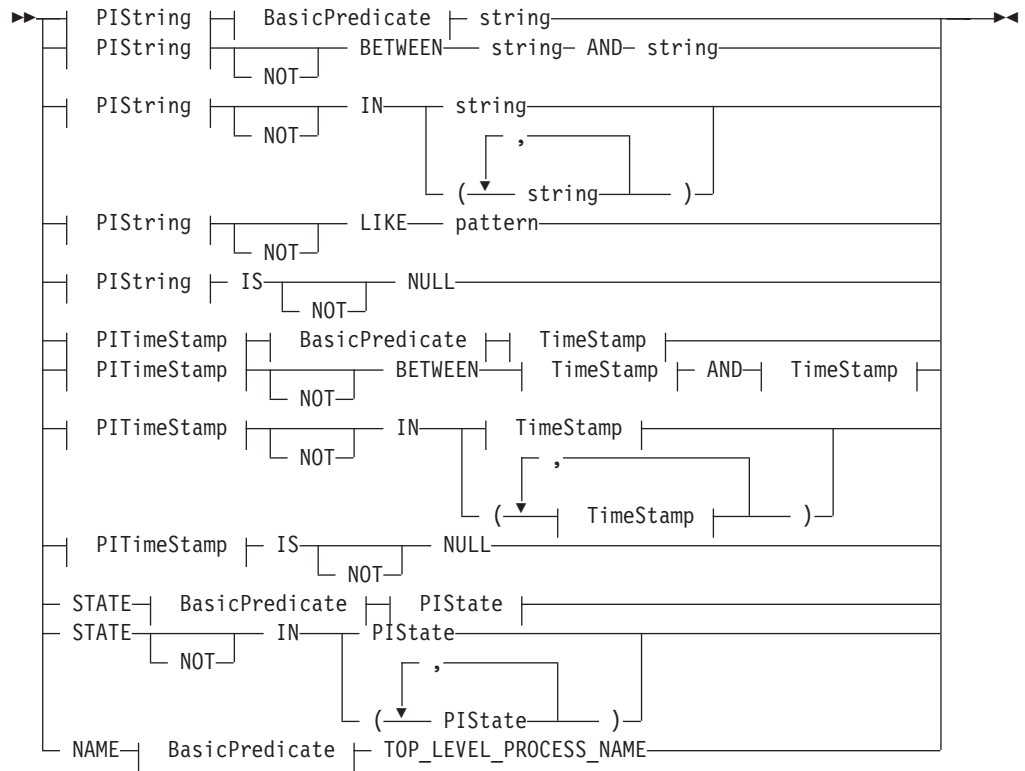
PIFilter



ExecutionService



PIPredicate



BasicPredicate



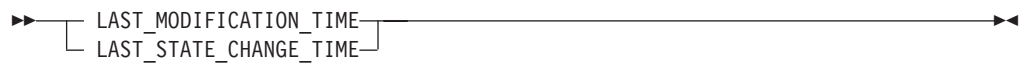
PIState



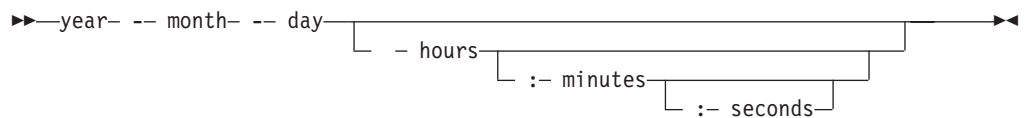
PIString



PITimeStamp



TimeStamp

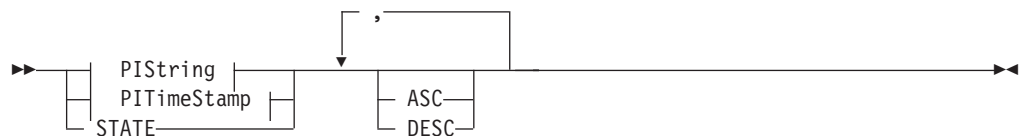


Process instances can be sorted. A process instance sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PIState diagram.

PIOrderBy



The number of process instances to be retrieved can be restricted via a threshold which specifies the maximum number of process instances to be returned to the client. That threshold is applied after the process instances have been sorted

ExecutionService

according to the sort criteria specified. Note that the process instances are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process instance is:

- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- StartTime
- State
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstances(
    FmcjExecutionServiceHandle service,
    char const * filter,
    char const * sortCriteria,
    unsigned long const * threshold,
    FmcjProcessInstanceVectorHandle * instances )
```

```
C++
APIRET QueryProcessInstances(
    string const * filter,
    string const * sortCriteria,
    unsigned long const * threshold,
    vector<FmcjProcessInstance> & instances ) const
```

Java

```
public abstract
ProcessInstance[] queryProcessInstances(
    String filter,
    String sortCriteria,
    Integer threshold ) throws FmcException
```

COBOL

```
FmcjESQueryProcInsts.

CALL "FmcjExecutionServiceQueryProcessInstances"
    USING
    BY VALUE
        serviceValue
        filter
        sortCriteria
        threshold
    BY REFERENCE
        instances
    RETURNING
        intReturnValue.
```

Parameters

| | |
|---------------------|--|
| <i>filter</i> | Input. The filter criteria which characterize the process instances to be retrieved. |
| <i>instances</i> | Input/Output. The qualifying vector of process instances. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the process instances found. |
| <i>threshold</i> | Input. The threshold which defines the maximum number of process instances to be returned to the client. |

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstance[]

The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is not applicable to process instances.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are not applicable to process instances.

FMC_ERROR_INVALID_THRESHOLD(807)

The specified threshold is invalid.

ExecutionService

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of process instances to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query process instances (C)” on page 536.
- For a C++ example, see “Query process instances (C++)” on page 538.
- For a Java example, see “Query process instances (Java)” on page 539.
- For a COBOL example, see “Query process instances (COBOL)” on page 542.

QueryProcessTemplateLists()

This API call retrieves the current process template lists the user has access to from the MQSeries Workflow execution server (action call).

In C, C++, and COBOL, any process template lists retrieved are appended to the supplied vector. If you want to read the current process template lists only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplateLists(
    FmcjExecutionServiceHandle service,
    FmcjProcessTemplateListVectorHandle * lists )
```

C++

```
APIRET QueryProcessTemplateLists(
    vector<FmcjProcessTemplateList> & lists ) const
```

Java

```
public abstract
ProcessTemplateList[] queryProcessTemplateLists() throws FmcException
```

COBOL

```
FmcjESQueryProcTemplLists.
    CALL "FmcjExecutionServiceQueryProcessTemplateLists"
        USING
        BY VALUE
        serviceValue
        BY REFERENCE
        lists
        RETURNING
        intReturnValue.
```

Parameters

lists Input/Output. The vector of process template lists.
service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

ProcessTemplateList[]

The qualifying process template lists.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of process template lists to be returned exceeds the

ExecutionService

maximum size allowed for query results - see the `MAXIMUM_QUERY_MESSAGE_SIZE` definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query worklists (C)” on page 522.
- For a C++ example, see “Query worklists (C++)” on page 524.
- For a Java example, see “Query worklists (Java)” on page 525.
- For a COBOL example, see “Query worklists (COBOL)” on page 528.

QueryProcessTemplates()

This API call retrieves the current process templates from the MQSeries Workflow execution server (action call).

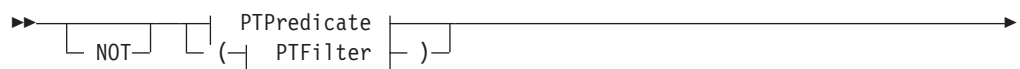
In C, C++, and COBOL, any process templates retrieved are appended to the supplied vector. If you want to read the current process templates only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

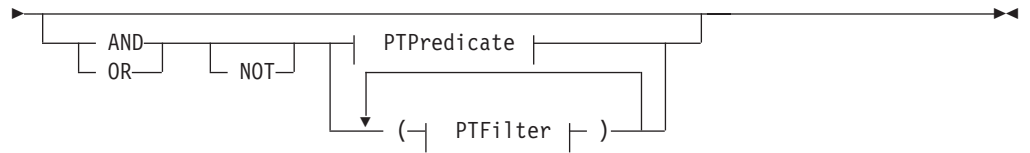
A filter on process templates is specified as a character string containing a filter predicate:

Notes:

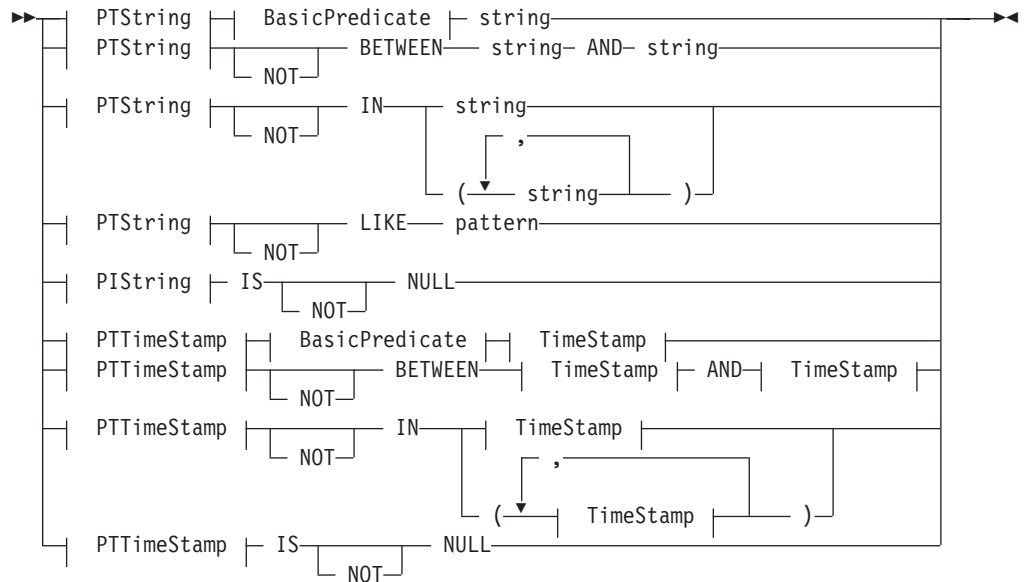
1. A *string* constant is to be enclosed in single quotes (`'`).
A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (`?`) represents any single character.
 - The asterisk (`*`) represents a string of zero or more characters.
 - The escape character is backslash (`\`) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
3. When using filters in API calls, be aware that due to the different order of special characters, numbers, uppercase letters, and lowercase letters, in ASCII and EBCDIC, calls like `QueryProcessTemplates()` can return different results on OS/390 compare to LAN servers.

PTFilter





PTPredicate



BasicPredicate



PTString

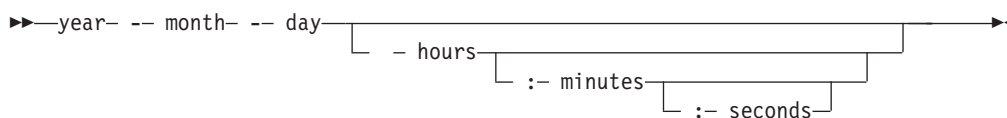


PTimeStamps



ExecutionService

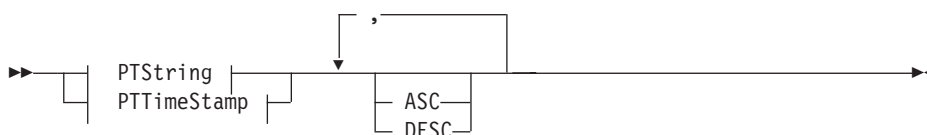
TimeStamp



Process templates can be sorted. A process template sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTOrderBy



The number of process templates to be retrieved can be restricted via a threshold which specifies the maximum number of process templates to be returned to the client. That threshold is applied after the process templates have been sorted according to the sort criteria specified. Note that the process templates are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process template is:

- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplates(
    FmcjExecutionServiceHandle      service,
    char const *                    filter,
    char const *                    sortCriteria,
    unsigned long const *           threshold,
    FmcjProcessTemplateVectorHandle * templates )
```

C++

```
APIRET QueryProcessTemplates(
    string const *                  filter,
    string const *                  sortCriteria,
    unsigned long const *           threshold,
    vector<FmcjProcessTemplate> & templates ) const
```

Java

```
public abstract
ProcessTemplates[] queryProcessTemplates(
    String filter,
    String sortCriteria,
    Integer threshold ) throws FmcException
```

COBOL

```
FmcjESQueryProcTempls.

    CALL      "FmcjExecutionServiceQueryProcessTemplates"
            USING
            BY VALUE
            serviceValue
            filter
            sortCriteria
            threshold
            BY REFERENCE
            templates
            RETURNING
            intReturnValue.
```

Parameters

| | |
|---------------------|--|
| <i>filter</i> | Input. The filter criteria which characterize the process templates to be retrieved. |
| <i>service</i> | Input. A handle to the service object representing the session with the execution server. |
| <i>sortCriteria</i> | Input. The sort criteria to be applied to the process templates found. |
| <i>templates</i> | Input/Output. The qualifying vector of process templates. |
| <i>threshold</i> | Input. The threshold which defines the maximum number of process templates to be returned to the client. |

Return type

ExecutionService

APIRET

The return code from this API call - see return codes below.

ProcessTemplate[]

The qualifying process templates.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is not applicable to process templates.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are not applicable to process templates.

FMC_ERROR_INVALID_THRESHOLD(807)

The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of process templates to be returned exceeds the maximum size allowed for query results - see the `MAXIMUM_QUERY_MESSAGE_SIZE` definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see "Query process instances (C)" on page 536.
- For a C++ example, see "Query process instances (C++)" on page 538.
- For a Java example, see "Query process instances (Java)" on page 539.
- For a COBOL example, see "Query process instances (COBOL)" on page 542.

QueryWorkitems()

This API call retrieves the work items the user has access to from the MQSeries Workflow execution server (action call).

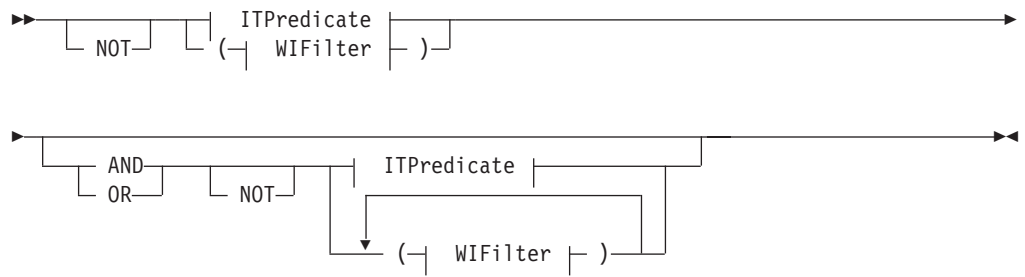
In C, C++, and COBOL, any work items retrieved are appended to the supplied vector. If you want to read the current work items only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or erase all elements of the vector in the C++ API.

The work items to be retrieved can be characterized by a filter. A work item filter is specified as a character string:

Notes:

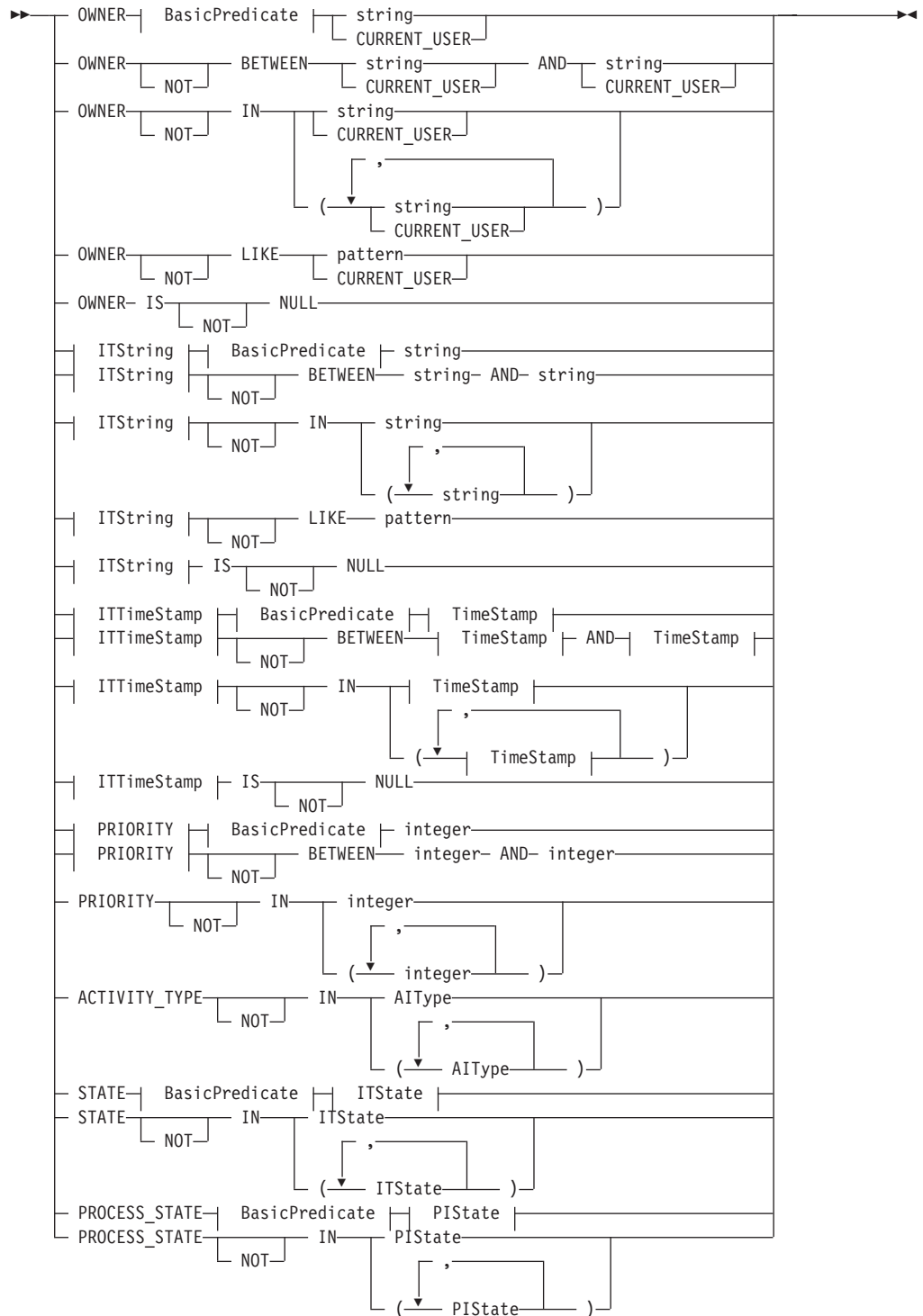
1. A *string* constant is to be enclosed in single quotes (').
 A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
 - The question mark (?) represents any single character.
 - The asterisk (*) represents a string of zero or more characters.
 - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

WIFilter



ITPredicate

ExecutionService



AType



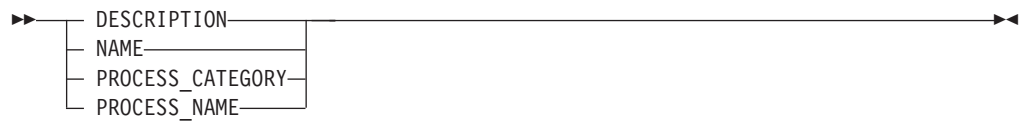
BasicPredicate



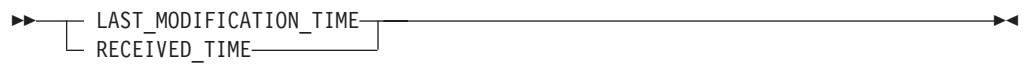
ITState



ITString



ITTimeStamp

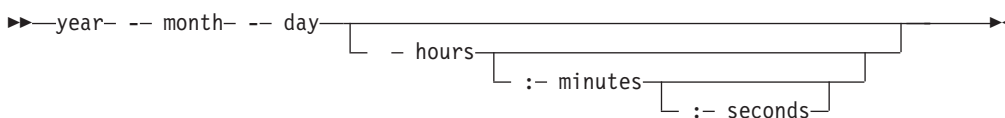


PIState



ExecutionService

TimeStamp



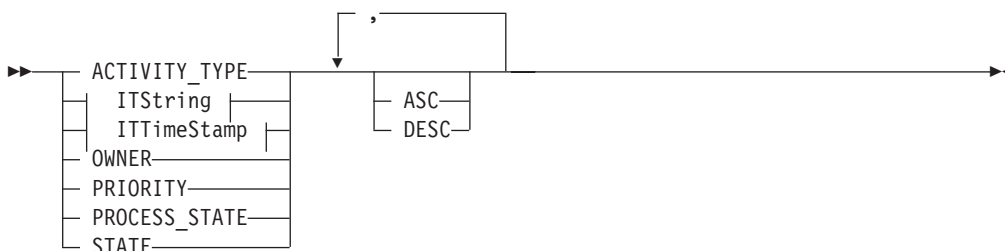
Work items can be sorted. A work item sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState or the PISState diagram.

WIOrderBy



The number of work items to be retrieved can be restricted via a threshold which specifies the maximum number of work items to be returned to the client. That threshold is applied after the items have been sorted according to the sort criteria specified. Note that the items are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each work item is:

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

- SupportTools

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorkitems(
    FmcjExecutionServiceHandle service,
    char const * filter,
    char const * sortCriteria,
    unsigned long const * threshold,
    FmcjWorkitemHandle * workitems )
```

C++

```
APIRET QueryWorkitems(
    string const * filter,
    string const * sortCriteria,
    unsigned long const * threshold,
    vector<FmcjWorkitem> & workitems ) const
```

Java

```
public abstract
WorkItem[] queryWorkItems(
    String filter,
    String sortCriteria,
    Integer threshold ) throws FmcException
```

```

COBOL

      FmcjESQueryWorkitems.

          CALL      "FmcjExecutionServiceQueryWorkitems"
                  USING
                  BY VALUE
                    serviceValue
                    filter
                    sortCriteria
                    threshold
                  BY REFERENCE
                    workitems
                  RETURNING
                    intReturnValue.
    
```

Parameters

- filter* Input. The filter criteria which characterize the work items to be retrieved.
- service* Input. A handle to the service object representing the session with the execution server.
- sortCriteria* Input. The sort criteria to be applied to the work items found.
- threshold* Input. The threshold which defines the maximum number of work items to be returned to the client.
- workitems* Input/Output. The qualifying vector of work items.

Return type

- APIRET** The return code from this API call - see return codes below.
- WorkItem[]** The qualifying work items.

Return codes/ FmcException

- FMC_OK(0)** The API call completed successfully.
- FMC_ERROR(1)** A parameter references an undefined location. For example, the address of a handle is 0.
- FMC_ERROR_INVALID_HANDLE(130)** The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
- FMC_ERROR_INVALID_FILTER(125)** The specified filter is not applicable to work items.
- FMC_ERROR_INVALID_SORT(808)** The specified sort criteria are not applicable to work items.
- FMC_ERROR_INVALID_THRESHOLD(807)** The specified threshold is invalid.
- FMC_ERROR_NOT_LOGGED_ON(106)** Not logged on.
- FMC_ERROR_QRY_RESULT_TOO_LARGE(817)** The number of work items to be returned exceeds the maximum size allowed for query results - see the **MAXIMUM_QUERY_MESSAGE_SIZE** definition in your system, system group, or domain.
- FMC_ERROR_COMMUNICATION(13)** The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query process instances (C)” on page 536.
- For a C++ example, see “Query process instances (C++)” on page 538.
- For a Java example, see “Query process instances (Java)” on page 539.
- For a COBOL example, see “Query process instances (COBOL)” on page 542.

QueryWorklists()

This API call retrieves the worklists the user has access to from the MQSeries Workflow execution server (action call).

In C, C++, and COBOL, any worklists retrieved are appended to the supplied vector. If you want to read the current worklists only, you have to clear the vector before you issue this API call. This means that you should set the vector handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorklists(
    FmcjExecutionServiceHandle service,
    FmcjWorklistVectorHandle * lists )

```

```

C++
APIRET QueryWorklists( vector<FmcjWorklist> & lists ) const

```

ExecutionService

Java

```
public abstract  
WorkList[] queryWorkLists() throws FmcException
```

COBOL

```
FmcjESQueryWorklists.  
  
CALL "FmcjExecutionServiceQueryWorklists"  
    USING  
    BY VALUE  
    serviceValue  
    BY REFERENCE  
    lists  
    RETURNING  
    intReturnValue.
```

Parameters

lists Input/Output. The vector of worklists.
service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.
WorkList[] The qualifying worklists.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

The number of worklists to be returned exceeds the maximum size allowed for query results - see the **MAXIMUM_QUERY_MESSAGE_SIZE** definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example, see “Query worklists (C)” on page 522.
- For a C++ example, see “Query worklists (C++)” on page 524.
- For a Java example, see “Query worklists (Java)” on page 525.
- For a COBOL example, see “Query worklists (COBOL)” on page 528.

Receive()

This API call allows for receiving data pushed by an MQSeries Workflow execution server or for receiving a response on an asynchronous request.

A correlation ID can be used to receive a specific response. To receive any data sent, it must be a 0 (NULL) pointer or specify FMCJ_NO_CORRELID. Note that the correlation ID is set on return provided that no 0 pointer is passed. This means that it has to be reset for each request.

The timeout value specifies how long the application should wait at a maximum for some data to arrive. If no data arrives, a timeout error is indicated. A timeout value of -1 indicates an indefinite wait time.

If data is successfully received, the execution data contains the data sent and can be used for updating objects or for creating new objects. See “ExecutionData” on page 234 for API calls supported by the execution data object.

The following enumeration types can be used to determine the contents of the execution data received:

| | |
|-------------|-----------------------------|
| C | FmcjExecutionDataKindEnum |
| C++ | FmcjExecutionData::KindEnum |
| Java | Not supported |

The enumeration constants can take the following values; it is strongly advised to use the symbolic names instead of the associated integer values.

| | |
|--------------------------|---|
| NotSet(0) | Indicates that nothing is known about the content of the execution data. |
| C | Fmc_DART_NotSet |
| C++ | FmcjExecutionData::NotSet |
| COBOL | Fmc-DART-NotSet |
| Terminate(2) | Indicates that receiving data can end. |
| C | Fmc_DART_Terminate |
| C++ | FmcjExecutionData::Terminate |
| COBOL | Fmc-DART-Terminate |
| ItemDeleted(1000) | Indicates that a work item, an activity instance notification, or a process instance notification has been deleted. |
| C | Fmc_DART_ItemDeleted |
| C++ | FmcjExecutionData::ItemDeleted |
| COBOL | Fmc-DART-ItemDeleted |
| Workitem(1002) | Indicates that a work item has been created or updated. |
| C | Fmc_DART_Workitem |

ExecutionService

| | | |
|---|-------|---|
| | C++ | FmcjExecutionData::Workitem |
| | COBOL | Fmc-DART-Workitem |
| ActivityInstanceNotification(1003) | | |
| Indicates that an activity instance notification has been created or updated. | | |
| | C | Fmc_DART_ActivityInstanceNotification |
| | C++ | FmcjExecutionData::ActivityInstanceNotification |
| | COBOL | Fmc-DART-ActInstNotif |
| ProcessInstanceNotification(1004) | | |
| Indicates that a process instance notification has been created or updated. | | |
| | C | Fmc_DART_ProcessInstanceNotification |
| | C++ | FmcjExecutionData::ProcessInstanceNotification |
| | COBOL | Fmc-DART-ProcInstNotif |
| ExecuteInstanceResponse(1100) | | |
| Indicates that the execution data contains the response on an ExecuteProcessInstance() request. | | |
| | C | Fmc_DART_ExecuteInstanceResponse |
| | C++ | FmcjExecutionData::ExecuteInstanceResponse |
| | COBOL | Fmc-DART-ExecuteInstResponse |
| ExecuteProgramResponse(1101) | | |
| Indicates that the execution data contains the response to an ExecuteProgram() request. | | |
| | C | Fmc_DART_ExecuteProgramResponse |
| | C++ | FmcjExecutionData::ExecuteProgramResponse |
| | COBOL | Fmc-DART-ExecuteProgResponse |

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server (present session mode)

API interface declarations

| | |
|------|---------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | not supported |

C

```
APIRET FMC_APIENTRY FmcjExecutionServiceReceive(
    FmcjExecutionServiceHandle service,
    FmcjCorrelID * correlID,
    FmcjExecutionDataHandle * data,
    signed long timeout )
```

C++

```
APIRET Receive( FmcjCorrelID * correlID,
                FmcjExecutionData & data,
                signed long timeout ) const
```

COBOL

```
FmcjESReceive.
    CALL "FmcjExecutionServiceReceive"
        USING
            BY VALUE
                serviceValue
            BY REFERENCE
                correlID
                data
            BY VALUE
                timeoutValue
        RETURNING
            intReturnValue.
```

Parameters

correlID Input/Output. The correlation ID by which this data can be correlated to a previous request. Must be a NULL (0) pointer or point to `Fmcj_No_CorrelID` if you want to receive any data.

data Output. The data sent by an MQSeries Workflow execution server.

service Input. A handle to the service object representing the present session with the execution server.

timeout Input. The maximum time period in milliseconds to wait for some data to arrive.

Return type

APIRET The return code from this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

ExecutionService

FMC_ERROR_MESSAGE_DATA(104)

The client received an unknown message.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

RemotePassthrough()

This API call can be used by an application program to establish a user session with an MQSeries Workflow execution server from within this program (activity implementation call).

When the activity implementation decides to distribute work among other programs and starts those programs as separate operating system processes, then those processes must be passed the CICS COMMAREA or IMS I/O Area in order to retrieve the information needed.

When successfully executed, a session is set up to the same execution server from which the original work item was started; the user on whose behalf the session is set up is the same one on whose behalf the original work item was started.

Note: This call will fail after the COMMAREA or I/O Area has been changed using SetOutContainer() or SetRemoteContainer().

Usage notes

- See "Activity implementation API calls" on page 121 for general information.

Authorization

Valid program identification

Required connection

None, but MQSeries Workflow program execution server must be active.

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjExecutionServiceRemotePassthrough(
    FmcjExecutionServiceHandle service )
char const *          programID )
```

C++

```
APIRET RemotePassthrough( string const & programID )
```

Java

```
public abstract
void remotePassthrough( String programID ) throws FmcException
```

COBOL

```
FmcjESRemotePassthrough.
      CALL      "FmcjExecutionServiceRemotePassthrough"
              USING
              BY VALUE
              serviceValue
              BY REFERENCE
              programID
              RETURNING
              intReturnValue.
```

Parameters

programID Input. The program identification by which the activity actually started. The program execution server knows the implementation of the activity.

service Input. A handle to the service object representing the session to be established with the execution server.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)

The program identification is invalid.

FMC_ERROR_PROGRAM_EXECUTION(126)

Passthrough was not called from a program started by an activity implementation, or the program execution server is not active.

FMC_ERROR_TOOL_FUNCTION(128)

Passthrough cannot be called from a program started by the program execution server.

FMC_ERROR_USERID_UNKNOWN(10)

The user who started the work item no longer exists.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

ExecutionService

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

TerminateReceive()

This API call causes information to be placed into the client input queue to tell that receiving data from an MQSeries Workflow execution server can end.

In this way, the receiving part of the application gets to know that receiving data can end. Any resulting actions are up to the application.

When the `correlID` parameter points to some buffer initialized to `FMCJ_NO_CORRELID`, then a correlation ID is returned which can be used to explicitly receive this data.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

None

API interface declarations

| | |
|-------------|---------------------------|
| C | <code>fmcjcrun.h</code> |
| C++ | <code>fmcjprun.hxx</code> |
| Java | not supported |

```
C
APIRET FMC_APIENTRY FmcjExecutionServiceTerminateReceive(
    FmcjExecutionServiceHandle service,
    FmcjCorrelID * correlID )
```

```
C++
APIRET TerminateReceive( FmcjCorrelID * correlID = 0 )
```


COBOL

```

FmcjESTerminateReceive.

    CALL      "FmcjExecutionServiceTerminateReceive"
              USING
              BY VALUE
                serviceValue
              BY REFERENCE
                correlID
              RETURNING
                intReturnValue.

```

Parameters

correlID Input/Output. The correlation ID by which this request can be correlated.

service Input. A handle to the service object.

Return type

APIRET The return code from this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_CORRELATION_ID(506)

The correlation ID passed is not FMCJ_NO_CORRELID.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Item actions

An Item object represents a work item or an activity instance notification or a process instance notification.

An FmcjItem or Item object represents the common aspects of work items and notifications. In C++, FmcjItem is thus the superclass of the FmcjWorkitem, FmcjActivityInstanceNotification, and FmcjProcessInstanceNotification classes and provides for all common properties and methods. In Java, Item is thus a superclass of the WorkItem, ActivityInstanceNotification, and ProcessInstanceNotification

Item

classes and provides for all common properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from `FmcjItem`. That is, common functions start with the prefix `FmcjItem`; they are also defined starting with the prefixes `FmcjWorkitem`, `FmcjActivityInstanceNotification`, and `FmcjProcessInstanceNotification`.

An item is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on an item. See “Item” on page 243 for a complete list of API calls.

Delete()

This API call deletes the specified item from the MQSeries Workflow execution server (action call).

A notification can always be deleted. A work item must be in states *Ready*, *Finished*, *ForceFinished*, or *Disabled*. If the work item is in the *Ready* state and represents the only work associated with the activity instance and when the associated process instance is not *Terminating* or *Terminated*, then deletion is rejected.

There are no impacts on the transient representation of your item; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--|
| C | <code>fmcjcrun.h</code> |
| C++ | <code>fmcjprun.hxx</code> |
| Java | <code>com.ibm.workflow.api.Item</code> |
| COBOL | <code>fmcvars.cpy, fmcperf.cpy</code> |

```
C
APIRET FMC_APIENTRY FmcjItemDelete( FmcjItemHandle hdlItem )

#define FmcjActivityInstanceNotificationDelete FmcjItemDelete
#define FmcjProcessInstanceNotificationDelete FmcjItemDelete
#define FmcjWorkitemDelete FmcjItemDelete
```

```
C++
APIRET Delete()
```

Java

```
public abstract
void delete() throws FmcException
```

COBOL

```
FmcjItemDelete.
CALL "FmcjItemDelete"
    USING
    BY VALUE
    hdItem
    RETURNING
    intReturnValue.
```

Parameters

hdlItem Input. The handle of the item to be deleted.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

FMC_ERROR_NOT_ALLOWED(507)

The item represents the only work associated with the activity instance.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ObtainProcessInstanceMonitor()

This API call retrieves the process instance monitor for the process instance the item is part of from the MQSeries Workflow execution server (action call).

When the *deep* option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also fetched from the server.

Note: *Deep* is not yet supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process administrator
- Be the process creator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Item |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjItemObtainProcessInstanceMonitor(
    FmcjItemHandle          hdItem,
    bool                    deep,
    FmcjProcessInstanceMonitorHandle * monitor)

#define FmcjActivityInstanceNotificationObtainProcessInstanceMonitor
    FmcjItemObtainProcessInstanceMonitor
#define FmcjProcessInstanceNotificationObtainProcessInstanceMonitor
    FmcjItemObtainProcessInstanceMonitor
#define FmcjWorkitem
    FmcjItemObtainProcessInstanceMonitor

```

C++

```
APIRET ObtainProcessInstanceMonitor(
    FmcjProcessInstanceMonitor & monitor,
    bool deep= false ) const
```

Java

```
public abstract
ProcessInstanceMonitor obtainProcessInstanceMonitor(
    boolean deep ) throws FmcException
```

COBOL

```
FmcjItemObtainProcInstMon.

CALL "FmcjItemObtainProcessInstanceMonitor"
    USING
    BY VALUE
    hdlItem
    deep
    BY REFERENCE
    monitor
    RETURNING
    intReturnValue.
```

Parameters

deep Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, *deep* is not yet supported.

hdlItem Input. The item whose process instance monitor is to be retrieved.

monitor Input/Output. The address of the handle to the process instance monitor or the process instance monitor object to be set.

returnCode Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstanceMonitor*/ ProcessInstanceMonitor

A pointer to the process instance monitor or the process instance monitor the item is a part of.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

Item

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ProcessInstance()

This API call retrieves the process instance the item is a part of from the MQSeries Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

In C++, when the process instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance handle already points to some object.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.Item

COBOL fmcvars.cpy, fmcperf.cpy

C

```

APIRET FMC_APIENTRY FmcjItemProcessInstance(
    FmcjItemHandle          hdlItem,
    FmcjProcessInstanceHandle * instance )

#define FmcjActivityInstanceNotificationProcessInstance
    FmcjItemProcessInstance
#define FmcjProcessInstanceNotificationProcessInstance
    FmcjItemProcessInstance
#define FmcjWorkitemProcessInstance
    FmcjItemProcessInstance

```

C++

```

APIRET ProcessInstance( FmcjProcessInstance & instance ) const

```

Java

```

public abstract
ProcessInstance processInstance() throws FmcException

```

COBOL

```

FmcjItemProcInst.

    CALL    "FmcjItemProcessInstance"
           USING
           BY VALUE
           hdlItem
           BY REFERENCE
           instance
           RETURNING
           intReturnValue.

```

Parameters

hdlItem Input. The handle of the item object to be queried.

instance Input/Output. The process instance object to be retrieved (initialized).

returnCode Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the process instance or the process instance the item is a part of.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

Item

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Refresh()

This API call refreshes the item from the MQSeries Workflow execution server (action call).

All information about the item, primary and secondary, is retrieved.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the item owner
- Work item authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.Item

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY FmcjItemRefresh( FmcjItemHandle hdlItem )

#define FmcjActivityInstanceNotificationRefresh FmcjItemRefresh
#define FmcjProcessInstanceNotificationRefresh FmcjItemRefresh
#define FmcjWorkitemRefresh FmcjItemRefresh
```

C++

```
APIRET Refresh()
```

Java

```
public abstract
void refresh() throws FmcException
```

COBOL

```
FmcjItemRefresh.

CALL "FmcjItemRefresh"
    USING
    BY VALUE
    hdlItem
    RETURNING
    intReturnValue.
```

Parameters

hdlItem Input. The handle of the item object to be refreshed.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

Item

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetDescription()

This API call sets the description of the item to the specified value (action call).

If no description is provided, the description of the item is reset to the description of the associated activity instance or process instance.

The following rules apply for specifying an item description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Item |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjItemSetDescription(
    FmcjItemHandle hdlItem,
    char const *   description )

#define FmcjActivityInstanceNotificationSetDescription
    FmcjItemSetDescription
#define FmcjProcessInstanceNotificationSetDescription
    FmcjItemSetDescription
#define FmcjWorkitemSetDescription
    FmcjItemSetDescription
```

C++

```
APIRET SetDescription( string const * description )
```

Java

```
public abstract
void setDescription( String description ) throws FmcException
```

COBOL

```
FmcjItemSetDescription.
CALL "FmcjItemSetDescription"
    USING
    BY VALUE
    hdlItem
    description
    RETURNING
    intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlItem Input. The handle of the item object whose description is to be set.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

FMC_ERROR_INVALID_DESCRIPTION(810)

The description does not conform to the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

Item

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetName()

This API call sets the name of the item (action call).

If no name is provided, the name of the item is reset to its default, the activity instance or the process instance name.

The following rules apply for specifying a work item or activity instance notification name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:
! " ' () * + , - . / : ; < = > [\] ^
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.
- You cannot use leading digits.
- You cannot use keywords AND, OR, NOT, IS, NULL, MOD, LOWER, UPPER, VALUE, SUBSTR, _BLOCK

The following rules apply for specifying a process instance notification name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : . \$
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the item owner

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.Item

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY FmcjItemSetName( FmcjItemHandle  hdlItem,
                                     char const *   name )

#define FmcjActivityInstanceNotificationSetName FmcjItemSetName
#define FmcjProcessInstanceNotificationSetName FmcjItemSetName
#define FmcjWorkitemSetName                   FmcjItemSetName
```

C++

```
APIRET SetName( string const * name )
```

Java

```
public abstract
void setName( String name ) throws FmcException
```

COBOL

```
FmcjItemSetName.

      CALL    "FmcjItemSetName"
            USING
            BY VALUE
            hdlItem
            name
            RETURNING
            intReturnValue.
```

Parameters

hdlItem Input. The handle of the item to be dealt with.
name Input. The new name of the item; can be a NULL (0) pointer or null object (Java).

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

Item

FMC_ERROR_INVALID_NAME(134)

The name does not conform to the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Transfer()

This API call transfers an item to the specified user (action call).

Notifications can always be transferred. A work item must be in states *Ready*, *InError*, *Executed*, *Suspending*, or *Suspended* and the associated process instance in states *Running*, *Suspending*, or *Suspended*.

The user who transfers the item must be the owner of the item or have work item authorization for the owner of the item and have work item authorization for the new owner.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Workitem authority for the persons to transfer from/to
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.Item

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY FmcjItemTransfer( FmcjItemHandle hdlItem,
                                       char const *   userID )

#define FmcjActivityInstanceNotificationTransfer FmcjItemTransfer
#define FmcjProcessInstanceNotificationTransfer FmcjItemTransfer
#define FmcjWorkitemTransfer                   FmcjItemTransfer
```

C++

```
APIRET Transfer( string const & userID )
```

Java

```
public abstract
void transfer( String userID ) throws FmcException
```

COBOL

```
FmcjItemTransfer.

      CALL      "FmcjItemTransfer"
              USING
              BY VALUE
              hdlItem
              userID
              RETURNING
              intReturnValue.
```

Parameters

hdlItem Input. The handle of the item object to be transferred.
userID Input. The ID of the user to whom the item is to be transferred.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item no longer exists.

Item

| | |
|--|--|
| FMC_ERROR_NEW_OWNER_ABSENT(110) | The user to whom the item is to be transferred is absent, that is, the item is not transferred. |
| FMC_ERROR_NEW_OWNER_NOT_FOUND(107) | The user to whom the item is to be transferred is unknown. |
| FMC_ERROR_NOT_AUTHORIZED(119) | Not authorized to use the API call. |
| FMC_ERROR_NOT_LOGGED_ON(106) | Not logged on. |
| FMC_ERROR_OWNER_ALREADY_ASSIGNED(133) | The user to whom the item is to be transferred does already have that item. |
| FMC_ERROR_WRONG_STATE(120) | The item or process instance is in the wrong state. |
| FMC_ERROR_COMMUNICATION(13) | The specified server cannot be reached; the server to which the connection should be established is not defined in your profile. |
| FMC_ERROR_INTERNAL(100) | An MQSeries Workflow internal error has occurred. Contact your IBM representative. |
| FMC_ERROR_MESSAGE_FORMAT(103) | An internal message format error. Contact your IBM representative. |
| FMC_ERROR_TIMEOUT(14) | Timeout has occurred. |

PersistentList actions

A PersistentList object represents a set of objects of the same type the user is authorized for. Moreover, all objects which are accessible through this list have the same characteristics. These characteristics are specified by a filter. Additionally, sort criteria can be applied and, after that, a threshold to restrict the number of objects to be transferred from a server to the client.

As the name indicates, the list definition is stored persistently. The objects contained in the list are, however, assembled dynamically when they are queried.

A persistent list can be a process template list, a process instance list, or a worklist.

An FmcjPersistentList or PersistentList object represents the common aspects of lists. In C++, FmcjPersistentList is thus the superclass of the FmcjProcessInstanceList, FmcjProcessTemplateList, and FmcjWorklist classes and provides for all common properties and methods. In Java, PersistentList is thus a superclass of the ProcessInstanceList, ProcessTemplateList, and Worklist classes and provides for all common properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefixes FmcjProcessInstanceList, FmcjProcessTemplateList, and FmcjWorklist.

A persistent list is uniquely identified by its name, type, and owner. It can be defined for general access purposes; it is then of a *public* type. Or, it can be defined for some specific user; it is then of a *private* type.

The following sections describe the actions which can be applied on a persistent list. See "PersistentList" on page 246 for a complete list of API calls.

Delete()

This API call deletes the specified persistent list from the MQSeries Workflow execution server (action call).

The transient representation of the persistent list is not impacted; in C, C++, and COBOL, you have to destruct or deallocate the transient object when it is no longer needed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.PersistentList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjPersistentListDelete( FmcjPersistentListHandle hdllist )

#define FmcjProcessInstanceListDelete FmcjPersistentListDelete
#define FmcjProcessTemplateListDelete FmcjPersistentListDelete
#define FmcjWorklistDelete           FmcjPersistentListDelete
```

C++

```
APIRET Delete()
```

Java

```
public abstract
void delete() throws FmcException
```

PersistentList

COBOL

```
FmcjPLDelete.  
  
    CALL    "FmcjPersistentListDelete"  
          USING  
          BY VALUE  
          hdllist  
          RETURNING  
          intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list to be deleted.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_DOES_NOT_EXIST(118)

The persistent list no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Refresh()

This API call refreshes the persistent list from the MQSeries Workflow execution server (action call).

All information about the persistent list is retrieved, for example, its description, its filter, or its sort criteria.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.PersistentList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjPersistentListRefresh( FmcjPersistentListHandle hdList )

#define FmcjProcessInstanceListRefresh FmcjPersistentListRefresh
#define FmcjProcessTemplateListRefresh FmcjPersistentListRefresh
#define FmcjWorklistRefresh           FmcjPersistentListRefresh
```

C++

```
APIRET Refresh()
```

Java

```
public abstract
void refresh() throws FmcException
```

COBOL

```
FmcjPLRefresh.

CALL "FmcjPersistentListRefresh"
    USING
    BY VALUE
    hdList
    RETURNING
    intReturnValue.
```

Parameters

hdList Input. The handle of the persistent list to be refreshed.

Return type

long/ APIRET The return code from this API call - see return codes below.

PersistentList

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This API call sets the description of the persistent list to the specified value (action call).

If no description is provided, the description of the persistent list is erased.

The following rules apply for specifying a persistent list description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.PersistentList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjPersistentListSetDescription( FmcjPersistentListHandle hdlList,
                                char const *          description )

#define FmcjProcessInstanceListSetDescription
    FmcjPersistentListSetDescription
#define FmcjProcessTemplateListSetDescription
    FmcjPersistentListSetDescription
#define FmcjWorklistSetDescription
    FmcjPersistentListSetDescription
```

C++

```
APIRET SetDescription( string const * description )
```

Java

```
public abstract
void setDescription( String description ) throws FmcException
```

COBOL

```
FmcjPLSetDescription.

CALL "FmcjPersistentListSetDescription"
    USING
    BY VALUE
    hdlList
    description
    RETURNING
    intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlList Input. The handle of the persistent list object whose description is to be set.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

PersistentList

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The persistent list no longer exists.

FMC_ERROR_INVALID_DESCRIPTION(810)

The description does not conform to the syntax rules.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetFilter()

This API call sets the filter of the persistent list to the specified value (action call).

If no filter is provided, the current filter of the persistent list is erased. This means that all objects authorized for will be selected via this list.

Refer to the appropriate list creation for a description of a valid filter syntax.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY
FmcjPersistentListSetFilter( FmcjPersistentListHandle  hdlList,
                           char const *              filter )

#define FmcjProcessInstanceListSetFilter FmcjPersistentListSetFilter
#define FmcjProcessTemplateListSetFilter FmcjPersistentListSetFilter
#define FmcjWorklistSetFilter           FmcjPersistentListSetFilter
```

C++

```
APIRET SetFilter( string const * filter )
```

Java

```
public abstract
void setFilter( String filter ) throws FmcException
```

COBOL

```
FmcjPLSetFilter.

CALL "FmcjPersistentListSetFilter"
     USING
     BY VALUE
     hdlList
     filter
     RETURNING
     intReturnValue.
```

Parameters

filter Input. The filter or a pointer to the filter to be set; can be a NULL (0) pointer or null object (Java).

hdlList Input. The handle of the persistent list object whose filter is to be set.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

PersistentList

FMC_ERROR_DOES_NOT_EXIST(118)

The persistent list no longer exists.

FMC_ERROR_INVALID_FILTER(125)

The specified filter is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetSortCriteria()

This API call sets the sort criteria of the persistent list to the specified value (action call).

If no sort criteria are provided, the current sort criteria of the persistent list are erased. This means that objects selected via this list will not be sorted.

Refer to the appropriate list creation for a description of a valid sort criteria syntax.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY
FmcjPersistentListSetSortCriteria( FmcjPersistentListHandle hdllist,
                                   char const *          sortCriteria )

#define FmcjProcessInstanceListSetSortCriteria
    FmcjPersistentListSetSortCriteria
#define FmcjProcessTemplateListSetSortCriteria
    FmcjPersistentListSetSortCriteria
#define FmcjWorklistSetSortCriteria
    FmcjPersistentListSetSortCriteria
```

C++

```
APIRET SetSortCriteria( string const * sortCriteria )
```

Java

```
public abstract
void setSortCriteria( String sortCriteria ) throws FmcException
```

COBOL

```
FmcjPLSetSortCriteria.

CALL "FmcjPersistentListSetSortCriteria"
    USING
    BY VALUE
    hdllist
    sortCriteria
    RETURNING
    intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list object whose sort criteria are to be set.

sortCriteria Input. The sort criteria or a pointer to the sort criteria to be set; can be a NULL (0) pointer or null object (Java).

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

PersistentList

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The persistent list no longer exists.

FMC_ERROR_INVALID_SORT(808)

The specified sort criteria are invalid.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetThreshold()

This API call sets the threshold of the persistent list to the specified value (action call).

If no threshold is provided, the threshold of the persistent list is erased. This means that all objects contained in the list will be provided when queried.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY
FmcjPersistentListSetThreshold( FmcjPersistentListHandle hdlList,
                               unsigned long const *   threshold )

#define FmcjProcessInstanceListSetThreshold FmcjPersistentListSetThreshold
#define FmcjProcessITemplateListSetThreshold FmcjPersistentListSetThreshold
#define FmcjWorklistSetThreshold           FmcjPersistentListSetThreshold
```

C++

```
APIRET SetThreshold( unsigned long const * threshold )
```

Java

```
public abstract
void setThreshold( Integer threshold ) throws FmcException
```

COBOL

```
FmcjPLSetThreshold.

      CALL      "FmcjPersistentListSetThreshold"
              USING
              BY VALUE
              hdlList
              threshold
              RETURNING
              intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list object whose threshold is to be set.

threshold Input. The threshold or a pointer to the threshold to be set; can be a NULL (0) pointer or null object (Java).

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

PersistentList

FMC_ERROR_DOES_NOT_EXIST(118)

The persistent list no longer exists.

FMC_ERROR_INVALID_THRESHOLD(807)

The threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Person actions

An FmcjPerson or a Person object represents an MQSeries Workflow user. A person is uniquely identified by its user identification.

The following sections describe the actions which can be applied on a person. See "Person" on page 247 for a complete list of API calls.

Refresh()

This API call refreshes the person from the MQSeries Workflow execution server (action call).

All information about the person, primary and secondary, is retrieved.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-----------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Person |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C  
APIRET FMC_APIENTRY FmcjPersonRefresh( FmcjPersonHandle hdlPerson )
```

```
C++
APIRET Refresh()
```

```
Java
public abstract
void refresh() throws FmcException
```

```
COBOL

      FmcjPRefresh.
          CALL      "FmcjPersonRefresh"
                  USING
                  BY VALUE
                  hdlPerson
                  RETURNING
                  intReturnValue.
```

Parameters

hdlPerson Input. The handle of the person to be refreshed.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Person

SetAbsence()

This API call sets the absence indication of the logged-on user to the specified value (action call).

When a person is absent, this person does not participate in staff resolution, that is, this person does not get assigned any work items.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-----------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Person |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjPersonSetAbsence(  
    FmcjPersonHandle hdlPerson,  
    bool              newValue )
```

C++

```
APIRET SetAbsence( bool newValue )
```

Java

```
public abstract  
void setAbsence( boolean newValue ) throws FmcException
```

COBOL

```
FmcjPSetAbsence.  
  
    CALL    "FmcjPersonSetAbsence"  
          USING  
          BY VALUE  
          hdlPerson  
          newValue  
          RETURNING  
          intReturnValue.
```

Parameters

hdlPerson Input. The handle of the person object whose absence is to be set.
newValue Input. True, if the person is denoted as absent, else false.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetSubstitute()

This API call sets the substitute of the logged-on user (action call).

The substitute must be a registered MQSeries Workflow user ID other than the logged-on user. If no substitute is provided, the substitute of the logged-on user is erased.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.Person

Person

COBOL fmcvars.cpy, fmcperf.cpy

```
C
APIRET FMC_APIENTRY FmcjPersonSetSubstitute(
    FmcjPersonHandle hdlPerson,
    char const * substitute )
```

```
C++
APIRET SetSubstitute( string const * substitute )
```

```
Java
public abstract
void setSubstitute( String substitute ) throws FmcException
```

```
COBOL

FmcjPSetSubstitute.

CALL "FmcjPersonSetSubstitute"
    USING
    BY VALUE
    hdlPerson
    substitute
    RETURNING
    intReturnValue.
```

Parameters

hdlPerson

Input. The handle of the person object whose substitute is to be set.

substitute

Input. The substitute or a pointer to the substitute to be set; can be a NULL (0) pointer or null object (Java).

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_USER(132)

The specified user ID does not correspond to the syntax rules or the user cannot be logged on and be the substitute at the same time.

- FMC_ERROR_NOT_LOGGED_ON(106)**
Not logged on.
- FMC_ERROR_USERID_UNKNOWN(10)**
The specified user ID is not a registered MQSeries Workflow user ID.
- FMC_ERROR_COMMUNICATION(13)**
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
- FMC_ERROR_INTERNAL(100)**
An MQSeries Workflow internal error has occurred. Contact your IBM representative.
- FMC_ERROR_MESSAGE_FORMAT(103)**
An internal message format error. Contact your IBM representative.
- FMC_ERROR_TIMEOUT(14)**
Timeout has occurred.

ProcessInstance actions

A ProcessInstance object represents an instance of a process template. A process instance is uniquely identified by its object identifier or by its name. Depending on the keep option when the process instance was created, the unique process instance name has been supplied by the user or has been generated by MQSeries Workflow.

The following diagram provides an overview of the possible process instance states and the actions which are allowed in those states, provided that the appropriate authority has been granted:

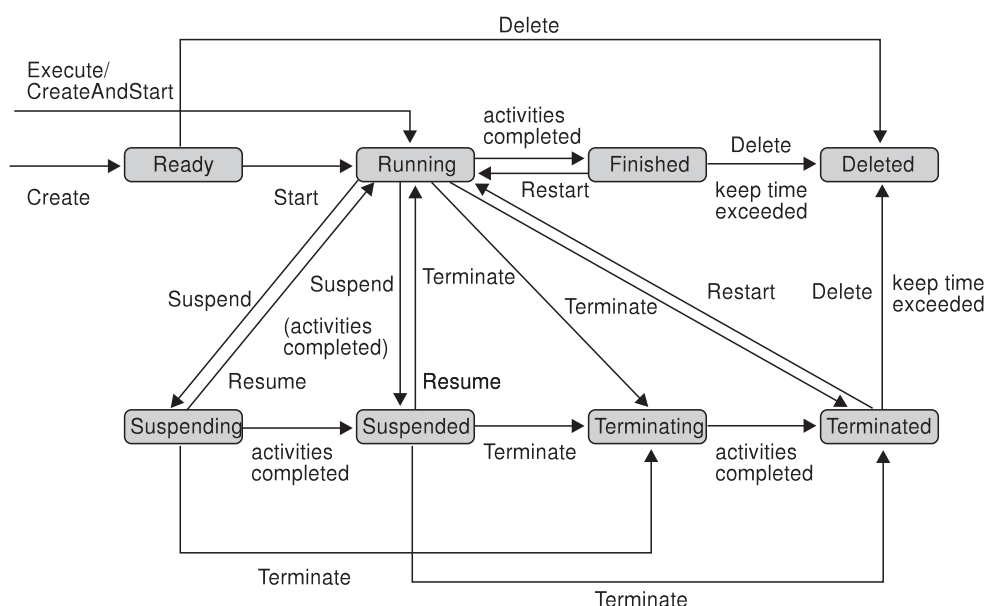


Figure 43. Process instance states

The following sections describe the actions which can be applied on a process instance. See "ProcessInstance" on page 251 for a complete list of API calls.

ProcessInstance

Delete()

This API call deletes the specified process instance from the MQSeries Workflow execution server (action call).

The process instance must be a top-level process and in states *Ready*, *Finished*, or *Terminated*. The creator can delete the process instance as long as it has not been started.

There are no impacts on your transient representation of the process instance; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceDelete( FmcjProcessInstanceHandle hdlInstance )
```

C++

```
APIRET Delete()
```

Java

```
public abstract  
void delete() throws FmcException
```

COBOL

```

FmcjPIDelete.

      CALL      "FmcjProcessInstanceDelete"
              USING
              BY VALUE
              hd1Instance
              RETURNING
              intReturnValue.

```

Parameters

hdlInstance Input. The handle of the process instance to be deleted.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the process instance from the MQSeries Workflow execution server (action call).

In C++, when the container object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely

ProcessInstance

responsible for the ownership of objects, that is, it is not checked whether the container handle already points to some object.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjProcessInstanceInContainer(  
    FmcjProcessInstanceHandle    hdlInstance,  
    FmcjReadWriteContainerHandle * input )
```

C++

```
APIRET InContainer( FmcjReadWriteContainer & input )
```

Java

```
public abstract  
ReadWriteContainer inContainer() throws FmcException
```

COBOL

```

FmcjPIInCtnr.

CALL      "FmcjProcessInstanceInContainer"
          USING
          BY VALUE
            hd1Instance
          BY REFERENCE
            inputValue
          RETURNING
            intReturnValue.

```

Parameters

hdlInstance Input. The handle of the process instance object whose input container is to be retrieved.

input Input/Output. The address of the input container or of its handle or the input container of the process instance to be set.

Return type

long/ APIRET The result of making this API call - see return codes below.

ReadWriteContainer

The input container of the process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ProcessInstance

ObtainMonitor()

This API call obtains a monitor for the process instance from the MQSeries Workflow execution server (action call).

When the deep option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also fetched from the server.

Note: Deep is not yet supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjProcessInstanceObtainMonitor(
    FmcjProcessInstanceHandle    hdlInstance,
    bool                          deep,
    FmcjProcessInstanceMonitorHandle * monitor )
```

```
C++
APIRET ObtainMonitor( FmcjProcessInstanceMonitor & monitor,
    bool                          deep= false )
```

Java

```
public abstract
ProcessInstanceMonitor obtainMonitor( boolean deep ) throws FmcException
```

COBOL

```
FmcjPIObtainMon.

CALL      "FmcjProcessInstanceObtainMonitor"
          USING
          BY VALUE
          hd1Instance
          deep
          BY REFERENCE
          monitor
          RETURNING
          intReturnValue.
```

Parameters

| | |
|--------------------|--|
| <i>deep</i> | Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is not yet supported. |
| <i>hdlInstance</i> | Input. The handle of the process instance object whose monitor is to be retrieved. |
| <i>monitor</i> | Input/Output. The address of the monitor handle or the monitor of the process instance to be set. |
| <i>returnCode</i> | Input/Output. A pointer to the result of the method call - see return codes below. |

Return type

APIRET The return code from this API call - see return codes below.

InstanceMonitor*/ProcessInstanceMonitor

A pointer to the process instance monitor or the process instance monitor.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

ProcessInstance

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

PersistentObject()

This API call retrieves the process instance identified by the passed object identifier from the MQSeries Workflow execution server (action call).

The MQSeries Workflow execution server from which the process instance is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information, primary and secondary, of the process instance.

In C++, when the process instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance handle already points to some object. In Java, a process instance is newly created; the execution service acts as a factory.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjProcessInstancePersistentObject(
    FmcjExecutionServiceHandle service,
    char const * oid,
    FmcjProcessInstanceHandle * hdlInstance )
```

C++

```
APIRET PersistentObject( FmcjExecutionService const & service,
    string const & oid )
```

Java

```
public abstract
    ProcessInstance ExecutionService.persistentObject( String oid )
throws FmcException
```

COBOL

```
FmcjPIPersistentObj.
    CALL "FmcjProcessInstancePersistentObject"
        USING
            BY VALUE
                serviceValue
                oid
            BY REFERENCE
                hdlInstance
        RETURNING
            intReturnValue.
```

Parameters

hdlInstance Input/Output. The address of the handle to the process instance object to be set.

oid Input. The object identifier of the process instance to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The result of making this API call - see return codes below.

ProcessInstance

The process instance retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

ProcessInstance

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_INVALID_OID(805)

The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Refresh()

This API call refreshes the process instance from the MQSeries Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceRefresh( FmcjProcessInstanceHandle hdlInstance )
```

```
C++
APIRET Refresh()
```

```
Java
public abstract
void refresh() throws FmcException
```

```
COBOL

      FmcjPIRefresh.

      CALL      "FmcjProcessInstanceRefresh"
              USING
              BY VALUE
              hdInstance
              RETURNING
              intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be refreshed.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

ProcessInstance

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This API call restarts the process instance on the MQSeries Workflow execution server (action call).

Only *finished* or *terminated* top-level process instances can be restarted. The process administrator does not change. The process starter is set to the requester of this API call.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceRestart( FmcjProcessInstanceHandle hdlInstance )
```

C++

```
APIRET Restart()
```

Java

```
public abstract  
void restart() throws FmcException
```

COBOL

```

FmcjPIRestart.

      CALL      "FmcjProcessInstanceRestart"
              USING
              BY VALUE
              hdInstance
              RETURNING
              intReturnValue.

```

Parameters

hdlInstance Input. The handle of the process instance object to be restarted.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_KIND(501)

The process instance is no top-level process instance.

FMC_ERROR_WRONG_STATE(120)

The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Resume()

This API call resumes processing of a suspended or suspending process instance (action call).

All non-autonomous subprocesses with respect to control autonomy are also resumed, if the deep option is true.

ProcessInstance

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceResume( FmcjProcessInstanceHandle hdlInstance,  
                           bool deep )
```

C++

```
APIRET Resume( bool deep )
```

Java

```
public abstract  
void resume( boolean deep ) throws FmcException
```

COBOL

```
FmcjPIResume.  
  
CALL "FmcjProcessInstanceResume"  
    USING  
    BY VALUE  
    hdlInstance  
    deep  
    RETURNING  
    intReturnValue.
```

Parameters

- deep* Input. If *deep* is true, processing of all non-autonomous subprocesses is also resumed.
- hdlInstance* Input. The handle of the process instance to be resumed.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetDescription()

This API call sets the description of the process instance to the specified value (action call).

If no description is provided, the description of the process instance is erased.

The following rules apply for specifying a process instance description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

ProcessInstance

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjProcessInstanceSetDescription(
    FmcjProcessInstanceHandle hdlInstance,
    char const * description )
```

```
C++
APIRET SetDescription( string const * description )
```

```
Java
public abstract
void setDescription( String description ) throws FmcException
```

```
COBOL
FmcjPISetDescription.
CALL "FmcjProcessInstanceSetDescription"
    USING
    BY VALUE
    hdlInstance
    description
    RETURNING
    intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlInstance Input. The handle of the process instance object whose description is to be set.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_INVALID_DESCRIPTION(810)

The description does not conform to the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetName()

This API call sets the name of the process instance to the specified value (action call).

The process instance must still be in the *Ready* state.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : . \$
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

ProcessInstance

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstance |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceSetName( FmcjProcessInstanceHandle hdlInstance,  
                             char const * name )
```

C++

```
APIRET SetName( string const & name )
```

Java

```
public abstract  
void setName( String name ) throws FmcException
```

COBOL

```
FmcjPISetName.  
  
CALL "FmcjProcessInstanceSetName"  
    USING  
    BY VALUE  
    hdlInstance  
    name  
    RETURNING  
    intReturnValue.
```

Parameters

| | |
|--------------------|---|
| <i>hdlInstance</i> | Input. The handle of the process instance object whose name is to be set. |
| <i>name</i> | Input. The name to be set. |

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

| | |
|--------------------------------------|--|
| FMC_ERROR_INVALID_HANDLE(130) | The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type. |
| FMC_ERROR_DOES_NOT_EXIST(118) | The process instance no longer exists. |
| FMC_ERROR_INVALID_NAME(134) | The name does not conform to the syntax rules. |
| FMC_ERROR_NOT_AUTHORIZED(119) | Not authorized to use the API call. |
| FMC_ERROR_NOT_LOGGED_ON(106) | Not logged on. |
| FMC_ERROR_NOT_UNIQUE(121) | The process instance name is not unique. |
| FMC_ERROR_WRONG_STATE(120) | The process instance is in the wrong state. |
| FMC_ERROR_COMMUNICATION(13) | The specified server cannot be reached; the server to which the connection should be established is not defined in your profile. |
| FMC_ERROR_INTERNAL(100) | An MQSeries Workflow internal error has occurred. Contact your IBM representative. |
| FMC_ERROR_MESSAGE_FORMAT(103) | An internal message format error. Contact your IBM representative. |
| FMC_ERROR_TIMEOUT(14) | Timeout has occurred. |

Start()

This API call starts a ready process instance (action call).

When successfully executed, the starter is set to the requestor of this action and the process administrator is determined.

When initial values are to be passed to the process instance to be started, an input container can be provided (see also `FmcjProcessInstance::InContainer()`). When the process instance requires input and is started without specifying an input container, the input-container values are not set. So, when, for example, input-container values are queried from within an activity implementation, `FMC_ERROR_MEMBER_NOT_SET` is returned.

For Java programs, `start2()` additionally allows to pass an input container.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

ProcessInstance

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h
C++ fmcjprun.hxx
Java com.ibm.workflow.api.ProcessInstance
COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY  
FmcjProcessInstanceStart( FmcjProcessInstanceHandle hdlInstance,  
                          FmcjReadWriteContainerHandle input )
```

C++

```
APIRET Start()  
  
APIRET Start( FmcjReadWriteContainer const & input )
```

Java

```
public abstract  
void start() throws FmcException  
  
public abstract  
void start2( ReadWriteContainer input ) throws FmcException
```

COBOL

```
FmcjPIStart.  
  
CALL "FmcjProcessInstanceStart"  
     USING  
     BY VALUE  
     hdlInstance  
     inputValue  
     RETURNING  
     intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be started.
input Input. The input container of the process instance.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Suspend()

This API call suspends (temporarily stops) the process instance (action call).

The process instance must be in state *Running*. All non-autonomous subprocesses with respect to control autonomy are also suspended if the deep option is true. Autonomous subprocesses are not considered.

The process instance remains in state *Suspending* as long as there are running program activity implementations or suspending non-autonomous subprocesses. When the activity implementations completed their executions and when the non-autonomous subprocesses reached the *Suspended* state, the process instance is put into the *Suspended* state.

Optionally, a date may be specified up to when the process instance is suspended; it is then automatically resumed, together with the non-autonomous subprocesses, if the deep option had been specified.

For Java programs, `suspend2()` additionally allows to provide a date at which the process instance is automatically resumed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process administration authorization
- Be the process administrator
- Be the system administrator

ProcessInstance

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h
C++ fmcjprun.hxx
Java com.ibm.workflow.api.ProcessInstance
COBOL fmcvars.cpy, fmcperf.cpy

```
C
APIRET FMC_APIENTRY FmcjProcessInstanceSuspend(
    FmcjProcessInstanceHandle hdlInstance,
    bool deep )

APIRET FMC_APIENTRY FmcjProcessInstanceSuspendUntil(
    FmcjProcessInstanceHandle hdlInstance,
    FmcjCDateTime const * time,
    bool deep )
```

```
C++
APIRET Suspend( bool deep )

APIRET Suspend( FmcjDateTime const & time, bool deep )
```

```
Java
public abstract
void suspend( boolean deep ) throws FmcException

public abstract
void suspend2( Calendar time, boolean deep ) throws FmcException
```

```
COBOL
FmcjPISuspend.

CALL "FmcjProcessInstanceSuspend"
    USING
    BY VALUE
    hdlInstance
    deep
    RETURNING
    intReturnValue.
```

Parameters

deep Input. An indicator whether also non-autonomous subprocesses are to be suspended.

hdlInstance Input. The handle of the process instance object to be suspended.

time Input. The date/time or a pointer to the date/time until which the process instance is to be suspended.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Terminate()

This API call terminates a process instance and all of its non-autonomous subprocesses (action call).

The process instance must be in states *Running*, *Suspended*, or *Suspending*.

The process instance is put into state terminating as long as there are running activity implementations or terminating non-autonomous subprocesses. When the activity implementations completed their executions or when the non-autonomous subprocesses terminated, the process instance is put into the *Terminated* state. When the process instance has reached the *Terminated* state, it is deleted depending on the setting of the “delete finished items” option.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process administration authorization
- Be the process administrator
- Be the system administrator

ProcessInstance

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h
C++ fmcjprun.hxx
Java com.ibm.workflow.api.ProcessInstance
COBOL fmcvars.cpy, fmcperf.cpy

```
C  
APIRET FMC_APIENTRY  
FmcjProcessInstanceTerminate( FmcjProcessInstanceHandle hdlInstance )
```

```
C++  
APIRET Terminate()
```

```
Java  
public abstract  
void terminate() throws FmcException
```

```
COBOL  
  
FmcjPITerminate.  
  
CALL "FmcjProcessInstanceTerminate"  
USING  
BY VALUE  
hdlInstance  
RETURNING  
intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be terminated.

Return type

long/ APIRET The result of making this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

| | |
|--------------------------------------|--|
| FMC_ERROR_INVALID_HANDLE(130) | The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type. |
| FMC_ERROR_DOES_NOT_EXIST(118) | The process instance no longer exists. |
| FMC_ERROR_NOT_AUTHORIZED(119) | Not authorized to use the API call. |
| FMC_ERROR_NOT_LOGGED_ON(106) | Not logged on. |
| FMC_ERROR_WRONG_STATE(120) | The process instance is in the wrong state. |
| FMC_ERROR_COMMUNICATION(13) | The specified server cannot be reached; the server to which the connection should be established is not defined in your profile. |
| FMC_ERROR_INTERNAL(100) | An MQSeries Workflow internal error has occurred. Contact your IBM representative. |
| FMC_ERROR_MESSAGE_FORMAT(103) | An internal message format error. Contact your IBM representative. |
| FMC_ERROR_TIMEOUT(14) | Timeout has occurred. |

ProcessInstanceList actions

A process instance list represents a set of process instances. All process instances which are accessible through this list have the same characteristics. These characteristics are specified by a filter. Additionally, sort criteria can be applied and, after that, a threshold to restrict the number of process instances to be transferred from the execution server to the client.

The process instance list definition is stored persistently.

A process instance list is uniquely identified by its name, type, and owner. It can be defined for general access purposes; it is then of a *public* type. Or, it can be defined for some specific user; it is then of a *private* type.

Other lists that can be defined are process template lists or worklists. FmcjPersistentList or PersistentList represents the common properties of all lists.

In C++, FmcjProcessInstanceList is a subclass of the FmcjPersistentList class and inherits all properties and methods. In the Java language, ProcessInstanceList is thus a subclass of the PersistentList class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjProcessInstanceList.

The following sections describe the actions which can be applied on a process instance list. See "ProcessInstanceList" on page 255 for a complete list of API calls.

QueryProcessInstances()

This API call retrieves the primary information for all process instances characterized by the specified process instance list from the MQSeries Workflow execution server (action call).

ProcessInstanceList

From the set of qualifying process instances, only those are retrieved the user is authorized for. The user is authorized for a process instance if the process instance:

- Does not belong to any category
- Does belong to a category and the user has global process authorization or global process administration authorization or selected process authorization or selected process administration authorization for that category

The primary information that is retrieved for each process instance is:

- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- StartTime
- State
- SuspensionExpirationTime
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

In C, C++, and COBOL, any process instances retrieved are appended to the supplied vector of process instances. If you want to read those process instances only which are currently included in the process instance list, you have to clear the vector before you issue this API call.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessInstanceList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjProcessInstanceListQueryProcessInstances(
    FmcjProcessInstanceListHandle    hdList,
    FmcjProcessInstanceVectorHandle * instances )
```

C++

```
APIRET QueryProcessInstances(
    vector<FmcjProcessInstance> & instances ) const
```

Java

```
public abstract
ProcessInstance[] queryProcessInstances() throws FmcException
```

COBOL

```
FmcjPILQueryProcInsts.

CALL      "FmcjProcessInstanceListQueryProcessInstances"
          USING
          BY VALUE
          hdllist
          BY REFERENCE
          instances
          RETURNING
          intReturnValue.
```

Parameters

hdlList Input. The handle of the process instance list to be queried.
instances Input/Output. The vector of qualifying process instances.

Return type

long/ APIRET The result returned by this API call - see return codes below.
ProcessInstance[]
The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance list no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

ProcessInstanceList

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see “Query worklists (C)” on page 522
- For a C++ example see “Query worklists (C++)” on page 524
- For a COBOL example, see “Query worklists (COBOL)” on page 528

ProcessInstanceNotification actions

A ProcessInstanceNotification object represents a notification on a process instance assigned to a user.

Other items assigned to users are activity instance notifications and work items. FmcjItem or Item represents the common properties of all items.

In C++, FmcjProcessInstanceNotification is thus a subclass of the FmcjItem class and inherits all properties and methods. In Java, ProcessInstanceNotification is thus a subclass of the Item class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjItem. That is, common functions start with the prefix FmcjItem; they are also defined starting with the prefix FmcjProcessInstanceNotification.

A process instance notification is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on a process instance notification. See “ProcessInstanceNotification” on page 256 for a complete list of API calls.

PersistentObject()

This API call retrieves the process instance notification identified by the passed object identifier from the MQSeries Workflow execution server (action call).

The MQSeries Workflow execution server from which the process instance notification is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the process instance notification.

In C++, when the process instance notification object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance notification handle already points to some object. In Java, a process instance notification is newly created; the execution service acts as a factory.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Be the item owner
- Work item authorization

- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjProcessInstanceNotificationPersistentObject(
    FmcjExecutionServiceHandle service,
    char const * oid,
    FmcjProcessInstanceNotificationHandle * hdlItem )
    
```

```

C++
APIRET PersistentObject( FmcjExecutionService const & service,
    string const & oid )
    
```

```

Java
public abstract
    ProcessInstanceNotification
    ExecutionService.processInstanceNotification( String oid )
    throws FmcException
    
```

```

COBOL
FmcjPINPersistentObj.
CALL "FmcjProcessInstanceNotificationPersistentObject"
    USING
    BY VALUE
        serviceValue
        oid
    BY REFERENCE
        hdlItem
    RETURNING
        intReturnValue.
    
```

Parameters

- hdlItem* Input/Output. The address of the handle to the process instance notification object to be set.
- oid* Input. The object identifier of the process instance notification to be retrieved.
- service* Input. The service object representing the session with the execution server.

ProcessInstanceNotification

Return type

long/ APIRET The result returned by this API call - see return codes below.

ProcessInstanceNotification

The process instance notification retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance notification .

FMC_ERROR_INVALID_OID(805)

The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ProcessTemplate actions

A ProcessTemplate object is the frozen state of a process model from which it is created via translation. All program definitions and data structures referenced by the process model are copied into the process template (early binding). Subprocesses are bound later. Their definitions are only located during execution.

A process template is uniquely identified by its object identifier or by its name and a valid-from date. This *valid-from date* determines since when the process template can be used to create process instances.

When process templates are queried from the execution server, then only currently valid process templates are returned.

The following sections describe the actions which can be applied on a process template. See "ProcessTemplate" on page 257 for a complete list of API calls.

CreateAndStartInstance()

This API call creates a process instance from the specified process template and starts the resulting process instance (action call).

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : . \$
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQSeries Workflow, the following applies:

- If no or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged;

`FmcjProcessInstance::Name()` returns the actual name of the process instance created. The newly created process instance contains the primary attribute values only.

When initial values are to be passed to the process instance to be created and started, an input container can be provided - see also `FmcjProcessTemplate::InContainer()`. When a process instance that requires input is started without specifying an input container, the input-container values are not set. When, for example, input-container values are queried from within an activity implementation, `FMC_ERROR_MEMBER_NOT_SET` is returned.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

See `createAndStartInstance`; additionally allows to pass an input container.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|------|---|
| C | <code>fmcjcrun.h</code> |
| C++ | <code>fmcjprun.hxx</code> |
| Java | <code>com.ibm.workflow.api.ProcessTemplate</code> |

ProcessTemplate

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateAndStartInstance(  
    FmcjProcessTemplateHandle    hdlTemplate,  
    char const *                 name,  
    char const *                 reserved1,  
    char const *                 reserved2,  
    FmcjReadWriteContainerHandle input,  
    bool                         keepName,  
    FmcjProcessInstanceHandle *  newInstance )
```

C++

```
APIRET CreateAndStartInstance(  
    string const *               name,  
    string const *               reserved1,  
    string const *               reserved2,  
    FmcjProcessInstance &       newInstance,  
    bool                         keepName = false ) const;  
  
APIRET CreateAndStartInstance(  
    string const *               name,  
    string const *               reserved1,  
    string const *               reserved2,  
    FmcjReadWriteContainer const & input,  
    FmcjProcessInstance &       newInstance,  
    bool                         keepName = false ) const;
```

Java

```
public abstract  
ProcessInstance createAndStartInstance(  
    String name,  
    String reserved1,  
    String reserved2,  
    boolean keepName ) throws FmcException  
  
public abstract  
ProcessInstance createAndStartInstance2(  
    String name,  
    String reserved1,  
    String reserved2,  
    ReadWriteContainer input,  
    boolean keepName ) throws FmcException
```


XML

```

<!-- ProcessTemplateCreateAndStart ===== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
  ( ProcTemplName,
    ProcInstName,
    KeepName,
    ProcInstInputData ) >
<!ELEMENT ProcTemplName          (#PCDATA) >
<!ELEMENT ProgInstName           (#PCDATA) >
<!ELEMENT KeepName               (#PCDATA) >
  <!-- Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >

<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
  ( ProcessInstance
    | Exception ) >
<!ELEMENT ProcessInstance
  ( ProcInstID,
    ProcInstName,
    ProcInstParentName?,
    ProcInstTopLevelName,
    ProcInstDescription?,
    ProcInstState,
    LastStateChangeTime,
    LastModificationTime,
    ProcTemplID,
    ProcTemplName,
    Icon,
    Category? ) >

<!ELEMENT ProcInstID              (#PCDATA) >
<!ELEMENT ProcInstDescription     (#PCDATA) >
<!ELEMENT ProcInstName            (#PCDATA) >
<!ELEMENT ProcInstParentName      (#PCDATA) >
<!ELEMENT ProcInstTopLevelName    (#PCDATA) >
<!ELEMENT ProcInstState           (#PCDATA) >
  <!-- Expected values: {Ready,Running,Finished,Terminated,
    Suspended, Terminating,
    Suspending,Deleted} -->
<!ELEMENT LastModificationTime    (#PCDATA) >
<!ELEMENT LastStateChangeTime     (#PCDATA) >
<!ELEMENT ProcTemplID            (#PCDATA) >
<!ELEMENT ProcTemplName          (#PCDATA) >
<!ELEMENT Icon                   (#PCDATA) >
<!ELEMENT Category               (#PCDATA) >
<!ELEMENT Exception
  ( Rc, Parameters, MessageText?, Origin ) >
  <!-- Message text is optional, as it will be ignored
    in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
  (Parameter*) >
<!ELEMENT Parameter              (#PCDATA) >
<!ELEMENT Rc                    (#PCDATA) >
<!ELEMENT MessageText           (#PCDATA) >
<!ELEMENT Origin                (#PCDATA) >

```

ProcessTemplate

COBOL

```
FmcjPTCreateAndStartInst.  
  
    CALL    "FmcjProcessTemplateCreateAndStartInstance"  
           USING  
           BY VALUE  
             hdlTemplate  
             name  
             reserved1  
             reserved2  
             inputValue  
             keepName  
           BY REFERENCE  
             newInstance  
           RETURNING  
             intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object to be used.
input Input. The input container of the process instance.
keepName Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated.
name Input. The name of the process instance to be created and started.
newInstance Input/Output. The newly created and started process instance.
returnCode Input/Output. The result of calling this method - see below.
reserved1/reserved2 Input. Pass a 0 (NULL) pointer or an empty string.

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the newly created and started process instance or the newly created and started process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_INVALID_NAME(134)

The specified process instance name does not comply with the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_NOT_UNIQUE(121)

The name of the process instance is not unique.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)

The document is not a valid XML document.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)

The document is not a valid MQSeries Workflow XML document.

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)

The type of the container is incorrect.

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)

The specified data member is not part of the container.

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)

The type of the data member value passed is incorrect.

XML example

```
<ProcessTemplateCreateAndStartInstance>
  <ProcTempName>OnlineCreditRequest</ProcTempName>
  <ProgInstName>Credit Request #658321</ProgInstName>
  <KeepName>true</KeepName>
  <ProcInstInputData>
    <CreditData>
      <!-- here comes the data for data structure CreditData -->
    </CreditData>
  </ProcInstInputData>
</ProcessTemplateCreateAndStartInstance>

<ProcessTemplateCreateAndStartInstanceResponse>
  <ProcessInstance>
    <ProcInstID>42424242EFEFEFEF</ProcInstID>
    <ProcInstName>Credit Request#658321</ProcInstName>
    <ProcInstTopLevelName>Credit Request#658321</ProcInstTopLevelName>
    <ProcInstDescription>Sample description</ProcInstDescription>
    <ProcInstState>Finished</ProcInstState>
    <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
    <LastModificationTime>1999-05-19 23:40:00</LastModTime>
    <ProcTempID>84848484EFEFEFEF</ProcTempID>
    <ProcTempName>OnlineCreditRequest</ProcTempName>
    <Icon>fmcpcr</Icon>
    <Category>Finance</Category>
  </ProcessInstance>
</ProcessTemplateCreateAndStartInstanceResponse>
```

CreateInstance()

This API call creates a process instance from the specified process template (action call).

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

ProcessTemplate

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : . \$
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQSeries Workflow, the following applies:

- If no name or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged;
`FmcjProcessInstance::Name()` returns the actual name of the process instance created. The newly created process instance contains the primary attribute values only.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplate |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateInstance(
    FmcjProcessTemplateHandle hdlTemplate,
    char const * name,
    char const * reserved1,
    char const * reserved2,
    bool keepName,
    FmcjProcessInstanceHandle * newInstance )
```

C++

```
APIRET CreateInstance(
    string const * name,
    string const * reserved1,
    string const * reserved2,
    FmcjProcessInstance & newInstance,
    bool keepName = false ) const
```

Java

```
public abstract
ProcessInstance createInstance(
    String name,
    String reserved1,
    String reserved2,
    boolean keepName ) throws FmcException
```

COBOL

```
FmcjPTCreateInst.
    CALL "FmcjProcessTemplateCreateInstance"
        USING
            BY VALUE
                hdlTemplate
                name
                reserved1
                reserved2
                keepName
            BY REFERENCE
                newInstance
        RETURNING
            intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object to be used.

keepName Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated.

name Input. The name of the process instance to be created.

newInstance Input/Output. The newly created process instance.

reserved1/reserved2 Input. Pass a 0 (NULL) pointer or an empty string.

returnCode Input/Output. The result of calling this method - see below.

ProcessTemplate

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the newly created process instance or the newly created process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_INVALID_NAME(134)

The specified process instance name does not comply with the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_NOT_UNIQUE(121)

The name of the process instance is not unique.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Delete()

This API call deletes the specified process template(s) from the execution server (action call).

Since process templates are versioned, you can specify whether you want to delete the currently valid process template, the past versions of the process template, or the future versions of the process template. When all options are specified, all versions of the process template are deleted. Deletion always applies to the currently existing process templates only.

For Java programs, delete2() additionally allows for specifying the versions to be deleted.

There are no impacts on your transient representation of the process template; in C, C++, or COBOL, you must destruct or deallocate the transient object when it is no longer needed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process modeling authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplate |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjProcessTemplateDelete( FmcjProcessTemplateHandle hdlTemplate,
                           bool pastVersions,
                           bool currentVersion,
                           bool futureVersions )
```

C++

```
APIRET Delete( bool pastVersions = true,
               bool currentVersion= true,
               bool futureVersions= true )
```

Java

```
public abstract
void delete() throws FmcException

public abstract
void delete2( boolean pastVersions,
              boolean currentVersion,
              boolean futureVersions ) throws FmcException
```

ProcessTemplate

COBOL

```
FmcjPTDelete.  
  
    CALL    "FmcjProcessTemplateDelete"  
          USING  
          BY VALUE  
          hd1Template  
          RETURNING  
          intReturnValue.
```

Parameters

- currentVersion* Input. An indication whether the current version of this process template is to be deleted.
- futureVersions* Input. An indication whether future versions of this process template are to be deleted.
- hdlTemplate* Input. The handle of the process template to be deleted.
- pastVersions* Input. An indication whether past versions of this process template are to be deleted.

Return type

long/ APIRET The result returned by this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template or its specified versions no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ExecuteProcessInstance()

This API call creates a process instance from the specified process template and executes the resulting process instance (action call).

The program execution server must have been started so that the activity implementations are executed.

This API call can be issued synchronously or asynchronously. When called synchronously, the process instance should be fast enough to complete within the application wait time. When called asynchronously, a user context can be specified to correlate the response received later. Additionally, a correlation ID can be received which can be used to wait for the specific response. If a buffer to hold the correlation ID is specified, then it must initially point to FMCJ_NO_CORRELID.

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
* ? " ; : . \$
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQSeries Workflow, the following applies:

- If no or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name\$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged;

FmcjProcessInstance::Name() returns the actual name of the process instance created. The newly created process instance contains all attributes, primary and secondary.

When initial values are to be passed to the process instance to be created and started, an input container can be provided - see also FmcjProcessTemplate::InContainer(). When a process instance that requires input is started without specifying an input container, the input-container values are not set. When, for example, input-container values are queried from within an activity implementation, FMC_ERROR_MEMBER_NOT_SET is returned.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

On completion, the executed process instance and its output container are returned. The process instance contains values for the primary attributes only. In case of process instance termination, a container is not returned, that is, a 0-pointer or empty container is returned.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

ProcessTemplate

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|------|---------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | not supported |

```
C
APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstance(
    FmcjProcessTemplateHandle    hdlTemplate,
    char const *                  name,
    char const *                  reserved1,
    char const *                  reserved2,
    FmcjReadWriteContainerHandle  input,
    bool                           keepName,
    FmcjProcessInstanceHandle *   newInstance,
    FmcjReadOnlyContainerHandle * output
)

APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstanceAsync(
    FmcjProcessTemplateHandle    hdlTemplate,
    char const *                  name,
    char const *                  reserved1,
    char const *                  reserved2,
    FmcjReadWriteContainerHandle  input,
    bool                           keepName,
    FmcjCorrelID *                correlID,
    char const *                  userContext
)
```

C++

```

APIRET ExecuteProcessInstance(
    FmcjProcessInstance &
    FmcjReadOnlyContainer &
    string const *
    string const *
    string const *
    bool
    newInstance,
    output,
    name = 0,
    reserved1 = 0,
    reserved2 = 0,
    keepName = false ) const

APIRET ExecuteProcessInstance(
    FmcjReadWriteContainer const &
    FmcjProcessInstance &
    FmcjReadOnlyContainer &
    string const *
    string const *
    string const *
    bool
    input,
    newInstance,
    output,
    name = 0,
    reserved1 = 0,
    reserved2 = 0,
    keepName = false ) const

APIRET ExecuteProcessInstanceAsync(
    string const *
    string const *
    string const *
    bool
    FmcjCorrelID *
    string const *
    name = 0,
    reserved1 = 0,
    reserved2 = 0,
    keepName = false,
    correlID = 0,
    userContext = 0 )

APIRET ExecuteProcessInstanceAsync(
    FmcjReadWriteContainer const &
    string const *
    string const *
    string const *
    bool
    FmcjCorrelID *
    string const *
    input,
    name = 0,
    reserved1 = 0,
    reserved2 = 0,
    keepName = false,
    correlID = 0,
    userContext = 0 )

```

ProcessTemplate

XML

```
<!-- ProcessTemplateExecute ===== -->
<!ELEMENT ProcessTemplateExecute
  ( ProcTemplName,
    ProcInstName,
    KeepName,
    ProcInstInputData ) >
<!ELEMENT ProcTemplName          (#PCDATA) >
<!ELEMENT ProgramName            (#PCDATA) >
<!ELEMENT KeepName               (#PCDATA) >
<!-- Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
<!ELEMENT ProcessTemplateExecuteResponse
  ( ( ProcessInstance,
    ProcInstOutputData )
  | Exception ) >
<!ELEMENT ProcessInstance
  ( ProcInstID,
    ProcInstName,
    ProcInstParentName?,
    ProcInstTopLevelName,
    ProcInstDescription?,
    ProcInstState,
    LastStateChangeTime,
    LastModificationTime,
    ProcTemplID,
    ProcTemplName,
    Icon,
    Category? ) >

<!ELEMENT ProcInstID              (#PCDATA) >
<!ELEMENT ProcInstDescription     (#PCDATA) >
<!ELEMENT ProcInstName            (#PCDATA) >
<!ELEMENT ProcInstParentName      (#PCDATA) >
<!ELEMENT ProcInstTopLevelName    (#PCDATA) >
<!ELEMENT ProcInstState           (#PCDATA) >
<!-- Expected values: { Ready, Running,
  Finished, Terminated,
  Suspended, Terminating,
  Suspending, Deleted } -->
<!ELEMENT LastModificationTime    (#PCDATA) >
<!ELEMENT LastStateChangeTime    (#PCDATA) >
<!ELEMENT ProcTemplID            (#PCDATA) >
<!ELEMENT ProcTemplName          (#PCDATA) >
<!ELEMENT Icon                   (#PCDATA) >
<!ELEMENT Category               (#PCDATA) >
<!ELEMENT ProcInstOutputData      (%CONTAINER;) >
<!ELEMENT Exception
  (Rc, Parameters, MessageText?, Origin) >
<!-- Message text is optional, as it will be ignored
  in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
  (Parameter*) >
<!ELEMENT Parameter              (#PCDATA) >
<!ELEMENT Rc                    (#PCDATA) >
<!ELEMENT MessageText           (#PCDATA) >
<!ELEMENT Origin                (#PCDATA) >
```

COBOL

```

FmcjPTEExecuteProcInst.

    CALL    "FmcjProcessTemplateExecuteProcessInstance"
           USING
           BY VALUE
           hdlTemplate
           name
           reserved1
           reserved2
           inputValue
           keepName
           BY REFERENCE
           newInstance
           outputValue
           RETURNING
           intReturnValue.

FmcjPTEExecuteProcInstAsync.

    CALL    "FmcjProcessTemplateExecuteProcessInstanceAsync"
           USING
           BY VALUE
           hdlTemplate
           name
           reserved1
           reserved2
           inputValue
           keepName
           BY REFERENCE
           correlID
           BY VALUE
           userContext
           RETURNING
           intReturnValue.

```

Parameters

| | |
|----------------------------|---|
| <i>correlID</i> | Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response. |
| <i>hdlTemplate</i> | Input. The handle of the process template object to be used. |
| <i>input</i> | Input. The input container of the process instance. |
| <i>keepName</i> | Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated. |
| <i>name</i> | Input. The name of the process instance to be executed. |
| <i>newInstance</i> | Input/Output. The executed process instance. |
| <i>output</i> | Output. The output container of the process instance. |
| <i>returnCode</i> | Input/Output. The result of calling this method - see below. |
| <i>reserved1/reserved2</i> | |
| <i>userContext</i> | Input. Pass a 0 (NULL) pointer or an empty string. Input. A user-defined context which can be used for correlation. |

Return type

APIRET The return code from this API call - see return codes below.

ProcessInstance*

A pointer to the newly created and executed process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

ProcessTemplate

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_INVALID_CORRELATION_ID

The specified correlation ID does not point to FMCJ_NO_CORRELID.

FMC_ERROR_INVALID_NAME(134)

The specified process instance name does not comply with the syntax rules.

FMC_ERROR_INVALID_USER_CONTEXT(819)

The specified user context is longer than 254 characters.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_NOT_UNIQUE(121)

The name of the process instance is not unique.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)

The document is not a valid XML document.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)

The document is not a valid MQSeries Workflow XML document.

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)

The type of the container is incorrect.

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)

The specified data member is not part of the container.

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)

The type of the data member value passed is incorrect.

XML example

```
<ProcessTemplateExecute>
  <ProcTemplateName>OnlineCreditRequest</ProcTemplateName>
  <ProcInstName>Credit Request #658321</ProcInstName>
  <KeepName>true</KeepName>
  <ProcInstInputData>
    </CreditData>
    <!-- here comes the data for data structure CreditData -->
    </CreditData>
  </ProcInstInputData>
</ProcessTemplateExecute>
```

```

<ProcessTemplateExecuteResponse>
  <ProcessInstance>
    <ProcInstID>42424242EFEFEFEF</ProcInstID>
    <ProcInstName>Credit Request #658321</ProcInstName>
    <ProcInstTopLevelName>Credit Request #658321</ProcInstTopLevelName>
    <ProcInstDescription>Sample description</ProcInstDescription>
    <ProcInstState>Finished</ProcInstState>
    <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
    <LastModificationTime>1999-05-19 23:40:00</LastModTime>
    <ProcTempID>84848484FEFEFEFE</ProcTempID>
    <ProcTempName>OnlineCreditRequest</ProcTempName>
    <Icon>fmcpcrd</Icon>
    <Category>Finance</Category>
  </ProcessInstance>
  <ProcInstOutputData>
    <CreditData>
      <!-- here comes the data structure data -->
    </CreditData>
  </ProcInstOutputData>
</ProcessTemplateExecuteResponse>

```

InitialInContainer()

This API call retrieves the input container associated with the process template from the MQSeries Workflow execution server (action call).

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplate |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjProcessTemplateInitialInContainer(
    FmcjProcessTemplateHandle    hdlTemplate,
    FmcjReadWriteContainerHandle * input )

```

```

C++
APIRET InContainer( FmcjReadWriteContainer & input )

```

ProcessTemplate

Java

```
public abstract  
ReadWriteContainer initialInContainer() throws FmcException
```

COBOL

```
FmcjPTInitialInCtnr.  
  
CALL      "FmcjProcessTemplateInitialInContainer"  
          USING  
          BY VALUE  
          hd1Template  
          BY REFERENCE  
          inputValue  
          RETURNING  
          intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object whose input container is to be retrieved.

input Input/Output. The address of the input container handle or the input container of the process template to be set.

Return type

long/ APIRET The result returned by this API call - see return codes below.

ReadWriteContainer

The input container of the process template.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

PersistentObject()

This API call retrieves the process template identified by the passed object identifier from the MQSeries Workflow execution server (action call).

The MQSeries Workflow execution server from which the process template is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the process template.

In C++, when the process template object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process template handle already points to some object. In Java, a process template is newly created; the execution service acts as a factory.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|---------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ExecutionService |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjProcessTemplatePersistentObject(
    FmcjExecutionServiceHandle service,
    char const *          oid,
    FmcjProcessTemplateHandle * hdlTemplate )
  
```

```

C++
APIRET PersistentObject( FmcjExecutionService const & service,
    string const &          oid )
  
```

ProcessTemplate

Java

```
public abstract  
    ProcessTemplate ExecutionService.processTemplate( String oid )  
    throws FmcException
```

COBOL

```
FmcjPTPersistentObj.  
  
    CALL      "FmcjProcessTemplatePersistentObject"  
            USING  
            BY VALUE  
                serviceValue  
                oid  
            BY REFERENCE  
                hdlTemplate  
            RETURNING  
                intReturnValue.
```

Parameters

hdlTemplate Input/Output. The address of the handle to the process template object to be set.

oid Input. The object identifier of the process template to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The result returned by this API call - see return codes below.

ProcessTemplate

The process template retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_INVALID_OID(805)

The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ProgramTemplate()

This API call retrieves the program template identified by the name passed from the MQSeries Workflow execution server (action call).

A program template comprises data about its associated input and output containers, implementation data for all specified platforms and various other properties. In case *structures from activity* was specified for the program during Buildtime, no input or output container information is available; any container can be passed to the program when executed.

When containers are provided for a program template, they are initial containers. Therefore, no default values are set for data members. Also, predefined data members are not set.

The result of calling this API call is dependent on the system where the request is executed, because there are values returned that can be inherited from the system.

The program template is versioned within the context of the corresponding process template.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplate |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```

C
APIRET FMC_APIENTRY FmcjProcessTemplateProgramTemplate(
    FmcjProcessTemplateHandle  hdlTemplate,
    char const *                programName,
    FmcjProgramTemplateHandle * program )

```

ProcessTemplate

C++

```
APIRET ProgramTemplate( string const &      programName,  
                        FmcjProgramTemplate & program ) const
```

Java

```
public abstract  
ProgramTemplate programTemplate( String programName )  
throws FmcException
```

COBOL

```
FmcjPTProgramTempl.  
  
CALL      "FmcjProcessTemplateProgramTemplate"  
          USING  
          BY VALUE  
            hdlTemplate  
            programName  
          BY REFERENCE  
            programValue  
          RETURNING  
            intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template where a program template is to be retrieved.

program Input/Output. The program template retrieved.

programName Input. The name of the program template to be retrieved.

Return type

APIRET The result of calling this API call - see return codes below.

ProgramTemplate/ProgramTemplate*

The program template or a pointer to the program template retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_BACK_LEVEL_OBJECT

The request can only be executed on process templates translated after MQSeries Workflow 3.2.1 has been installed.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid or the program template does not exist within the process template.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Refresh()

This API call refreshes the process template from the MQSeries Workflow execution server (action call).

All information about the process template - primary and secondary - is retrieved.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--------------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplate |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjProcessTemplateRefresh( FmcjProcessTemplateHandle hdlTemplate )
```

C++

```
APIRET Refresh()
```

ProcessTemplate

Java

```
public abstract  
void refresh() throws FmcException
```

COBOL

```
FmcjPRefresh.  
  
CALL    "FmcjProcessTemplateRefresh"  
        USING  
        BY VALUE  
        hdITemplate  
        RETURNING  
        intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object to be refreshed.

Return type

long/ APIRET The result returned by this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template no longer exists or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ProcessTemplateList actions

A process template list represents a set of process templates. All process templates which are accessible through this list have the same characteristics. These characteristics are specified by a filter. Additionally, sort criteria can be applied and, after that, a threshold to restrict the number of process templates to be transferred from the execution server to the client.

The process template list definition is stored persistently.

A process template list is uniquely identified by its name, type, and owner. It can be defined for general access purposes; it is then of a *public* type. Or, it can be defined for some specific user; it is then of a *private* type.

Other lists that can be defined are process instance lists or worklists. FmcjPersistentList or PersistentList represents the common properties of all lists.

In C++, FmcjProcessTemplateList is thus a subclass of the FmcjPersistentList class and inherits all properties and methods. In Java, ProcessTemplateList is thus a subclass of the PersistentList class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjProcessTemplateList.

The following sections describe the actions which can be applied on a process template list. See “ProcessTemplateList” on page 259 for a complete list of API calls.

QueryProcessTemplates()

This API call retrieves the primary information for all process templates characterized by the specified process template list from the MQSeries Workflow execution server (action call).

From the set of qualifying process templates, only those are retrieved, the user is authorized for. The user is authorized for a process template if the process template:

- Does not belong to any category
- Does belong to a category and the user has global process authorization or global process administration authorization or selected process authorization or selected process administration authorization for that category

The primary information that is retrieved for each process template is:

- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name
- ValidFromTime

ProcessTemplateList

In C, C++, and COBOL, any process templates retrieved are appended to the supplied vector of process templates. If you want to read those process templates only which are currently included in the process template list, you have to clear the vector before you make this API call.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|--|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.ProcessTemplateList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjProcessTemplateListQueryProcessTemplates(
    FmcjProcessTemplateListHandle    hdList,
    FmcjProcessTemplateVectorHandle * templates )
```

```
C++
APIRET QueryProcessTemplates(
    vector<FmcjProcessTemplate> & templates ) const;
```

```
Java
public abstract
ProcessTemplate[] queryProcessTemplates() throws FmcException
```

```
COBOL
FmcjPTLQueryProcTempls.
CALL "FmcjProcessTemplateListQueryProcessTemplates"
    USING
    BY VALUE
        hdList
    BY REFERENCE
        templates
    RETURNING
        intReturnValue.
```


Parameters

hdlList Input. The handle of the process template list to be queried.
templates Input/Output. The vector of qualifying process templates.

Return type

long/ APIRET The result returned by this API call - see return codes below.
ProcessTemplate[]
 The qualifying process templates.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process template list no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see "Query worklists (C)" on page 522
- For a C++ example see "Query worklists (C++)" on page 524
- For a Java example, see "Query worklists (Java)" on page 525
- For a COBOL example, see "Query worklists (COBOL)" on page 528

ProgramTemplate actions

A ProgramTemplate object represents the definition of a program within a process template.

A program template is uniquely identified by its name and the process template in which it is contained. This means that it is versioned via the containing process template.

The following sections describe the actions which can be applied to a program template. See "ProgramTemplate" on page 261 for a complete list of API calls.

ProgramTemplate

Execute()

This API call requests the execution of the specified program template on the program execution server (PES) of the system where the user is logged on.

This API call can be issued synchronously or asynchronously. When called synchronously, the program should be fast enough to complete within the application wait time. When called asynchronously, a user context can be specified to correlate the response received later. Additionally, a correlation ID can be received which can be used to wait for the specific response. If a buffer to hold the correlation ID is specified, then it must initially point to FMCJ_NO_CORRELID.

Depending on the *input container access* definition of the program template, an input container must be specified for execution. Depending on the *output container access* definition, an output container can be specified to hold the values returned by program execution. If an output container is specified for the program and the output parameter is not provided, the output container defined for the program is used. When *structures from activity* is defined, containers passed can be of any type, since the program thus states that it is able to handle any container. When *structures from activity* is not defined, any containers passed must conform to the type defined in the program settings.

Initial containers returned by `FmcjProcessTemplate::ProgramTemplate()` do not contain any default values. When initial values are to be passed to the program, they can be set in the input or output container before making this API call.

The output container, if any, is returned on completion. The `_RC` data member of the output container denotes the program return code. The RC is thus available only if an output container is defined.

Specification of a priority influences OS/390 Workload management. The priority must be a value between 0 and 9.

Notes:

1. `Passthrough()` cannot be called from a program executed via the PES.
2. The output container is an input/output parameter. For Java, this means that it is passed as an input parameter and is the return value of the `Execute()` method; the input parameter is not changed.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be logged on

Required connection

MQSeries Workflow program execution server

API interface declarations

| | |
|------|---|
| C | <code>fmcjcrun.h</code> |
| C++ | <code>fmcjprun.hxx</code> |
| Java | <code>com.ibm.workflow.api.ProgramTemplate</code> |

COBOL fmcjvars.cpy, fmcjperf.cpy

C

```

APIRET FMC_APIENTRY FmcjProgramTemplateExecute(
    FmcjProcessTemplateHandle hdlTemplate,
    FmcjReadWriteContainerHandle input,
    FmcjReadWriteContainerHandle output )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptions(
    FmcjProcessTemplateHandle hdlTemplate,
    unsigned long priority,
    FmcjReadWriteContainerHandle input,
    FmcjReadWriteContainerHandle output )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteAsync(
    FmcjProcessTemplateHandle hdlTemplate,
    FmcjReadWriteContainerHandle input,
    FmcjReadWriteContainerHandle output,
    FmcjCorrelID * correlID,
    char const * userContext )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptionsAsync(
    FmcjProcessTemplateHandle hdlTemplate,
    unsigned long priority,
    FmcjReadWriteContainerHandle input,
    FmcjReadWriteContainerHandle output,
    FmcjCorrelID * correlID,
    char const * userContext )
    
```

C++

```

APIRET Execute( FmcjReadWriteContainer const * input = 0,
                FmcjReadWriteContainer * output = 0,
                unsigned long priority = 0 ) const

APIRET ExecuteAsync( FmcjReadWriteContainer const * input = 0,
                    FmcjReadWriteContainer const * output = 0,
                    FmcjCorrelID * correlID = 0,
                    string const * userContext = 0,
                    unsigned long priority = 0 ) const
    
```

Java

```

public abstract
ReadWriteContainer execute() throws FmcException

public abstract
ReadWriteContainer execute2( ReadWriteContainer input,
                            ReadWriteContainer output,
                            long priority )
throws FmcException
    
```

ProgramTemplate

COBOL

```
FmcjPgTExecute.  
  
    CALL    "FmcjProgramTemplateExecute"  
           USING  
           BY VALUE  
             hdTemplate  
             inputValue  
             outputValue  
           BY REFERENCE  
             returnCode  
           RETURNING  
             intReturnValue.  
  
FmcjPgTExecuteWithOptions.  
  
    CALL    "FmcjProgramTemplateExecuteWithOptions"  
           USING  
           BY VALUE  
             hdTemplate  
             priority  
             inputValue  
             outputValue  
           BY REFERENCE  
             returnCode  
           RETURNING  
             intReturnValue.  
  
FmcjPgTExecuteAsync.  
  
    CALL    "FmcjProgramTemplateExecuteAsync"  
           USING  
           BY VALUE  
             hdTemplate  
             inputValue  
             outputValue  
           BY REFERENCE  
             correID  
           BY VALUE  
             userContext  
           RETURNING  
             intReturnValue.  
  
FmcjPgTExecuteWithOptionsAsync.  
  
    CALL    "FmcjProgramTemplateExecuteWithOptionsAsync"  
           USING  
           BY VALUE  
             hdTemplate  
             priority  
             inputValue  
             outputValue  
           BY REFERENCE  
             correID  
           BY VALUE  
             userContext  
           RETURNING  
             intReturnValue.
```

Parameters

| | |
|--------------------|--|
| <i>correlID</i> | Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response. |
| <i>hdlTemplate</i> | Input. The handle of the program template object to be executed. |
| <i>input</i> | Input. The input container of the program. |
| <i>output</i> | Input/Output. The output container of the program. |
| <i>priority</i> | Input. The priority of the program to be executed. |
| <i>userContext</i> | Input. A user-defined context which can be used for correlation. |

Return type

long/APIRET The return code of calling this API call - see return codes below.

ReadWriteContainer

The output container of the program.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_IMPLEMENTATION_SUPPORT_MISMATCH(32015)

The program definition for the operating system platform the PES is running on is not found.

FMC_ERROR_INVALID_CONTAINER(509)

The type of the container is incorrect or a container is expected but not passed.

FMC_ERROR_INVALID_CORRELATION_ID(506)

The specified correlation ID does not point to FMCJ_NO_CORRELID.

FMC_ERROR_INVALID_USER_CONTEXT(819)

The specified user context is longer than 254 characters.

FMC_ERROR_SUPPORT_MODE_MISMATCH(32014)

The execution mode of the program and the execution mode of the PES do not match.

FMC_ERROR_UNEXPECTED_CONTAINER(510)

A container is passed but not expected by the program.

FMC_ERROR_USER_SUPPORT_MISMATCH(32013)

The execution user of the program and the execution user of the PES do not match.

FMC_ERROR_COMMUNICATION(13)

The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)

The document is not a valid XML document.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)

The document is not a valid MQSeries Workflow XML document.

ProgramTemplate

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)

The type of the container is incorrect.

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)

The specified data member is not part of the container.

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)

The type of the data member value passed is incorrect.

Service actions

A Service object represents the common aspects of MQSeries Workflow service objects.

In C++, FmcjService is the superclass of the FmcjExecutionService class and provides for all common properties and methods. In Java, Service is thus a superclass of the ExecutionService class and provides for all common properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjService. That is, common functions start with the prefix FmcjService; they are also defined starting with the prefix FmcjExecutionService.

The following sections describe the actions which can be applied on a service. See "Service" on page 265 for a complete list of API calls.

Refresh()

This API call refreshes the logon status from the server (action call).

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Logon required

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Service |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_FMC_APIENTRY
    FmcjServiceRefresh( FmcjServiceHandle service )

#define FmcjExecutionServiceRefresh FmcjServiceRefresh
```

```
C++
APIRET Refresh()
```

Java

```
public abstract
void refresh() throws FmcException
```

COBOL

```
FmcjSrvRefresh.
CALL    "FmcjServiceRefresh"
        USING
        BY VALUE
        serviceValue
        RETURNING
        intReturnValue.
```

Parameters

service Input. A handle to the service object representing the session with an MQSeries Workflow server.

Return type

APIRET/long The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

SetPassword()

This API call allows a user's password to be changed (action call).

Note: The password is case-sensitive.

The following rules apply for specifying a password:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale.
- Do not use DBCS characters.

Service

Note: If you intend to work in a multi-platform environment or switch between code pages, it is recommended that you use alphabetic characters, digits, and blanks only. This is because it cannot be guaranteed that special characters are available in all code pages.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Logon required

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.Service |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_FMC_APIENTRY
    FmcjServiceSetPassword( FmcjServiceHandle service,
                           char const *      newPassword )

#define FmcjExecutionServiceSetPassword FmcjServiceSetPassword
```

```
C++
APIRET SetPassword( string const & newPassword ) const
```

```
Java
public abstract
void setPassword( String newPassword ) throws FmcException
```

```
COBOL
FmcjSrvSetPassword.
    CALL "FmcjServiceSetPassword"
        USING
            BY VALUE
                serviceValue
                newPassword
        RETURNING
            intReturnValue.
```


Parameters

newPassword Input. The new password to be used.
service Input. A handle to the service object representing the session with an MQSeries Workflow server.

Return type

long/ APIRET The return code from this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_USERID_UNKNOWN(10)

The user no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_PASSWORD(12)

The password does not comply with the MQSeries Workflow syntax rules.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

UserSettings()

This API call returns all settings of the logged on user (action call).

An empty object or a null pointer is returned if no user has logged on yet via this service object.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Logon required

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjrun.h

C++ fmcjprun.hxx

Service

Java com.ibm.workflow.api.Service

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_FMC_APIENTRY
        FmcjServiceUserSettings( FmcjServiceHandle service,
                                FmcjPersonHandle * user )

#define FmcjExecutionServiceUserSettings FmcjServiceUserSettings
```

C++

```
APIRET UserSettings( FmcjPerson & user ) const
```

Java

```
public abstract
Person userSettings() throws FmcException
```

COBOL

```
FmcjSrvUserSettings.

        CALL      "FmcjServiceUserSettings"
                USING
                BY VALUE
                serviceValue
                BY REFERENCE
                user
                RETURNING
                intReturnValue.
```

Parameters

returnCode Input/Output. The return code of calling this method - see return codes below.

service Input. A handle to the service object representing the session with an MQSeries Workflow server.

user Input/Output. The person object to contain or the address of the person handle to point to the settings of the logged on user.

Return type

APIRET The return code from this API call - see return codes below.

IDispatch*/ Person

A pointer to the person settings or the person settings of the logged on user.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Workitem actions

A Workitem object represents an activity instance assigned to a user in order to be worked on.

Other items assigned to users are process instance notifications and activity instance notifications. FmcjItem or Item represents the common properties of all items.

In C++, FmcjWorkitem is thus a subclass of the FmcjItem class and inherits all properties and methods. In Java, WorkItem is thus a subclass of the Item class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjItem. That is, common functions start with the prefix FmcjItem; they are also defined starting with the prefix FmcjWorkitem.

A work item is uniquely identified by its object identifier.

The following diagrams provide an overview of the possible work item states and the actions which are allowed in those states, provided that the appropriate authority has been granted. Note that the actions and possible states are dependent on the process instance state, the work item is a part of.

Workitem

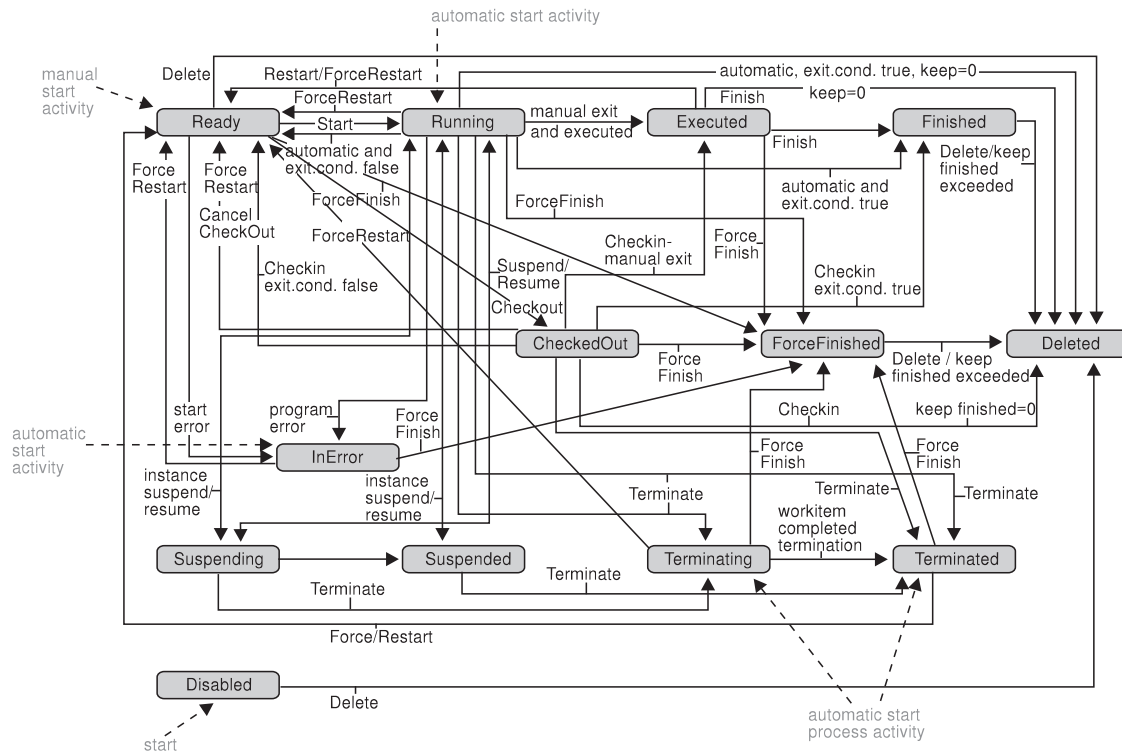


Figure 44. Work item states - process instance state running

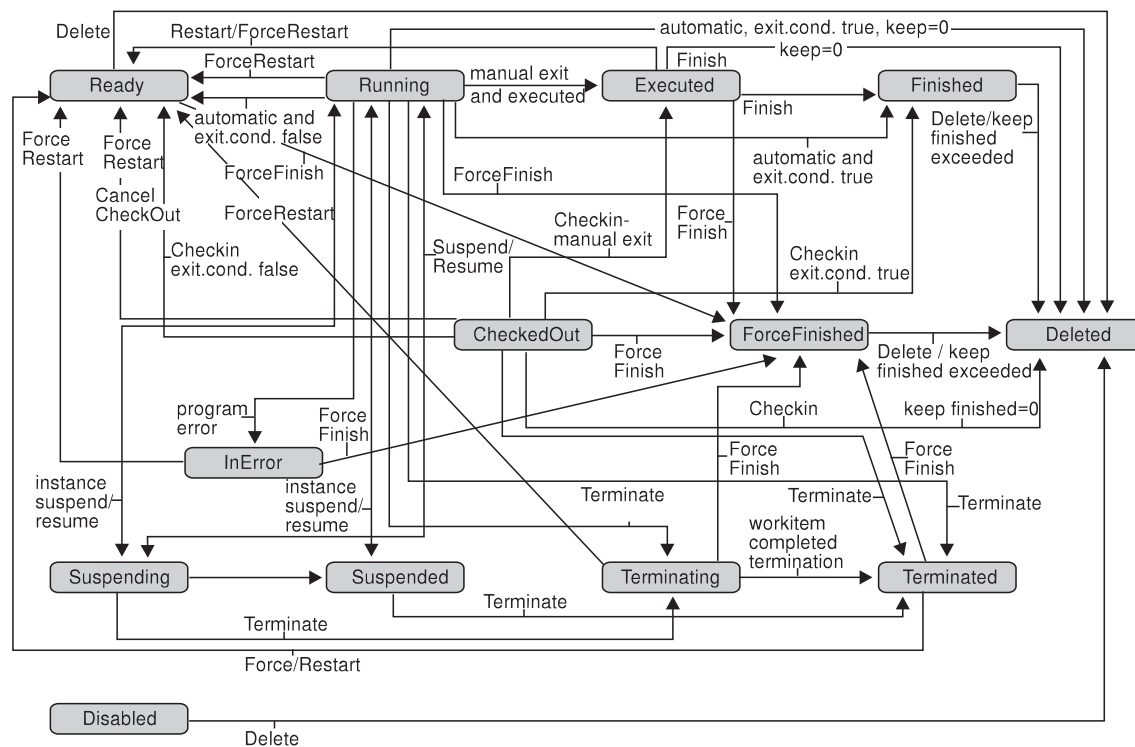


Figure 45. Work item states - process instance state suspending or suspended

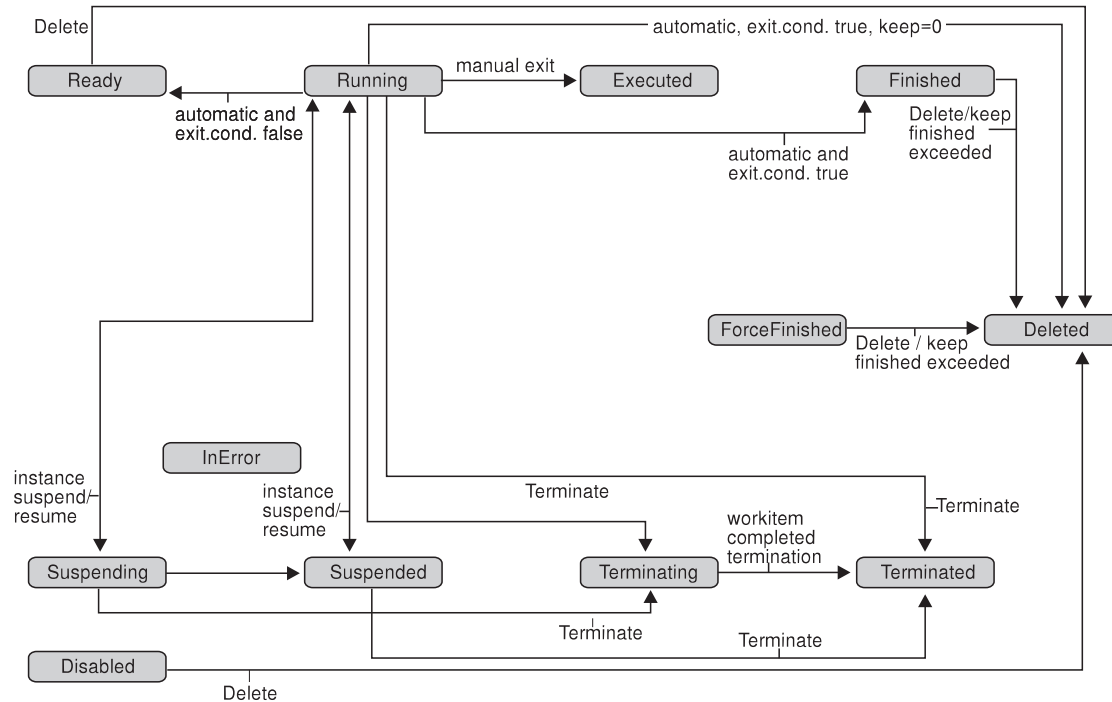


Figure 46. Work item states - process instance state terminating or terminated

The following sections describe the actions which can be applied on a work item. See “WorkItem” on page 268 for a complete list of API calls.

CancelCheckout()

This API call cancels the checkout of the work item (action call).

The work item must have been checked out and is put into the *Ready* state. The associated process instance must be in the *Running*, *Suspending*, *Suspended*, or *Terminating* state.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

Workitem

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemCancelCheckOut( FmcjWorkitemHandle hdlWorkitem )
```

C++

```
APIRET CancelCheckOut()
```

Java

```
public abstract  
void cancelCheckOut() throws FmcException
```

COBOL

```
FmcjWICancelCheckOut.  
  
CALL "FmcjWorkitemCancelCheckOut"  
USING  
BY VALUE  
hdlWorkitem  
RETURNING  
intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item or process instance is not in a required state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

CheckIn()

This API call allows for the check in of a work item that was previously checked out for user processing (action call).

Checking in a work item tells MQSeries Workflow that user processing has finished and workflow processing under the control of MQSeries Workflow can continue. The return code of the user processing and, optionally, the output container values are passed back to MQSeries Workflow. As usual, these container values and the return code can be used in exit conditions to let navigation continue depending on the success of the processing and in transition conditions to indicate how to proceed.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjWorkitemCheckIn( FmcjWorkitemHandle      hdlWorkitem,
                    FmcjReadWriteContainerHandle output,
                    long                       returnCode )
```

C++

```
APIRET CheckIn( FmcjReadWriteContainer const * output,
                long                       returnCode )
```

Workitem

Java

```
public abstract
void checkIn( ReadWriteContainer output,
              int                returnCode ) throws FmcException
```

COBOL

```
FmcjWICheckIn.

CALL      "FmcjWorkitemCheckIn"
          USING
          BY VALUE
          hdlWorkitem
          outputValue
          returnCode
          RETURNING
          intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
output Input. A handle or pointer to the output container; can be a NULL pointer.
returnCode Input. The return code of user processing.

Return type

long/ APIRET

The return code from this API call- see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item is not checked out.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

CheckOut()

This API call checks out a ready work item for user processing (action call).

The work item must be processed by a program.

Checkout then means that processing is not done by MQSeries Workflow's inherent program-invocation mechanism. MQSeries Workflow assumes that processing is done by user-specific means and changes the state of the work item to *CheckedOut*.

The associated process instance must be in the *Running* state.

The caller can request program definitions for specific operating system platforms. The following enumeration types can be used to specify the requested program data.

| | |
|-------------|---|
| C | FmcjWorkitemProgramRetrieval |
| C++ | FmcjWorkitem::ProgramRetrieval |
| Java | com.ibm.workflow.api.WorkItemPackage.ProgramRetrieval |

The enumeration constants can take the following values; it is strongly advised to use the symbolic names instead of the associated integer values.

NotSet indicates that no value is set.

| | |
|--------------|--------------------------|
| C | Fmc_DP_NotSet |
| C++ | FmcjWorkitem::NotSet |
| Java | ProgramRetrieval.NOT_SET |
| COBOL | Fmc-DP-NotSet |

CommonDataOnly

returns only data common to all platforms, the description, the icon, the unattended indicator, and the input and output containers. Any platform specification is ignored.

| | |
|--------------|-----------------------------------|
| C | Fmc_WS_CommonDataOnly |
| C++ | FmcjWorkitem::CommonDataOnly |
| Java | ProgramRetrieval.COMMON_DATA_ONLY |
| COBOL | Fmc-WS-CommonDataOnly |

SpecifiedDefinitions

returns the program definition for the specified platform. A platform must be specified.

| | |
|--------------|--|
| C | Fmc_WS_SpecifiedDefinitions |
| C++ | FmcjWorkitem::SpecifiedDefinitions |
| Java | ProgramRetrieval.SPECIFIED_DEFINITIONS |
| COBOL | Fmc-WS-SpecifiedDefs |

Workitem

AllDefinitions

returns all available program definitions. Any platform specification is ignored.

| | |
|--------------|----------------------------------|
| C | Fmc_WS_AllDefinitions |
| C++ | FmcjWorkitem::AllDefinitions |
| Java | ProgramRetrieval.ALL_DEFINITIONS |
| COBOL | Fmc-WS-AllDefs |

The following enumeration types can be used to specify the platform for which program definitions are to be retrieved.

| | |
|-------------|---|
| C | FmcjImplementationDataBasis |
| C++ | FmcjImplementationData::Basis |
| Java | com.ibm.workflow.api.ProgramDataPackage.Basis |

The enumeration constants can take the following values; it is strongly advised to use the symbolic names instead of the associated integer values.

| | |
|------------------|---|
| NotSet | indicates that no value is set. |
| C | Fmc_DP_NotSet |
| C++ | FmcjImplementationData::NotSpecified |
| Java | Basis.NOT_SPECIFIED |
| COBOL | Fmc-DP-NotSet |
| OS2 | indicates that the program definition for the OS/2 platform is requested. |
| C | Fmc_DP_OS2 |
| C++ | FmcjImplementationData::OS2 |
| Java | Basis.OS2 |
| COBOL | Fmc-DP-OS2 |
| AIX | indicates that the program definition for the AIX platform is requested. |
| C | Fmc_DP_AIX |
| C++ | FmcjImplementationData::AIX |
| Java | Basis.AIX |
| COBOL | Fmc-DP-AIX |
| HPUX | indicates that the program definition for the HP-UX platform is requested. |
| C | Fmc_DP_HPUX |
| C++ | FmcjImplementationData::HPUX |
| Java | Basis.HPUX |
| COBOL | Fmc-DP-HPUX |
| Windows95 | indicates that the program definition for the Windows 95 platform is requested. |
| C | Fmc_DP_Windows95 |
| C++ | FmcjImplementationData::Windows95 |

| | | |
|------------------|--------------|---|
| | Java | Basis.WINDOWS_95 |
| WindowsNT | COBOL | Fmc-DP-Windows95 |
| | | indicates that the program definition for the Windows NT platform is requested. |
| | C | Fmc_DP_WindowsNT |
| | C++ | FmcjImplementationData::WindowsNT |
| OS390 | Java | Basis.WINDOWS_NT |
| | COBOL | Fmc-DP-WindowsNT |
| | | indicates that the program definition for the OS/390 platform is requested. |
| | C | Fmc_DP_OS390 |
| | C++ | FmcjImplementationData::OS390 |
| Solaris | Java | Basis.WINDOWS_OS390 |
| | COBOL | Fmc-DP-OS390 |
| | | indicates that the program definition for the Solaris platform is requested. |
| | C | Fmc_DP_Solaris |
| | C++ | FmcjImplementationData::Solaris |
| | Java | Basis.Solaris |
| | COBOL | Fmc-DP-Solaris |

For Java programs, checkOut2() additionally allows for specifying which program definitions to retrieve.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

Workitem

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemCheckOut( FmcjWorkitemHandle         hdlWorkitem,  
                      enum FmcjWorkitemProgramRetrieval requestedData,  
                      enum FmcjImplementationDataBasis platform,  
                      FmcjProgramDataHandle *      programData )
```

C++

```
APIRET CheckOut( ProgramRetrieval          requestedData,  
                 FmcjImplementationData::Basis platform,  
                 FmcjProgramData &       programData )
```

Java

```
public abstract  
ReadOnlyContainer checkOut() throws FmcException  
  
public abstract  
ProgramData      checkOut2(  
    ProgramRetrieval requestedData,  
    Basis            platform    ) throws FmcException
```

COBOL

```
FmcjWICheckOut.  
  
CALL "FmcjWorkitemCheckOut"  
    USING  
    BY VALUE  
        hdlWorkitem  
        requestedData  
        platform  
    BY REFERENCE  
        programData  
    RETURNING  
        intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
platform Input. The platform for which the program definition is to be returned.
programData Input/Output. The address of a handle to the program definition or the program definition object to be set.
requestedData Input. An indicator which program definitions are to be returned.
returnCode Input/Output. The return code of calling this method - see below.

Return type

APIRET The return code of calling this method - see below.

ProgramData The program definition.

ReadOnlyContainer

The input container of the work item; the container is part of the program definition. Returned for Version 2 compatibility reasons.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_CHECKOUT_NOT_POSSIBLE(503)

The work item cannot be checked out.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Finish()

This API call ends the execution of a manual-exit work item (action call).

The work item must be in state *Executed*, that is, must have run at least once. The work item is then put into the *Finished* state. Depending on the “delete finished items” option, it is deleted.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.WorkItem

Workitem

COBOL fmcvars.cpy, fmcperf.cpy

```
C
APIRET FMC_APIENTRY FmcjWorkitemFinish( FmcjWorkitemHandle hdlWorkitem )
```

```
C++
APIRET Finish()
```

```
Java
public abstract
void finish() throws FmcException
```

```
COBOL

      FmcjWIFinish.
      CALL    "FmcjWorkitemFinish"
              USING
              BY VALUE
              hdlWorkitem
              RETURNING
              intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ForceFinish()

This API call ends the execution of a work item which is known to have completed in cases where MQSeries Workflow did not recognize this event (action call).

This situation can occur when the execution server aborted before it received the activity implementation completion message.

A work item implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in the states *Ready*, *Executed*, *InError*, or *Terminated*. The associated process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

The work item is then put into the *ForceFinished* state. The exit condition is considered to be true and navigation proceeds.

Depending on the "delete finished items" option, the work item is deleted.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the work item owner and one of

- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjWorkitemForceFinish( FmcjWorkitemHandle hdlWorkitem )
```

Workitem

```
C++  
APIRET ForceFinish()
```

```
Java  
public abstract  
void forceFinish() throws FmcException
```

```
COBOL  
  
FmcjWIForceFinish.  
  
CALL "FmcjWorkitemForceFinish"  
USING  
BY VALUE  
hdlWorkitem  
RETURNING  
intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

ForceRestart()

This API call forces MQSeries Workflow to enable the restart of a work item (action call).

A work item implemented by a program must be in states *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in states *Executed*, *InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

It is then reset into the *Ready* state. Note that automatic activity instances must now be started manually.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the work item owner and one of

- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

```
C
APIRET FMC_APIENTRY FmcjWorkitemForceRestart(
    FmcjWorkitemHandle hdlWorkitem )
```

```
C++
APIRET ForceRestart()
```

```
Java
public abstract
void forceRestart() throws FmcException
```

Workitem

COBOL

```
FmcjWIForceRestart.  
  
    CALL    "FmcjWorkitemForceRestart"  
           USING  
           BY VALUE  
           hdlWorkitem  
           RETURNING  
           intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the work item from the MQSeries Workflow execution server (action call).

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the work item owner
- Work item authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjWorkitemInContainer( FmcjWorkitemHandle         hdlWorkitem,
                        FmcjReadOnlyContainerHandle * input )
```

C++

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

COBOL

```
FmcjWIIInCtnr.

CALL "FmcjWorkitemInContainer"
    USING
    BY VALUE
        hdlWorkitem
    BY REFERENCE
        inputValue
    RETURNING
        intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
input Input/Output. The input container.

Return type

long/ APIRET The return code of calling this method - see below.

ReadOnlyContainer

The input container.

Workitem

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the work item from the MQSeries Workflow execution server (action call).

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the work item owner
- Work item authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY
FmcjWorkitemOutContainer( FmcjWorkitemHandle          hdlWorkitem,
                          FmcjReadWriteContainerHandle * output )
```

C++

```
APIRET OutContainer( FmcjReadWriteContainer & output ) const
```

Java

```
public abstract
ReadWriteContainer outContainer() throws FmcException
```

COBOL

```
FmcjWIOutCtnr.

CALL      "FmcjWorkitemOutContainer"
          USING
          BY VALUE
            hdlWorkitem
          BY REFERENCE
            outputValue
          RETURNING
            intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
output Input/Output. The output container.

Return type

long/ APIRET The return code of calling this method - see below.

ReadWriteContainer

The output container.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

Workitem

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

PersistentObject()

This API call retrieves the work item identified by the passed object identifier from the MQSeries Workflow execution server (action call).

The MQSeries Workflow execution server from which the work item is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the work item.

In C++, when the work item object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the Work item handle already points to some object. In Java, a work item is newly created; the execution service acts as a factory.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

One of:

- Be the work item owner
- Work item authorization
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.ExecutionService

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY
FmcjWorkitemPersistentObject( FmcjExecutionServiceHandle service,
                               char const *                oid,
                               FmcjWorkitemHandle *       hdlWorkitem )
```

C++

```
APIRET PersistentObject( FmcjExecutionService const & service,
                          string const &                oid )
```

Java

```
public abstract
  WorkItem ExecutionService.persistentWorkItem( String oid )
throws FmcException
```

COBOL

```
FmcjWIPersistentObj.

      CALL    "FmcjWorkitemPersistentObject"
            USING
            BY VALUE
              serviceValue
              oid
            BY REFERENCE
              hdlWorkitem
            RETURNING
              intReturnValue.
```

Parameters

hdlWorkitem Input/Output. The address of the handle to the work item object to be set.

oid Input. The object identifier of the work item to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The return code of calling this method - see below.

WorkItem The work item retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

Workitem

FMC_ERROR_INVALID_OID(805)

The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Restart()

This API call asks MQSeries Workflow to enable the restart of a work item (action call).

The work item must be in state *Executed*. It is then reset into the *Ready* state.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h

C++ fmcjprun.hxx

Java com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemRestart( FmcjWorkitemHandle hdlWorkitem )
```

C++

```
APIRET Restart()
```


Java

```
public abstract
void restart() throws FmcException
```

COBOL

```
FmcjWIRestart.

CALL    "FmcjWorkitemRestart"
        USING
        BY VALUE
        hdlWorkitem
        RETURNING
        intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Workitem

Start()

This API call starts a ready work item (action call).

The associated process instance must be in the *Running* state.

If the associated activity instance is implemented by a program, the program is started on the program execution server associated with the program.

The work item is put into the *Running* state. If the activity implementation or an associated process activity cannot be started, the work item is put into the *InError* state.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemStart( FmcjWorkitemHandle hdlWorkitem )
```

C++

```
APIRET Start()
```

Java

```
public abstract  
void start() throws FmcException
```

COBOL

```

FmcjWIStart.

        CALL      "FmcjWorkitemStart"
                USING
                BY VALUE
                hdlWorkitem
                RETURNING
                intReturnValue.

```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

StartTool()

This API call starts the specified support tool (action call).

The support tool must be one of the tools associated with the activity instance the work item is derived from. It is then started via the program execution agent associated with the logged-on user.

Workitem

Note: A support tool can be started only via a program execution agent in the LAN environment; starting via a program execution server (in either environment) is currently not supported. Since there are only unattended processes under MQSeries Workflow for OS/390, it is not meaningful to start a support tool in this environment. The PES will simply ignore such an attempt.

Usage notes

- See "Action API calls" on page 121 for general information.

Authorization

Be the work item owner

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkItem |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemStartTool( FmcjWorkitemHandle hdlWorkitem,  
                       char const *      toolName )
```

C++

```
APIRET StartTool( string const & toolName ) const
```

Java

```
public abstract  
void startTool( String toolName ) throws FmcException
```

COBOL

```
FmcjWISStartTool.  
  
CALL "FmcjWorkitemStartTool"  
  USING  
  BY VALUE  
  hdlWorkitem  
  toolName  
  RETURNING  
  intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
toolName Input. The support tool to be started.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_INVALID_TOOL(129)

No tool name is provided or the specified tool is not defined for the work item.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Terminate()

This API call terminates a work item implemented by a program or process (action call).

If the work item is implemented by a program, it must be in the states *CheckedOut* or *Running* and the process instance must be in the states *Running*, *Suspending*, or *Suspended*. If the work item is implemented by a process, it must be in the states *Running*, *Suspending*, or *Suspended* and the process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

A work item implemented by a process is terminated together with all its non-autonomous subprocesses with respect to control autonomy.

The work item is then put into the *Terminating* or *Terminated* state.

Depending on the "delete finished items" option, the work item is deleted.

Workitem

When the *Terminated* state has been reached, the exit condition is considered to be false, the output container and especially the return code (`_RC`) are not set, and navigation ends. If not yet deleted, navigation can be explicitly continued by a user with process administration rights, that is, `ForceFinish()` or `ForceRestart()` repair actions can be called.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

Be the work item owner

For work items implemented by a process, additionally one of:

- Process administration authority
- Be the process administrator
- Be the system administrator

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|--|
| C | <code>fmcjcrun.h</code> |
| C++ | <code>fmcjprun.hxx</code> |
| Java | <code>com.ibm.workflow.api.WorkItem</code> |
| COBOL | <code>fmcvars.cpy, fmcperf.cpy</code> |

C

```
APIRET FMC_APIENTRY  
FmcjWorkitemTerminate( FmcjWorkitemHandle hdlWorkitem )
```

C++

```
APIRET Terminate()
```

Java

```
public abstract  
void terminate() throws FmcException
```

COBOL

```

FmcjWITerminate.

      CALL      "FmcjWorkitemTerminate"
              USING
              BY VALUE
              hdlWorkitem
              RETURNING
              intReturnValue.

```

Parameters

hdlWorkitem Input. The handle of the work item to be terminated.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The work item no longer exists.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_WRONG_STATE(120)

The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Worklist actions

A Worklist object represents a set of items, that is, a set of work items or notifications. All items which are accessible through this list have the same characteristics. These characteristics are specified by a filter. Additionally, sort criteria can be applied and, after that, a threshold to restrict the number of items to be transferred from the execution server to the client.

Worklist

The worklist definition is stored persistently. The items contained in the worklist are, however, assembled dynamically when they are queried.

A worklist is uniquely identified by its name, type, and owner. It can be defined for general access purposes; it is then of a *public* type. Or, it can be defined for some specific user; it is then of a *private* type.

Other lists that can be defined are process template lists or process instance lists. FmcjPersistentList or PersistentList represents the common properties of all lists.

In C++, FmcjWorklist is thus a subclass of the FmcjPersistentList class and inherits all properties and methods. In the Java language, WorkList is thus a subclass of the PersistentList class and inherits all properties and methods. Similarly, in C or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjWorklist.

The following sections describe the actions which can be applied on a worklist. See “Worklist” on page 270 for a complete list of API calls.

QueryActivityInstanceNotifications()

This API call retrieves the primary information for all activity instance notifications characterized by the specified worklist from the MQSeries Workflow execution server (action call).

From the set of qualifying activity instance notifications, only those are retrieved, the user is authorized for. The user is authorized for an activity instance notification if

- He is the owner of the activity instance notification
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each activity instance notification is:

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

In C, C++, and COBOL, any activity instance notifications retrieved are appended to the supplied vector of activity instance notifications. If you want to read those activity instance notifications only which are currently included in the worklist,

you have to clear the vector before you issue this API call. This means that you should set the handle to 0 in C or COBOL, or erase all elements of the vector in C++.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY FmcjWorklistQueryActivityInstanceNotifications(
    FmcjWorklistHandle          hdList,
    FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++

```
APIRET QueryActivityInstanceNotifications(
    vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications()
throws FmcException
```

COBOL

```
FmcjWLQueryActInstNotifs.

CALL "FmcjWorklistQueryActivityInstanceNotifications"
    USING
    BY VALUE
        hdList
    BY REFERENCE
        notifications
    RETURNING
        intReturnValue.
```

Worklist

Parameters

hdlList Input. The handle of the worklist to be queried.
notifications Input/Output. The vector of qualifying activity instance notifications.

Return type

long/ APIRET The return code of calling this method - see below.

ActivityInstanceNotification[]

The qualifying activity instance notifications.

Return codes/ **FmcException**

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The worklist no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see "Query work items from a worklist (C)" on page 546
- For a C++ example see "Query work items from a worklist (C++)" on page 548
- For a Java example see "Query work items from a worklist (Java)" on page 550

QueryItems()

This API call retrieves the primary information for all items characterized by the specified worklist from the MQSeries Workflow execution server (action call).

From the set of qualifying items, only those are retrieved, the user is authorized for. The user is authorized for an item if

- He is the owner of the item
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each item is:

- Category
- CreationTime
- Description

- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

If the item is an actual work item or an activity instance notification, then additional primary information is retrieved:

- ActivityType
- Implementation
- Priority
- SupportTools

In C, C++, and COBOL, any items retrieved are appended to the supplied vector of items. If you want to read those items only which are currently included in the worklist, you have to clear the vector before you issue this API call. This means that you should set the handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|--------------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

C

```
APIRET FMC_APIENTRY
FmcjWorklistQueryItems( FmcjWorklistHandle    hdList,
                        FmcjItemVectorHandle * items )
```

C++

```
APIRET QueryItems( vector<FmcjItem> & items ) const
```

Worklist

Java

```
public abstract Item[] queryItems() throws FmcException
```

COBOL

```
FmcjWLQueryItems.  
  
CALL "FmcjWorklistQueryItems"  
  USING  
    BY VALUE  
      hdlList  
  BY REFERENCE  
    items  
  RETURNING  
    intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.
items Input/Output. The vector of qualifying items.

Return type

APIRET The return code of calling this method - see below.
Item[] The qualifying items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The worklist no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see "Query work items from a worklist (C)" on page 546
- For a C++ example see "Query work items from a worklist (C++)" on page 548

- For a Java example see “Query work items from a worklist (Java)” on page 550

QueryProcessInstanceNotifications()

This API call retrieves the primary information for all process instance notifications characterized by the specified worklist from the MQSeries Workflow execution server (action call).

From the set of qualifying process instance notifications, only those are retrieved, the user is authorized for. The user is authorized for a process instance notification if

- He is the owner of the process instance notification
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each process instance notification is:

- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

In C, C++, and COBOL, any process instance notifications retrieved are appended to the supplied vector of process instance notifications. If you want to read those process instance notifications only which are currently included in the worklist, you have to clear the vector before you issue this API call. This means that you should set the handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

| | |
|-------|-------------------------------|
| C | fmcjcrun.h |
| C++ | fmcjprun.hxx |
| Java | com.ibm.workflow.api.WorkList |
| COBOL | fmcvars.cpy, fmcperf.cpy |

Worklist

C

```
APIRET FMC_APIENTRY FmcjWorklistQueryProcessInstanceNotifications(  
    FmcjWorklistHandle          hdlList,  
    FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++

```
APIRET QueryProcessInstanceNotifications(  
    vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java

```
public abstract  
ProcessInstanceNotification[] queryProcessInstanceNotifications()  
throws FmcException
```

COBOL

```
FmcjWLQueryProcInstNotifs.  
  
    CALL      "FmcjWorklistQueryProcessInstanceNotifications"  
            USING  
            BY VALUE  
            hdlList  
            BY REFERENCE  
            notifications  
            RETURNING  
            intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.
notifications Input/Output. The vector of qualifying process instance notifications.

Return type

long/ APIRET The return code of calling this method - see below.

ProcessInstanceNotification[]

The qualifying process instance notifications.

Return codes/ **FmcException**

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The worklist no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see "Query work items from a worklist (C)" on page 546
- For a C++ example see "Query work items from a worklist (C++)" on page 548
- For a Java example see "Query work items from a worklist (Java)" on page 550

QueryWorkitems()

This API call retrieves the primary information for all work items characterized by the specified worklist from the MQSeries Workflow execution server (action call).

From the set of qualifying work items, only those are retrieved, the user is authorized for. The user is authorized for a work item if

- He is the owner of the work item
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each work item is:

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

In C, C++, and COBOL, any work items retrieved are appended to the supplied vector of work items. If you want to read those work items only which are currently included in the worklist, you have to clear the vector before you issue this API call. This means that you should set the handle to 0 in C or COBOL, or erase all elements of the vector in the C++ API.

Worklist

Usage notes

- See “Action API calls” on page 121 for general information.

Authorization

None

Required connection

MQSeries Workflow execution server

API interface declarations

C fmcjcrun.h
C++ fmcjprun.hxx
Java com.ibm.workflow.api.WorkList
COBOL fmcvars.cpy, fmcperf.cpy

```
C
APIRET FMC_APIENTRY FmcjWorklistQueryWorkitems(
    FmcjWorklistHandle    hdlList,
    FmcjWorkitemVectorHandle * workitems )
```

```
C++
APIRET QueryWorkitems( vector<FmcjWorkitem> & workitems ) const
```

```
Java
public abstract
WorkItem[] queryWorkItems() throws FmcException
```

```
COBOL
FmcjWLQueryWorkitems.
CALL "FmcjWorklistQueryWorkitems"
    USING
    BY VALUE
    hdlList
    BY REFERENCE
    workitems
    RETURNING
    intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.
workitems Input/Output. The vector of qualifying work items.

Return type

long/ APIRET The return code of calling this method - see below.

WorkItem[] The qualifying work items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The worklist no longer exists.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQSeries Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Examples

- For a C example see “Query work items from a worklist (C)” on page 546
- For a C++ example see “Query work items from a worklist (C++)” on page 548
- For a Java example see “Query work items from a worklist (Java)” on page 550

Worklist

Chapter 6. Examples

The following samples are provided in InstHLQ.SFMCSRC:

FMCHSCFA C Full API sample (native OS/390)
FMCHSCCA C Container API sample (CICS and IMS)
FMCHSCBF COBOL Full API sample (native OS/390)
FMCHSCBC COBOL Container API sample (IMS)
FMCHSCBN COBOL Container API sample (CICS)

In addition, the following sections illustrate examples for:

- Creating persistent lists, such as process instances
- Querying persistent lists, such as process instances
- Querying a set of objects, such as process instances and work items
- Programming an activity implementation (executable)

How to create persistent lists

The following examples show how to create a persistent list, that is, a persistent view of a set of objects. They define a view of process instances. Other possible lists to define are process template lists or worklists.

Examples

Create a process instance list (C)

```
#include <stdio.h>
#include <fmcjcrun.h>          /* MQ Workflow Runtime API */
int main()
{
    APIRET rc = FMC_OK;
    FmcjExecutionServiceHandle service = 0;
    FmcjProcessInstanceListHandle instanceList = 0;
    unsigned long threshold = 10;
    int enumValue = 0;
    char name[50] = "MyTenInstances";
    char desc[50] = "This list contains no more than 10 instances";

    FmcjGlobalConnect();
    /* logon */
    rc= FmcjExecutionServiceAllocate( &service );
    if (rc != FMC_OK)
    {
        printf("Service object could not be allocated - rc: %u\n",rc);
        return -1;
    }
    rc= FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_NotSet
                                   );
    if (rc != FMC_OK)
    {
        printf("Logon failed - rc: %u\n",rc);
        FmcjExecutionServiceDeallocate( &service );
        return -1;
    }
    /* create a process instance list */
    rc = FmcjExecutionServiceCreateProcessInstanceList(
        service,
        name,
        Fmc_LT_Private,
        "USERID",
        desc,
        FmcjNoFilter,
        FmcjNoSortCriteria,
        &threshold,
        &instanceList );

    if ( rc != FMC_OK)
        printf( "CreateProcessInstanceList returns: %u\n",rc );
    else
        printf( "CreateProcessInstanceList okay\n" );

    FmcjExecutionServiceLogoff( service );
    FmcjExecutionServiceDeallocate( &service );
    FmcjGlobalDisconnect();
    return 0;
}
```

Figure 47. Sample C program to create a process instance list

Create a process instance list (C++)

```

#include <iomanip.h>
#include <bool.h> // bool
#include <fmcjstr.hxx> // string
#include <vector.h> // vector
#include <fmcjprun.hxx> // MQ Workflow Runtime API
int main()
{
    FmcjGlobal::Connect();

    // logon
    FmcjExecutionService service;
    APIRET rc = service.Logon("USERID", "password");
    if ( rc != FMC_OK )
    {
        cout << "Logon failed, - rc: " << rc << endl;
        return -1;
    }

    // create a process instance list

    FmcjProcessInstanceList instanceList;
    string name ("MyTenInstances");
    string desc ("List contains no more than 10 instances");
    string owner ("USERID");
    unsigned long threshold= 10;

    rc = service.CreateProcessInstanceList(
        name,
        FmcjPersistentList::Private,
        &owner,
        &desc,
        FmcjNoFilter,
        FmcjNoSortCriteria,
        &threshold,
        instanceList );

    if ( rc != FMC_OK)
        cout << "CreateProcessInstanceList returns: " << rc << endl;
    else
        cout << "CreateProcessInstanceList okay" << endl;

    service.Logoff();

    FmcjGlobal::Disconnect();
    return 0;
}

```

Figure 48. Sample C++ program to create a process instance list

Examples

Create a process instance list (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class CreateProcInstList
{
    public static void main(String[] args)
    {
        // Check the arguments. The first argument is the name of the MQSeries
        // Workflow agent the client will connect to. The second argument defines
        // the locator policy the client will use when trying to contact the agent.
        // The third/fourth argument define the userid/password, which, if not
        // specified, default to USERID and password

        if ((args.length < 2 ) || (args.length > 4 ))
        {
            System.out.println("Usage:");
            System.out.println("java CreateProcessInstanceList
                               <agent> <LOC|RMI|OSA|IOR|COS>
                               [userid] [password]");
            System.exit(0);
        }
        try
        {
            // An agent bean representing an MQSeries Workflow domain
            String userid = "USERID";
            String passwd = "password";
            Agent agent = new Agent();
            // Parse the command line and set the locator to be used to
            // communicate with the agent.
            if (args[1].equalsIgnoreCase("LOC"))
            {
                agent.setLocator(Agent.LOC_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("RMI"))
            {
                agent.setLocator(Agent.RMI_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("OSA"))
            {
                agent.setLocator(Agent.OSA_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("IOR"))
            {
                agent.setLocator(Agent.IOR_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("COS"))
            {
                agent.setLocator(Agent.COS_LOCATOR);
            }
            else
            {
                System.out.println("Invalid locator policy: " + args[1]);
                System.exit(0);
            }

            if (args.length >=3 ) userid = args[2].toUpperCase();
            if (args.length >=4 ) passwd = args[3];
        }
    }
}
```

Figure 49. Sample Java program to create a process instance list (Part 1 of 3)

```

// Set the name of the Agent to be contacted. Setting the name
// automatically instructs the agent bean to contact the Agent using
// the current locator policy. For this reason the 'setLocator' must be
// called before 'setName' is invoked. If the agent bean cannot contact
// the Agent, it will raise a java.beans.PropertyVetoException instead
// of returning from the 'setName' call.
agent.setName(args[0]);

// Locate the default execution service in the system group named
// 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
// always returns successful (to prevent intrusion attempts which guess
// at service names until they find a valid one). Of course, only using
// a valid systemgroup and/or system name will return an ExecutionService
// which can be used to log on.
ExecutionService service = agent.locate("", "");

// Log on to the execution service. If the UserID and/or the password is
// invalid, an FmcException will be thrown.
service.logon(userid, passwd);
System.out.println("Logon successful");

String ListName      ="MyTenInstances";
String ListDesc      ="List contains no more than 10 instances";
String ListFilter    =" ";
String ListSort      =" ";
int    ListThreshold = 10;

try
{
    service.createProcessInstanceList( ListName, TypeOfList.PRIVATE,
                                       userid , ListDesc, ListFilter,
                                       ListSort, ListThreshold);
    System.out.println("Private ProcessInstanceList created successfully");
}
catch(FmcException e)
{
    if ( e.rc == FmcException.FMC_ERROR_NOT_UNIQUE )
    {
        System.out.println("ProcessInstanceList: '" + ListName +
                           "' already exists");
    }
}

finally
{
    // Logoff from the execution service. This (like any other remote call)
    // may raise an FmcException indicating a communication failure.
    service.logoff();

    System.out.println("Logoff successful");
}
}

```

Figure 49. Sample Java program to create a process instance list (Part 2 of 3)

Examples

```
catch(FmcException e)
{
    // Catch and report details about the FmcException
    System.out.println("FmcException occurred");
    System.out.println(" RC          : " + e.rc);
    System.out.println(" Origin       : " + e.origin);
    System.out.println(" MessageText: " + e.messageText);
    System.out.println(" Exception  : " + e.getMessage());
    System.out.println(" Parameters : ");
    for ( int i = 0; i < e.parameters.length ; i++)
    {
        System.out.println("    " + e.parameters[i] );
    }
    System.out.println(" StackTrace : ");
    e.printStackTrace();
}

catch(Exception e)
{
    // Catch and report any exception that occurred.
    e.printStackTrace();
}

    System.exit(0);
}
}
```

Figure 49. Sample Java program to create a process instance list (Part 3 of 3)

Create a process instance list (COBOL)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VECTOR".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcconst.
COPY fmcrs.

01 localUserID    PIC X(30) VALUE z"USERID".
01 localPassword  PIC X(30) VALUE z"PASSWORD".
01 listName       PIC X(50) VALUE z"MyTenInstances".
01 desc           PIC X(50)
                  VALUE z"This list contains no more than 10 instances".

LINKAGE SECTION.

01 retCode        PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.

* logon
    PERFORM FmcjESAllocate.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        PERFORM FmcjESDeallocate
        MOVE -1 TO retCode
        GOBACK
    END-IF

* create a process instance list
    CALL "SETADDR" USING listName name.
    CALL "SETADDR" USING localUserID ownerValue.
    CALL "SETADDR" USING desc description.
    CALL "SETADDR" USING FmcjNoFilter filter.
    CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
    MOVE FmcjNoThreshold TO threshold.
    MOVE Fmc-LT-Private TO typeValue.
    PERFORM FmcjESCreateProcInstList.

```

Figure 50. Sample COBOL program to create a process instance list (via PERFORM) (Part 1 of 2)

Examples

```
MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK
  DISPLAY "CreateProcessInstanceList returns - rc: "
  DISPLAY retCode
ELSE
  DISPLAY "CreateProcessInstanceList okay"
END-IF

PERFORM FmcjESLogoff.
PERFORM FmcjESDeallocate.
PERFORM FmcjGlobalDisconnect.
MOVE FMC-OK TO retCode.
GOBACK.

COPY fmcperf.
```

Figure 50. Sample COBOL program to create a process instance list (via PERFORM) (Part 2 of 2)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "VECTOR".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 listName     PIC X(50) VALUE z"MyTenInstances".
01 desc        PIC X(50)
               VALUE z"This list contains no more than 10 instances".

LINKAGE SECTION.

01 retCode      PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    CALL "FmcjGlobalConnect".
* logon
    CALL "FmcjExecutionServiceAllocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    CALL "FmcjExecutionServiceLogon"
        USING BY VALUE serviceValue
                    userID
                    passwordValue
                    Fmc-SM-Default
                    Fmc-SA-Reset
        RETURNING intReturnValue.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
        MOVE -1 TO retCode
        GOBACK
    END-IF

```

Figure 51. Sample COBOL program to create a process instance list (via CALL) (Part 1 of 2)

Examples

```
* create a process instance list
CALL "SETADDR" USING listName name.
CALL "SETADDR" USING localUserID ownerValue.
CALL "SETADDR" USING desc description.
CALL "SETADDR" USING FmcjNoFilter filter.
CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
CALL "FmcjExecutionServiceCreateProcessInstanceList"
  USING BY VALUE serviceValue
          name
          Fmc-LT-Private
          ownerValue
          description
          filter
          sortCriteria
          FmcjNoThreshold
  BY REFERENCE
  newList
  RETURNING
  intReturnValue.

MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK
  DISPLAY "CreateProcessInstanceList returns - rc: "
  DISPLAY retCode
ELSE
  DISPLAY "CreateProcessInstanceList okay"
END-IF

CALL "FmcjExecutionServiceLogoff"
  USING BY VALUE serviceValue
  RETURNING intReturnValue.
CALL "FmcjExecutionServiceDeallocate"
  USING BY REFERENCE serviceValue
  RETURNING intReturnValue.
CALL "FmcjGlobalDisconnect".
MOVE FMC-OK TO retCode.
GOBACK.
```

Figure 51. Sample COBOL program to create a process instance list (via CALL) (Part 2 of 2)

How to query persistent lists

The following examples show how to retrieve persistent lists from the MQSeries Workflow execution server and how to query the characteristics of a list. They use worklists as example. Other possible lists to query are process template lists or process instance lists.

Examples

Query worklists (C)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h> /* MQ Workflow Runtime API */
int main()
{
    APIRET rc = FMC_OK;
    FmcjExecutionServiceHandle service = 0;
    FmcjWorklistHandle worklist = 0;
    FmcjWorklistVectorHandle lists = 0;
    unsigned long numWList = 0;
    unsigned long i = 0;
    unsigned long enumValue = 0;
    char tInfo[4096+1]= "";

    FmcjGlobalConnect();

    /* logon */
    rc= FmcjExecutionServiceAllocate( &service );
    if (rc != FMC_OK)
    {
        printf("Service object could not be allocated - rc: %u\n",rc);
        return -1;
    }
    rc= FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_NotSet
                                   );
    if (rc != FMC_OK)
    {
        printf("Logon failed - rc: %u\n",rc);
        FmcjExecutionServiceDeallocate( &service );
        return -1;
    }

    /* query worklists */
    rc = FmcjExecutionServiceQueryWorklists( service, &lists );
    if ( rc != FMC_OK)
        printf( "QueryWorklists() returns: %u\n",rc );
    else
        printf( "QueryWorklists() returns okay\n" );

    if (rc == FMC_OK)
    {
        numWList= FmcjWorklistVectorSize(lists);
        printf( "Number of worklists returned : %u\n", numWList);
        for( i=1; i<= numWList; i++ )
        {
            worklist= FmcjWorklistVectorNextElement(lists);
            FmcjWorklistName( worklist, tInfo, 4097 );
            printf("- Name : %s\n",tInfo);
        }
    }
}
```

Figure 52. Sample C program to query worklists (Part 1 of 2)

```

enumValue= FmcjWorklistType(worklist);
if ( enumValue == Fmc_LT_Private )
    printf("- Type : %s\n","private");
if ( enumValue == Fmc_LT_Public )
    printf("- Type : %s\n","public");

FmcjWorklistOwnerOfList( worklist, tInfo, 4097 );
printf("- OwnerOfList : %s\n",tInfo);
printf("- OwnerOfList is null ? : %u\n",
        FmcjWorklistOwnerOfListIsNull(worklist) );

FmcjWorklistDescription( worklist, tInfo, 4097 );
printf("- Description : %s\n",tInfo);
printf("- Description is null ? : %u\n",
        FmcjWorklistDescriptionIsNull(worklist) );

FmcjWorklistFilter( worklist, tInfo, 4097 );
printf("- Filter : %s\n",tInfo);
printf("- Filter is null ? : %u\n",
        FmcjWorklistFilterIsNull(worklist) );

FmcjWorklistSortCriteria( worklist, tInfo, 4097 );
printf("- SortCriteria : %s\n",tInfo);
printf("- SortCriteria is null ? : %u\n",
        FmcjWorklistSortCriteriaIsNull(worklist) );

printf("- Threshold : %u\n",
        FmcjWorklistThreshold(worklist) );
printf("- Threshold is null ? : %u\n",
        FmcjWorklistThresholdIsNull(worklist) );
        /* deallocate just read object */
FmcjWorklistDeallocate(&worklist);
}
FmcjWorklistVectorDeallocate(&lists);
}

FmcjExecutionServiceLogoff( service );
FmcjExecutionServiceDeallocate( &service );

FmcjGlobalDisconnect();
return 0;
}

```

Figure 52. Sample C program to query worklists (Part 2 of 2)

Examples

Query worklists (C++)

```
#include <iomanip.h>
#include <bool.h> // bool
#include <fmcjstr.hxx> // string
#include <vector.h> // vector
#include <fmcjprun.hxx> // MQ Workflow Runtime API
int main()
{
    FmcjGlobal::Connect();

    // logon
    FmcjExecutionService service;
    APIRET rc = service.Logon("USERID", "password");
    if ( rc != FMC_OK )
    {
        cout << "Logon failed, - rc: " << rc << endl;
        return -1;
    }

    // query worklists

    vector<FmcjWorklist> lists;
    FmcjWorklist worklist;
    rc = service.QueryWorklists( lists );
    if ( rc != FMC_OK)
        cout << "QueryWorklists() returns: " << rc << endl;
    else
        cout << "QueryWorklists returns okay" << endl;
    if (rc == FMC_OK)
    {
        unsigned int numWList= lists.size();
        cout << "Number of worklists returned : " << numWList << endl;
        for( unsigned long i=0; i< numWList; i++ )
        {
            worklist= lists[i];
            cout << "Name : " << worklist.Name() << endl;
            cout << "Type : " <<
                ((worklist.Type() == FmcjPersistentList::Private) ? "private" :
                 (worklist.Type() == FmcjPersistentList::Public) ? "public" :
                 "not set" ) << endl;

            cout << "Owner : " << worklist.OwnerOfList() << endl;
            cout << "Owner null ? : " << worklist.OwnerOfListIsNull() << endl;
            cout << "Description : " << worklist.Description() << endl;
            cout << "Description null ? : " << worklist.DescriptionIsNull() << endl;
            cout << "Filter : " << worklist.Filter() << endl;
            cout << "Filter null ? : " << worklist.FilterIsNull() << endl;
            cout << "SortCriteria : " << worklist.SortCriteria() << endl;
            cout << "SortCriteria null?: " << worklist.SortCriteriaIsNull() << endl;
            cout << "Threshold : " << worklist.Threshold() << endl;
            cout << "Threshold null ? : " << worklist.ThresholdIsNull() << endl;
            cout << endl; } cout << endl; }

        rc = service.Logoff();
        FmcjGlobal::Disconnect();
        return 0;
    }
}
```

Figure 53. Sample C++ program to query worklists

Query worklists (Java)

```

import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class QueryWorkLists
{
    public static void main(String[] args)
    {
        // Check the arguments. The first argument is the name of the MQSeries
        // Workflow agent the client will connect to. The second argument defines
        // the locator policy the client will use when trying to contact the agent.
        // The third/fourth argument define the userid/password, which, if not
        // specified, default to USERID and password
        //
        if ((args.length < 2 ) || (args.length > 4 ))
        {
            System.out.println("Usage:");
            System.out.println("java QueryWorkLists [userid] [password]");
            System.exit(0);
        }

        try
        {
            // An agent bean representing an MQSeries Workflow domain
            String userid = "USERID";
            String passwd = "password";
            Agent agent = new Agent();

            // Parse the command line and set the locator to be used to
            // communicate with the agent.
            if (args[1].equalsIgnoreCase("LOC"))
            {
                agent.setLocator(Agent.LOC_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("RMI"))
            {
                agent.setLocator(Agent.RMI_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("OSA"))
            {
                agent.setLocator(Agent.OSA_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("IOR"))
            {
                agent.setLocator(Agent.IOR_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("COS"))
            {
                agent.setLocator(Agent.COS_LOCATOR);
            }
            else
            {
                System.out.println("Invalid locator policy: " + args[1]);
                System.exit(0);
            }
        }
    }
}

```

Figure 54. Sample Java program to query worklists (Part 1 of 3)

Examples

```
if (args.length >=3 ) userid = args[2].toUpperCase();
if (args.length >=4 ) passwd = args[3];

// Set the name of the Agent to be contacted. Setting the name
// automatically instructs the agent bean to contact the Agent using
// the current locator policy. For this reason the 'setLocator' must be
// called before 'setName' is invoked. If the agent bean cannot contact
// the Agent, it will raise a java.beans.PropertyVetoException instead
// of returning from the 'setName' call.
agent.setName(args[0]);

// Locate the default execution service in the system group named
// 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
// always returns successful (to prevent intrusion attempts which guess
// at service names until they find a valid one). Of course, only using
// a valid systemgroup and/or system name will return an ExecutionService
// which can be used to log on.
ExecutionService service = agent.locate("", "");

// Log on to the execution service. If the UserID and/or the password is
// invalid, an FmcException will be thrown.

// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
              AbsenceIndicator.LEAVE );
System.out.println("Logon successful");

// Query the set of worklists the logged on user can access.
WorkList[] worklists = service.queryWorkLists();

if (worklists.length == 0)
{
    System.out.println(" No worklist found");
}
else
{
    System.out.println(" Number of worklists returned: " + worklists.length
);
};

// Iterate over the worklists, printing out their names.
for (int ndx = 0; ndx < worklists.length; ndx++)
{
    System.out.println("    Name                : " + worklists[ndx].name());

    if (worklists[ndx].type() == TypeOfList.PUBLIC )
    {
        System.out.println("    Type                :Public " );
    }
    else if (worklists[ndx].type() == TypeOfList.PRIVATE )
    {
        System.out.println("    Type                :Private" );
    }
    else
    {
        System.out.println("    Type                :NotSet " );
    }
}
```

Figure 54. Sample Java program to query worklists (Part 2 of 3)

Examples

```
        System.out.println(" Owner           :"+ worklists[ndx].ownerOfList());
        System.out.println(" Description        :"+ worklists[ndx].description());
        System.out.println(" Filter           :"+ worklists[ndx].filter());
        System.out.println(" SortCriteria     :"+ worklists[ndx].sortCriteria());
        System.out.println(" Threshold        :"+ worklists[ndx].threshold());
        System.out.println(" ");
    }
}

}/* End if*/

// Logoff from the execution service. This (like any other remote call)
// may raise an FmcException indicating a communication failure.
service.logoff();

System.out.println("Logoff successful");
}

catch(FmcException e)
{
    // Catch and report details about the FmcException
    System.out.println("FmcException occured");
    System.out.println(" RC           : "+ e.rc);
    System.out.println(" Origin        : "+ e.origin);
    System.out.println(" MessageText: "+ e.messageText);
    System.out.println(" Exception    : "+ e.getMessage());
    System.out.println(" Parameters   : ");
    for ( int i = 0; i < e.parameters.length ; i++)
    {
        System.out.println("     "+ e.parameters[i] );
    }
    System.out.println(" StackTrace  : ");
    e.printStackTrace();
}

catch(Exception e)
{
    // Catch and report any exception that occurred.
    e.printStackTrace();
}

System.exit(0);
}
}
```

Figure 54. Sample Java program to query worklists (Part 3 of 3)

Examples

Query worklists (COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYWL".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID    PIC X(30) VALUE z"USERID".
01 localPassword  PIC X(30) VALUE z"PASSWORD".
01 numWList       PIC 9(9) BINARY VALUE 0.
01 tInfo          PIC X(4097).
01 i              PIC 9(9) BINARY VALUE 0.

LINKAGE SECTION.

01 retCode        PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.

* logon
    PERFORM FmcjESAllocate.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        PERFORM FmcjESDeallocate
        MOVE -1 TO retCode
        GOBACK
    END-IF

* query worklists
    PERFORM FmcjESQueryWorklists.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "QueryWorklists returns - rc: " retCode
    ELSE
        DISPLAY "QueryWorklists returns okay"
    END-IF
```

Figure 55. Sample COBOL program to query worklists (via PERFORM) (Part 1 of 3)

```

IF retCode = FMC-OK
  SET hdIVector TO lists
  PERFORM FmcjWLVectorSize
  MOVE ulongReturnValue TO numWList
  DISPLAY "Number of worklists returned : " numWList
  PERFORM VARYING i FROM 1 BY 1 UNTIL i >= numWList
    PERFORM FmcjWLVectorNextElement
    SET hdIList TO FmcjWLHandleReturnValue
    MOVE 4097 TO bufferLength
    CALL "SETADDR" USING tInfo listNameBuffer
    PERFORM FmcjWLName
    DISPLAY "- Name           : " tInfo
    PERFORM FmcjWLType
    IF intReturnValue = Fmc-LT-Private
      DISPLAY "- Type           : private"
    END-IF
    IF intReturnValue = Fmc-LT-Public
      DISPLAY "- Type           : public"
    END-IF
    CALL "SETADDR" USING tInfo userIdBuffer
    PERFORM FmcjWLOwnerOfList
    DISPLAY "- OwnerOfList      : " tInfo
    PERFORM FmcjWLOwnerOfListIsNull
    IF boolReturnValue = 0
      DISPLAY "- OwnerOfList is null ? : false"
    ELSE
      DISPLAY "- OwnerOfList is null ? : true"
    END-IF
    CALL "SETADDR" USING tInfo descriptionBuffer
    PERFORM FmcjWLDescription
    DISPLAY "- Description      : " tInfo
    PERFORM FmcjWLDescriptionIsNull
    IF boolReturnValue = 0
      DISPLAY "- Description is null ? : false"
    ELSE
      DISPLAY "- Description is null ? : true"
    END-IF
    CALL "SETADDR" USING tInfo filterBuffer
    PERFORM FmcjWLFilter
    DISPLAY "- Filter           : " tInfo
    PERFORM FmcjWLFilterIsNull
    IF boolReturnValue = 0
      DISPLAY "- Filter is null ?   : false"
    ELSE
      DISPLAY "- Filter is null ?   : true"
    END-IF
    CALL "SETADDR" USING tInfo sortCriteriaBuffer
    PERFORM FmcjWLSortCriteria
    DISPLAY "- SortCriteria      : " tInfo
    PERFORM FmcjWLSortCriteriaIsNull
    IF boolReturnValue = 0
      DISPLAY "- SortCriteria is null ? : false"
    ELSE
      DISPLAY "- SortCriteria is null ? : true"
    END-IF
    PERFORM FmcjWLThreshold
    DISPLAY "- Threshold         : " ulongReturnValue
    PERFORM FmcjWLThresholdIsNull

```

Figure 55. Sample COBOL program to query worklists (via PERFORM) (Part 2 of 3)

Examples

```
IF boolReturnValue = 0
  DISPLAY "- Threshold is null ? : false"
ELSE
  DISPLAY "- Threshold is null ? : true"
END-IF
PERFORM FmcjWLDeallocate
END-PERFORM
PERFORM FmcjWLVectorDeallocate
END-IF
PERFORM FmcjESLogoff.
PERFORM FmcjESDeallocate.
PERFORM FmcjGlobalDisconnect.
MOVE FMC-OK TO retCode.
GOBACK.

COPY fmcperf.
```

Figure 55. Sample COBOL program to query worklists (via PERFORM) (Part 3 of 3)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYWL".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcconst.
COPY fmcrs.

01 localUserID    PIC X(30) VALUE z"USERID".
01 localPassword  PIC X(30) VALUE z"PASSWORD".
01 numWList       PIC 9(9) BINARY VALUE 0.
01 tInfo          PIC X(4097).
01 i              PIC 9(9) BINARY VALUE 0.
01 bufferPtr      USAGE IS POINTER VALUE NULL.

LINKAGE SECTION.

01 retCode        PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    CALL "FmcjGlobalConnect".
* logon
    CALL "FmcjExecutionServiceAllocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    CALL "FmcjExecutionServiceLogon"
        USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
        RETURNING intReturnValue.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
        MOVE -1 TO retCode
        GOBACK
    END-IF

```

Figure 56. Sample COBOL program to query worklists (via CALL) (Part 1 of 4)

Examples

```
* query worklists
  CALL "FmcjExecutionServiceQueryWorklists"
    USING BY VALUE serviceValue
        BY REFERENCE lists
    RETURNING intReturnValue.
  MOVE intReturnValue TO retCode
  IF retCode NOT = FMC-OK
    DISPLAY "QueryWorklists returns - rc: " retCode
  ELSE
    DISPLAY "QueryWorklists returns okay"
  END-IF

  IF retCode = FMC-OK
    SET hd1Vector TO lists
    CALL "FmcjWorklistVectorSize"
      USING BY VALUE hd1Vector
      RETURNING ulongReturnValue
    MOVE ulongReturnValue TO numWList
    DISPLAY "Number of worklists returned : " numWList
    PERFORM VARYING i FROM 1 BY 1 UNTIL i >= numWList
      CALL "FmcjWorklistVectorNextElement"
        USING BY VALUE hd1Vector
        RETURNING FmcjWLHandleReturnValue
      SET hd1List TO FmcjWLHandleReturnValue
      MOVE 4097 TO bufferLength
      CALL "SETADDR" USING tInfo bufferPtr
      CALL "FmcjPersistentListName"
        USING BY VALUE hd1List
            bufferPtr
            bufferLength
        RETURNING pointerReturnValue
      DISPLAY "- Name           : " tInfo
      CALL "FmcjPersistentListType"
        USING BY VALUE hd1List
        RETURNING intReturnValue
      IF intReturnValue = Fmc-LT-Private
        DISPLAY "- Type           : private"
      END-IF
      IF intReturnValue = Fmc-LT-Public
        DISPLAY "- Type           : public"
      END-IF
      CALL "FmcjPersistentListOwnerOfList"
        USING BY VALUE hd1List
            bufferPtr
            bufferLength
        RETURNING pointerReturnValue
      DISPLAY "- OwnerOfList       : " tInfo
      CALL "FmcjPersistentListOwnerOfListIsNull"
        USING BY VALUE hd1List
        RETURNING boolReturnValue
      IF boolReturnValue = 0
        DISPLAY "- OwnerOfList is null ? : false"
      ELSE
        DISPLAY "- OwnerOfList is null ? : true"
      END-IF
      CALL "FmcjPersistentListDescription"
        USING BY VALUE hd1List
            bufferPtr
            bufferLength
        RETURNING pointerReturnValue
```

Figure 56. Sample COBOL program to query worklists (via CALL) (Part 2 of 4)


```

DISPLAY "- Description          : " tInfo
CALL "FmcjPersistentListDescriptionIsNull"
  USING BY VALUE hdList
  RETURNING boolReturnValue
IF boolReturnValue = 0
  DISPLAY "- Description is null ? : false"
ELSE
  DISPLAY "- Description is null ? : true"
END-IF
CALL "FmcjPersistentListFilter"
  USING BY VALUE hdList
                bufferPtr
                bufferLength
  RETURNING pointerReturnValue
DISPLAY "- Filter              : " tInfo
CALL "FmcjPersistentListFilterIsNull"
  USING BY VALUE hdList
  RETURNING boolReturnValue
IF boolReturnValue = 0
  DISPLAY "- Filter is null ?      : false"
ELSE
  DISPLAY "- Filter is null ?      : true"
END-IF
CALL "FmcjPersistentListSortCriteria"
  USING BY VALUE hdList
                bufferPtr
                bufferLength
  RETURNING pointerReturnValue
DISPLAY "- SortCriteria         : " tInfo
CALL "FmcjPersistentListSortCriteriaIsNull"
  USING BY VALUE hdList
  RETURNING boolReturnValue
IF boolReturnValue = 0
  DISPLAY "- SortCriteria is null ? : false"
ELSE
  DISPLAY "- SortCriteria is null ? : true"
END-IF
CALL "FmcjPersistentListThreshold"
  USING BY VALUE hdList
  RETURNING ulongReturnValue
DISPLAY "- Threshold           : " ulongReturnValue
CALL "FmcjPersistentListThresholdIsNull"
  USING BY VALUE hdList
  RETURNING boolReturnValue
IF boolReturnValue = 0
  DISPLAY "- Threshold is null ?    : false"
ELSE
  DISPLAY "- Threshold is null ?    : true"
END-IF
CALL "FmcjWorklistDeallocate"
  USING BY REFERENCE hdList
  RETURNING intReturnValue
END-PERFORM
CALL "FmcjWorklistVectorDeallocate"
  USING BY REFERENCE hdVector
  RETURNING intReturnValue
END-IF

```

Figure 56. Sample COBOL program to query worklists (via CALL) (Part 3 of 4)

Examples

```
CALL "FmcjExecutionServiceLogoff"  
    USING BY VALUE serviceValue  
    RETURNING intReturnValue.  
CALL "FmcjExecutionServiceDeallocate"  
    USING BY REFERENCE serviceValue  
    RETURNING intReturnValue.  
CALL "FmcjGlobalDisconnect".  
MOVE FMC-OK TO retCode.  
GOBACK.
```

Figure 56. Sample COBOL program to query worklists (via CALL) (Part 4 of 4)

How to query a set of objects

The following examples show how to query objects for which you are authorized. They use a query for process instances in order to demonstrate an ad-hoc query. They use work items in order to demonstrate how to query the contents of a predefined list, a worklist.

Examples

Query process instances (C)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h> /* MQ Workflow Runtime API */
int main()
{
    APIRET rc = FMC_OK;
    FmcjExecutionServiceHandle service = 0;
    FmcjProcessInstanceHandle instance = 0;
    FmcjProcessInstanceVectorHandle iList = 0;
    unsigned long numIList = 0;
    unsigned long i = 0;
    char tInfo[4096+1]= "";

    FmcjGlobalConnect();

    /* logon */
    rc= FmcjExecutionServiceAllocate( &service );
    if (rc != FMC_OK)
    {
        printf("Service object could not be allocated - rc: %u\n",rc);
        return -1;
    }
    rc= FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_NotSet
                                   );
    if (rc != FMC_OK)
    {
        printf("Logon failed - rc: %u\n",rc);
        FmcjExecutionServiceDeallocate( &service );
        return -1;
    }
    /* query process instances */
    rc= FmcjExecutionServiceQueryProcessInstances(
        service,
        FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
        &iList
    );
    if ( rc != FMC_OK)
        printf( "QueryProcessInstances() returns: %u\n",rc );
    else
        printf( "QueryProcessInstances() returns okay\n" );

    if (rc == FMC_OK)
    {
        numIList= FmcjProcessInstanceVectorSize(iList);
        printf ("Number of instances returned : %u\n", numIList);

        for( i=1; i<= numIList; i++ )
        {
            instance= FmcjProcessInstanceVectorNextElement(iList);
            FmcjProcessInstanceName( instance, tInfo, 4097 );
            printf("- Name : %s\n",tInfo);
            FmcjProcessInstanceDeallocate(&instance);
        }

        FmcjProcessInstanceVectorDeallocate(&iList);
    }
}
```

Figure 57. Sample C program to query process instances (Part 1 of 2)

```
FmcjExecutionServiceLogoff( service );  
FmcjExecutionServiceDeallocate( &service );  
  
FmcjGlobalDisconnect();  
return 0;  
}
```

Figure 57. Sample C program to query process instances (Part 2 of 2)

Examples

Query process instances (C++)

```
#include <iomanip.h>
#include <bool.h> // bool
#include <fmcjstr.hxx> // string
#include <vector.h> // vector
#include <fmcjprun.hxx> // MQ Workflow Runtime API
int main()
{
    FmcjGlobal::Connect();

    // logon
    FmcjExecutionService service;
    APIRET rc = service.Logon("USERID", "password");
    if ( rc != FMC_OK )
    {
        cout << "Logon failed, - rc: " << rc << endl;
        return -1;
    }

    // query process instances

    vector<FmcjProcessInstance> instances;

    rc = service.QueryProcessInstances(
        FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
        instances );
    if ( rc != FMC_OK)
        cout << "QueryProcessInstances returns: " << rc << endl;
    else
        cout << "QueryProcessInstances okay" << endl;
    if ( rc == FMC_OK )
    {
        cout << "Number of instances returned: " << instances.size() << endl;

        for ( int i=0; i < instances.size(); i++ )
            cout << "- Name: " << instances[i].Name() << endl;
    }

    service.Logoff();

    FmcjGlobal::Disconnect();
    return 0;
}
```

Figure 58. Sample C++ program to query process instances

Query process instances (Java)

```

import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryProcInst
{
    public static void main(String[] args)
    {
        // Check the arguments. The first argument is the name of the MQSeries
        // Workflow agent the client will connect to. The second argument defines
        // the locator policy the client will use when trying to contact the agent.
        // The third/fourth argument define the userid/password, which, if not
        // specified, default to USERID and password

        if ((args.length < 2 ) || (args.length > 4 ))
        {
            System.out.println("Usage:");
            System.out.println(" java QueryProcessInstances [userid] [password]");
            System.exit(0);
        }

        try
        {
            // An agent bean representing an MQSeries Workflow domain
            String userid = "USERID";
            String passwd = "password";
            Agent agent = new Agent();
            // Parse the command line and set the locator to be used to
            // communicate with the agent.
            if (args[1].equalsIgnoreCase("LOC"))
            {
                agent.setLocator(Agent.LOC_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("RMI"))
            {
                agent.setLocator(Agent.RMI_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("OSA"))
            {
                agent.setLocator(Agent.OSA_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("IOR"))
            {
                agent.setLocator(Agent.IOR_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("COS"))
            {
                agent.setLocator(Agent.COS_LOCATOR);
            }
            else
            {
                System.out.println("Invalid locator policy: " + args[1]);
                System.exit(0);
            }
        }
    }
}

```

Figure 59. Sample Java program to query process instances (Part 1 of 3)

Examples

```
if (args.length >=3 ) userid = args[2].toUpperCase();
if (args.length >=4 ) passwd = args[3];

// Set the name of the Agent to be contacted. Setting the name
// automatically instructs the agent bean to contact the Agent using
// the current locator policy. For this reason the 'setLocator' must be
// called before 'setName' is invoked. If the agent bean cannot contact
// the Agent, it will raise a java.beans.PropertyVetoException instead
// of returning from the 'setName' call.
agent.setName(args[0]);

// Locate the default execution service in the system group named
// 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
// always returns successful (to prevent intrusion attempts which guess
// at service names until they find a valid one). Of course, only using
// a valid systemgroup and/or system name will return an ExecutionService
// which can be used to log on.
ExecutionService service = agent.locate("", "");

// Log on to the execution service. If the UserID and/or the password is
// invalid, an FmcException will be thrown.
// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
               AbsenceIndicator.LEAVE );
System.out.println("Logon successful");

// Query a set of processinstances (30 at maximum), sort them by name
ProcessInstance[] procInstances =
    service.queryProcessInstances("", "NAME DESC", 30);

if (procInstances.length == 0)
{
    System.out.println(" No process instances found");
}
else
{
    System.out.println("Number of instances returned: " + procInstances.length);

    // Iterate over the process instances, printing out their names.
    for (int ndx = 0; ndx < procInstances.length; ndx++)
    {
        System.out.println(" - Name: " + procInstances[ndx].name());
    }
}

// Logoff from the execution service. This (like any other remote call)
// may raise an FmcException indicating a communication failure.
service.logoff();

System.out.println("Logoff successful");
}
```

Figure 59. Sample Java program to query process instances (Part 2 of 3)


```

catch(FmcException e)
{
    // Catch and report details about the FmcException
    System.out.println("FmcException occured");
    System.out.println(" RC      : " + e.rc);
    System.out.println(" Origin   : " + e.origin);
    System.out.println(" MessageText: " + e.messageText);
    System.out.println(" Exception : " + e.getMessage());
    System.out.println(" Parameters :");
    for ( int i = 0; i < e.parameters.length ; i++)
    {
        System.out.println("    " + e.parameters[i] );
    }
    System.out.println(" StackTrace : ");
    e.printStackTrace();
}

catch(Exception e)
{
    // Catch and report any exception that occurred.
    e.printStackTrace();
}

System.exit(0);
}
}

```

Figure 59. Sample Java program to query process instances (Part 3 of 3)

Examples

Query process instances (COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYPI".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 numIList      PIC 9(9)  BINARY VALUE 0.
01 tInfo         PIC X(4097).
01 i             PIC 9(9)  BINARY VALUE 0.

LINKAGE SECTION.

01 retCode       PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.

* logon
    PERFORM FmcjESAllocate.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        PERFORM FmcjESDeallocate
        MOVE -1 TO retCode
        GOBACK
    END-IF
```

Figure 60. Sample COBOL program to query process instances (via PERFORM) (Part 1 of 2)

```

* query process instances
  CALL "SETADDR" USING FmcjNoFilter filter.
  CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
  MOVE FmcjNoThreshold TO threshold.
  PERFORM FmcjESQueryProcInsts.

  SET hdlVector TO instances.
  MOVE intReturnValue TO retCode
  IF retCode NOT = FMC-OK
    DISPLAY "QueryProcessInstances returns: " retCode
  ELSE
    DISPLAY "QueryProcessInstances returns okay"
  END-IF

  IF retCode = FMC-OK
    PERFORM FmcjPIVSize
    MOVE ulongReturnValue TO numIList
    DISPLAY "Number of instances returned: " numIList
    MOVE 4097 TO bufferLength
    CALL "SETADDR" USING tInfo instanceNameBuffer
    PERFORM VARYING i FROM 1 BY 1 UNTIL i > numIList
      PERFORM FmcjPIVNextElement
      SET hdlInstance TO FmcjPIHandleReturnValue
      PERFORM FmcjPIName
      DISPLAY "- name: " tInfo
      PERFORM FmcjPIDeallocate
    END-PERFORM
    PERFORM FmcjPIVDeallocate
  END-IF

  PERFORM FmcjESLogoff.
  PERFORM FmcjESDeallocate.
  PERFORM FmcjGlobalDisconnect.
  MOVE FMC-OK TO retCode.
  GOBACK.

  COPY fmcperf.

```

Figure 60. Sample COBOL program to query process instances (via PERFORM) (Part 2 of 2)

Examples

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYPI".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 numIList     PIC 9(9) BINARY VALUE 0.
01 tInfo       PIC X(4097).
01 i           PIC 9(9) BINARY VALUE 0.

LINKAGE SECTION.

01 retCode      PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    CALL "FmcjGlobalConnect".
* logon
    CALL "FmcjExecutionServiceAllocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    CALL "FmcjExecutionServiceLogon"
        USING BY VALUE serviceValue
                    userID
                    passwordValue
                    Fmc-SM-Default
                    Fmc-SA-Reset
        RETURNING intReturnValue.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
        MOVE -1 TO retCode
        GOBACK
    END-IF
```

Figure 61. Sample COBOL program to query process instances (via CALL) (Part 1 of 2)

```

* query process instances
  CALL "SETADDR" USING FmcjNoFilter filter.
  CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
  CALL "FmcjExecutionServiceQueryProcessInstances"
    USING BY VALUE      serviceValue
                       filter
                       sortCriteria
                       FmcjNoThreshold
    BY REFERENCE instances
    RETURNING intReturnValue.

SET hd1Vector TO instances.
MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK
  DISPLAY "QueryProcessInstances returns: " retCode
ELSE
  DISPLAY "QueryProcessInstances returns okay"
END-IF

IF retCode = FMC-OK
  CALL "FmcjProcessInstanceVectorSize"
    USING BY VALUE hd1Vector
    RETURNING ulongReturnValue
  MOVE ulongReturnValue TO numIList
  DISPLAY "Number of instances returned: " numIList
  MOVE 4097 TO bufferLength
  CALL "SETADDR" USING tInfo instanceNameBuffer
  PERFORM VARYING i FROM 1 BY 1 UNTIL i > numIList
    CALL "FmcjProcessInstanceVectorNextElement"
      USING BY VALUE hd1Vector
      RETURNING FmcjPIHandleReturnValue
    SET hd1Instance TO FmcjPIHandleReturnValue
    CALL "FmcjProcessInstanceName"
      USING BY VALUE hd1Instance
                  instanceNameBuffer
                  FMC-PROC-INST-NAME-LENGTH
      RETURNING pointerReturnValue
    DISPLAY "- name: " tInfo
    CALL "FmcjProcessInstanceDeallocate"
      USING BY REFERENCE hd1Instance
      RETURNING intReturnValue
    END-PERFORM
  CALL "FmcjProcessInstanceVectorDeallocate"
    USING BY REFERENCE hd1Vector
    RETURNING intReturnValue
  END-IF

  CALL "FmcjExecutionServiceLogoff"
    USING BY VALUE serviceValue
    RETURNING intReturnValue.
  CALL "FmcjExecutionServiceDeallocate"
    USING BY REFERENCE serviceValue
    RETURNING intReturnValue.
  CALL "FmcjGlobalDisconnect".
  MOVE FMC-OK TO retCode.
  GOBACK.

```

Figure 61. Sample COBOL program to query process instances (via CALL) (Part 2 of 2)

Examples

Query work items from a worklist (C)

```
#include <stdio.h>
#include <string.h>
#include <fmcjcrun.h>          /* MQ Workflow Runtime API */

int main (int argc, char ** argv)
{
    APIRET                rc          = FMC_OK;
    FmcjExecutionServiceHandle  service    = 0;
    FmcjWorklistVectorHandle    wLists    = 0;
    FmcjWorklistHandle          worklist  = 0;
    FmcjWorkitemVectorHandle    wVector   = 0;
    FmcjWorkitemHandle          workitem   = 0;
    unsigned long               numWList  = 0;
    char                         tInfo[4096+1] = "";

    FmcjGlobalConnect();

    /* Logon */
    rc= FmcjExecutionServiceAllocate( &service );
    if (rc != FMC_OK)
    {
        printf("Service object could not be allocated: %u%\n",rc);
        return -1;
    }

    rc= FmcjExecutionServiceLogon( service,
                                   "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_NotSet );

    if ( rc != FMC_OK )
    {
        printf("Logon failed - rc : %u%\n",rc);
        rc= FmcjExecutionServiceDeallocate( &service );
        return -1;
    }

    /* query worklists */
    rc = FmcjExecutionServiceQueryWorklists( service, &wLists );
    if ( rc != FMC_OK)
        printf( "QueryWorklists() returns: %u%\n",rc );
    else
        printf( "QueryWorklists() returns okay\n" );

    if (rc == FMC_OK)
    {
        numWList= FmcjWorklistVectorSize(wLists);
        printf ("Number of worklists returned : %u%\n", numWList);
        if ( numWList == 0 )
        {
            printf("No worklist found \n");
            FmcjWorklistVectorDeallocate(&wLists);
            rc= FmcjExecutionServiceDeallocate( &service );
            return -1;
        }

        worklist= FmcjWorklistVectorFirstElement(wLists);
        FmcjWorklistName( worklist, tInfo, 4097 );
        printf("Name                : %s%\n",tInfo);
    }
}
```

Figure 62. Sample C program to query work items from a worklist (Part 1 of 2)

```

/* query workitems */
rc= FmcjWorklistQueryWorkitems( worklist, &wVector );
printf("\nQuery workitems of list returns rc: %u\n",rc);

if (rc == FMC_OK)
{
  while ( 0 != (workitem= FmcjWorkitemVectorNextElement(wVector)) )
  {
    FmcjWorkitemName( workitem, tInfo, 4097 );
    printf("- Name           : %s\n",tInfo);

    FmcjWorkitemDeallocate(&workitem);
  }
}

FmcjWorklistDeallocate(&worklist);
FmcjWorklistVectorDeallocate(&wLists);
}

/* Logoff */
rc= FmcjExecutionServiceLogoff(service);
rc= FmcjExecutionServiceDeallocate( &service );

FmcjGlobalDisconnect();
return 0;
}

```

Figure 62. Sample C program to query work items from a worklist (Part 2 of 2)

Examples

Query work items from a worklist (C++)

```
#include <iomanip.h>
#include <bool.h> // bool
#include <fmcjstr.hxx> // string
#include <vector.h> // vector
#include <fmcjprun.hxx> // MQ Workflow Runtime API
int main()
{
    FmcjGlobal::Connect();

    // logon
    FmcjExecutionService service;
    APIRET rc = service.Logon("USERID", "password");
    if ( rc != FMC_OK )
    {
        cout << "Logon failed, - rc: " << rc << endl;
        return -1;
    }

    // query worklists

    vector<FmcjWorklist> lists;
    FmcjWorklist worklist;

    rc = service.QueryWorklists( lists );
    if ( rc != FMC_OK )
        cout << "QueryWorklists() returns: " << rc << endl;
    else
        cout << "QueryWorklists returns okay" << endl;

    if (rc == FMC_OK)
    {
        unsigned int numWList= lists.size();
        cout << "Number of worklists returned : " << numWList << endl;
        if ( numWList == 0 )
        {
            cout << "No worklist found" << endl;
            return -1;
        }

        worklist= lists[0];
        cout << "Name : " << worklist.Name() << endl;

        vector<FmcjWorkitem> wVector;
        FmcjWorkitem workitem;

        rc= worklist.QueryWorkitems( wVector );
        cout << "Query workitems of list returns: " << rc << endl;
        cout << "Number of workitems " << wVector.size() << endl;
    }
}
```

Figure 63. Sample C++ program to query work items from a worklist (Part 1 of 2)


```
if (rc == FMC_OK)
{
    for ( int i= 0; i < wVector.size(); i++ )
    {
        workitem= wVector[i];
        cout << "Name           : " << workitem.Name() << endl;
    }
}

rc = service.Logoff();

FmcjGlobal::Disconnect();
return 0;
}
```

Figure 63. Sample C++ program to query work items from a worklist (Part 2 of 2)

Examples

Query work items from a worklist (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryWorkItems
{
    public static void main(String[] args)
    {
        // Check the arguments. The first argument is the name of the MQSeries
        // Workflow agent the client will connect to. The second argument defines
        // the locator policy the client will use when trying to contact the agent.
        // The third/fourth argument define the userid/password, which, if not
        // specified, default to USERID and password

        if ((args.length < 2 ) || (args.length > 4 ))
        {
            System.out.println("Usage:");
            System.out.println(" java QueryWorkItems [userid] [password]");
            System.exit(0);
        }

        try
        {
            // An agent bean representing an MQSeries Workflow domain
            String userid = "USERID";
            String passwd = "password";
            Agent agent = new Agent();

            // Parse the command line and set the locator to be used to
            // communicate with the agent.
            if (args[1].equalsIgnoreCase("LOC"))
            {
                agent.setLocator(Agent.LOC_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("RMI"))
            {
                agent.setLocator(Agent.RMI_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("OSA"))
            {
                agent.setLocator(Agent.OSA_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("IOR"))
            {
                agent.setLocator(Agent.IOR_LOCATOR);
            }
            else if (args[1].equalsIgnoreCase("COS"))
            {
                agent.setLocator(Agent.COS_LOCATOR);
            }
            else
            {
                System.out.println("Invalid locator policy: " + args[1]);
                System.exit(0);
            }

            if (args.length >=3 ) userid = args[2].toUpperCase();
            if (args.length >=4 ) passwd = args[3];
        }
    }
}
```

Figure 64. Sample Java program to query work items from a worklist (Part 1 of 3)

```

// Set the name of the Agent to be contacted. Setting the name
// automatically instructs the agent bean to contact the Agent using
// the current locator policy. For this reason the 'setLocator' must be
// called before 'setName' is invoked. If the agent bean cannot contact
// the Agent, it will raise a java.beans.PropertyVetoException instead
// of returning from the 'setName' call.
agent.setName(args[0]);

// Locate the default execution service in the system group named
// 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
// always returns successful (to prevent intrusion attempts which guess
// at service names until they find a valid one). Of course, only using
// a valid systemgroup and/or system name will return an ExecutionService
// which can be used to log on.
ExecutionService service = agent.locate("", "");

// Log on to the execution service. If the UserID and/or the password is
// invalid, a FmcException will be thrown.
// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
               AbsenceIndicator.LEAVE );
System.out.println("Logon successful");

// Query the set of worklists the logged on user can access.
WorkList[] worklists = service.queryWorkLists();

if (worklists.length == 0)
{
    System.out.println(" No worklist found");
}
else
{
    System.out.println(" Number of worklists returned: " + worklists.length);

    WorkList worklist = worklists[0];
    System.out.println(" Name: "+worklist.name());

    // Query the set of workitems in the first worklist.
    WorkItem[] workitems = worklist.queryWorkItems();
    System.out.println(" Number of workitems: " + workitems.length);

    // Iterate over the workitems, printing out their names.
    for (int ndx = 0; ndx < workitems.length; ndx++)
    {
        System.out.println("    " + workitems[ndx].name());
    }
}/* End if*/

// Logoff from the execution service. This (like any other remote call)
// may raise an FmcException indicating a communication failure.
service.logoff();

System.out.println("Logoff successful");
}

```

Figure 64. Sample Java program to query work items from a worklist (Part 2 of 3)

Examples

```
catch(FmcException e)
{
    // Catch and report details about the FmcException
    System.out.println("FmcException occurred");
    System.out.println(" RC      : " + e.rc);
    System.out.println(" Origin   : " + e.origin);
    System.out.println(" MessageText: " + e.messageText);
    System.out.println(" Exception : " + e.getMessage());
    System.out.println(" Parameters :");
    for ( int i = 0; i < e.parameters.length ; i++)
    {
        System.out.println("    " + e.parameters[i] );
    }
    System.out.println(" StackTrace : ");
    e.printStackTrace();
}

catch(Exception e)
{
    // Catch and report any exception that occurred.
    e.printStackTrace();
}

System.exit(0);
}
```

Figure 64. Sample Java program to query work items from a worklist (Part 3 of 3)

Query work items from a worklist (COBOL)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYWI".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcconst.
COPY fmcrs.

01 localUserID    PIC X(30) VALUE z"USERID".
01 localPassword  PIC X(30) VALUE z"PASSWORD".
01 numWList       PIC 9(9) BINARY VALUE 0.
01 tInfo          PIC X(4097).

LINKAGE SECTION.

01 retCode        PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    PERFORM FmcjGlobalConnect.

* logon
    PERFORM FmcjESAllocate.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    MOVE Fmc-SM-Default TO sessionMode.
    MOVE Fmc-SA-Reset TO absenceIndicator.
    PERFORM FmcjESLogon.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        PERFORM FmcjESDeallocate
        MOVE -1 TO retCode
        GOBACK
    END-IF

* query worklists
    PERFORM FmcjESQueryWorklists.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "QueryWorklists returns - rc: " retCode
    ELSE
        DISPLAY "QueryWorklists returns okay"
    END-IF

```

Figure 65. Sample COBOL program to query work items from a worklist (via PERFORM)
(Part 1 of 2)

Examples

```
IF retCode = FMC-OK
  SET hd1Vector TO lists
  PERFORM FmcjWLVectorSize
  MOVE ulongReturnValue TO numWList
  DISPLAY "Number of worklists returned : " numWList
  IF numWList = 0
    DISPLAY "No worklist found"
    PERFORM FmcjWLDeallocate
    PERFORM FmcjESDeallocate
    MOVE -1 TO retCode
    GOBACK
  END-IF

  PERFORM FmcjWLVectorFirstElement
  SET hd1List TO FmcjWLHandleReturnValue
  MOVE 4097 TO bufferLength
  CALL "SETADDR" USING tInfo listNameBuffer
  PERFORM FmcjWLName
  DISPLAY "Name           : " tInfo

* query workitems
  PERFORM FmcjWLQueryWorkitems
  MOVE intReturnValue TO retCode
  DISPLAY "Query workitems of list returns rc:" retCode
  SET hd1Vector TO workitems
  CALL "SETADDR" USING tInfo itemNameBuffer
  IF retCode = FMC-OK
    PERFORM FmcjWIVNextElement
    SET hd1Item TO FmcjWIHandleReturnValue
    PERFORM UNTIL pointerReturnValue = NULL
      PERFORM FmcjWIName
      DISPLAY "- Name           : " tInfo
      PERFORM FmcjWIDeallocate
      PERFORM FmcjWIVNextElement
    END-PERFORM
  END-IF
  PERFORM FmcjWLDeallocate
  PERFORM FmcjWLVectorDeallocate
END-IF

PERFORM FmcjESLogoff.
PERFORM FmcjESDeallocate.
PERFORM FmcjGlobalDisconnect.
MOVE FMC-OK TO retCode.
GOBACK.

COPY fmcperf.
```

Figure 65. Sample COBOL program to query work items from a worklist (via PERFORM)
(Part 2 of 2)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYWI".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmconst.
COPY fmcrs.

01 localUserID   PIC X(30) VALUE z"USERID".
01 localPassword PIC X(30) VALUE z"PASSWORD".
01 numWList      PIC 9(9) BINARY VALUE 0.
01 tInfo         PIC X(4097).

LINKAGE SECTION.

01 retCode       PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    CALL "FmcjGlobalConnect".
* logon
    CALL "FmcjExecutionServiceAllocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING localUserId userId.
    CALL "SETADDR" USING localPassword passwordValue.
    CALL "FmcjExecutionServiceLogon"
        USING BY VALUE serviceValue
        userID
        passwordValue
        Fmc-SM-Default
        Fmc-SA-Reset
        RETURNING intReturnValue.

    MOVE intReturnValue TO retCode
    IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
        MOVE -1 TO retCode
        GOBACK
    END-IF

```

Figure 66. Sample COBOL program to query work items from a worklist (via CALL) (Part 1 of 3)

Examples

```
* query worklists
  CALL "FmcjExecutionServiceQueryWorklists"
    USING BY VALUE serviceValue
        BY REFERENCE lists
    RETURNING intReturnValue.
  MOVE intReturnValue TO retCode
  IF retCode NOT = FMC-OK
    DISPLAY "QueryWorklists returns - rc: " retCode
  ELSE
    DISPLAY "QueryWorklists returns okay"
  END-IF

  IF retCode = FMC-OK
    SET hd1Vector TO lists
    CALL "FmcjWorklistVectorSize"
      USING BY VALUE hd1Vector
      RETURNING ulongReturnValue
    MOVE ulongReturnValue TO numWList
    DISPLAY "Number of worklists returned : " numWList
    IF numWList = 0
      DISPLAY "No worklist found"
      CALL "FmcjWorklistDeallocate"
        USING BY REFERENCE hd1List
        RETURNING intReturnValue
      CALL "FmcjExecutionServiceDeallocate"
        USING BY REFERENCE serviceValue
        RETURNING intReturnValue
      MOVE -1 TO retCode
      GOBACK
    END-IF

    CALL "FmcjWorklistVectorFirstElement"
      USING BY VALUE hd1Vector
      RETURNING FmcjWLHandleReturnValue
    SET hd1List TO FmcjWLHandleReturnValue
    MOVE 4097 TO bufferLength
    CALL "SETADDR" USING tInfo listNameBuffer
    CALL "FmcjPersistentListName"
      USING BY VALUE hd1List
          listNameBuffer
          bufferLength
      RETURNING pointerReturnValue
    DISPLAY "Name          : " tInfo

* query workitems
  CALL "FmcjWorklistQueryWorkitems"
    USING BY VALUE hd1List
        BY REFERENCE workitems
    RETURNING intReturnValue
  MOVE intReturnValue TO retCode
  DISPLAY "Query workitems of list returns rc:" retCode
  SET hd1Vector TO workitems
  CALL "SETADDR" USING tInfo itemNameBuffer
  IF retCode = FMC-OK
    CALL "FmcjWorkitemVectorNextElement"
      USING BY VALUE hd1Vector
      RETURNING FmcjWIHandleReturnValue
    SET hd1Item TO FmcjWIHandleReturnValue
```

Figure 66. Sample COBOL program to query work items from a worklist (via CALL) (Part 2 of 3)


```

PERFORM UNTIL pointerReturnValue = NULL
  CALL "FmcjItemName"
    USING BY VALUE hdItem
      itemNameBuffer
      bufferLength
    RETURNING pointerReturnValue
  DISPLAY "- Name          : " tInfo
  CALL "FmcjWorkitemDeallocate"
    USING BY REFERENCE hdWorkitem
    RETURNING intReturnValue
  CALL "FmcjWorkitemVectorNextElement"
    USING BY VALUE hdVector
    RETURNING FmcjWIHandleReturnValue
END-PERFORM
END-IF
CALL "FmcjWorklistDeallocate"
  USING BY REFERENCE hdList
  RETURNING intReturnValue
CALL "FmcjWorklistVectorDeallocate"
  USING BY REFERENCE hdVector
  RETURNING intReturnValue
END-IF

  CALL "FmcjExecutionServiceLogoff"
  USING BY VALUE serviceValue
  RETURNING intReturnValue.
CALL "FmcjExecutionServiceDeallocate"
  USING BY REFERENCE serviceValue
  RETURNING intReturnValue.
CALL "FmcjGlobalDisconnect".
MOVE FMC-OK TO retCode.
GOBACK.

```

Figure 66. Sample COBOL program to query work items from a worklist (via CALL) (Part 3 of 3)

Examples

How to code an activity implementation

The following examples show the concept of how to query and set containers from within an activity implementation. Refer to the examples provided with the product for more details.

Programming an activity implementation (C)

```

#include <stdio.h>
#include <fmcjcon.h>          /* MQ Workflow Container API */
int main()
{
    FILE                * file1          = 0;
    APIRET              rc               = FMC_OK;
    FmcjReadOnlyContainerHandle input    = 0;
    FmcjReadWriteContainerHandle output = 0;
    char                stringBuffer[4097]="";

    /*- keep results in a file -----*/
    file1 = fopen ("sample.out", "a");
    if ( file1 == 0 )
        return -1;
    fprintf(file1, "\n----- C-API Activity Implementation called ----- \n");
    fflush(file1);

    FmcjGlobalConnect();

    /*-- retrieve the input container from the PEA who started the program ---*/
    rc = FmcjContainerInContainer( &input );
    fprintf(file1, "Get Input Container - rc: %u\n", rc);
    if (rc != FMC_OK)
    {
        fclose(file1);
        return 1;
    }

    fprintf(file1, "Input Container Name: %s\n",
               FmcjReadOnlyContainerType(input, stringBuffer, 4097));

    /*-- retrieve the output container from the PEA who started the program --*/
    rc = FmcjContainerOutContainer( &output );
    fprintf(file1, "Get Output Container - rc: %u\n", rc);
    if (rc != FMC_OK)
    {
        fclose(file1);
        return 1;
    }

    fprintf(file1, "Output Container Name: %s\n",
               FmcjReadWriteContainerType(output, stringBuffer, 4097));

    /*----- Modify output values -----*/
    rc= FmcjReadWriteContainerSetLongValue(output, "aFieldInTheOutput",42);
    fprintf(file1, "\nSetting long value returns rc: %u\n", rc);

    ...

    /*-- return the output container to the PEA who started the program -----*/
    rc = FmcjContainerSetOutContainer( output );
    fprintf(file1, "\nSet Output Container - rc: %u\n",rc);
    fflush(file1);

    FmcjGlobalDisconnect();
    fclose(file1);
    return 0;                          // _RC passed to MQSeries Workflow
}

```

Figure 67. Sample activity implementation (C)

Examples

Programming an activity implementation (C++)

```
#include <fstream.h>
#include <bool.h>                // bool
#include <fmcjstr.hxx>           // string
#include <vector.h>             // vector
#include <fmcjpcon.hxx>         // MQ Workflow Container API
int main()
{
/*- keep results in a file -----*/
ofstream file1("sample.out");
if ( file1 == 0 )
    return -1;

    file1 << "\n----- C++-API Activity Implementation called -----\n" << endl;
    FmcjGlobal::Connect();

/*-- retrieve the input container from the PEA who started the program --*/
    FmcjReadOnlyContainer input;

    APIRET rc = FmcjContainer::InContainer( input );
    file1 << "Get Input Container - rc: " << rc << endl;
    if (rc != FMC_OK)
    {
        file1.close();
        return 1;
    }

    file1 << "Input Container Name: " << input.Type() << endl;
/*-- retrieve the output container from the PEA who started the program --*/
    FmcjReadWriteContainer output;

    rc = FmcjContainer::OutContainer( output );
    file1 << "Get Output Container - rc: " << rc << endl;
    if (rc != FMC_OK)
    {
        file1.close();
        return 1;
    }

    file1 << "Output Container Name: " << output.Type() << endl;
/*----- Modify output values -----*/
    rc= output.SetValue("aFieldInTheOutput",42L);
    file1 << "Setting long value returns rc: " << rc << endl;

    ...
/*-- return the output container to the PEA who started the program -----*/
    rc = FmcjContainer::SetOutContainer( output );
    file1 << "Set Output Container - rc: " << rc << endl;

    FmcjGlobal::Disconnect();
    file1.close();
    return 0;                // _RC passed to MQSeries Workflow
}
```

Figure 68. Sample activity implementation (C++)

Programming an activity implementation (COBOL)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "EXEC".

DATA DIVISION.

WORKING-STORAGE SECTION.

COPY fmcvars.
COPY fmcrcs.

01 stringBuffer PIC X(4097).
01 fieldName PIC X(39) VALUE z"aFieldInTheOutput".

LINKAGE SECTION.

01 retCode PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    DISPLAY "-- COBOL-API Activity Implementation called --"
    PERFORM FmcjGlobalConnect.

* retrieve the input container
    PERFORM FmcjInCtnr.
    MOVE intReturnValue TO retCode.
    DISPLAY "Get Input Container - rc: " retCode.
    IF retCode NOT = FMC-OK
        MOVE 1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING stringBuffer containerTypeBuffer.
    MOVE 4097 TO bufferLength.
    SET hd1Container TO inputValue.
    PERFORM FmcjROType.
    DISPLAY "Input Container Name: " stringBuffer.

* retrieve the output container
    PERFORM FmcjOutCtnr.
    MOVE intReturnValue TO retCode.
    DISPLAY "Get Output Container - rc: " retCode.
    IF retCode NOT = FMC-OK
        MOVE 1 TO retCode
        GOBACK
    END-IF

    SET hd1Container TO outputValue.
    PERFORM FmcjRWType.
    DISPLAY "Output Container Name: " stringBuffer.

```

Figure 69. Sample activity implementation (COBOL, via PERFORM) (Part 1 of 2)

Examples

```
* modify output values
  MOVE 42 TO intValue.
  CALL "SETADDR" USING fieldName qualifiedName.
  PERFORM FmcjRWCSetLongValue.
  MOVE intReturnValue TO retCode.
  DISPLAY "Setting long value returns rc: " retCode.

* return the output container
  PERFORM FmcjCSetOutCtnr.
  MOVE intReturnValue TO retCode.
  DISPLAY "Set Output Container - rc: " retCode.

  PERFORM FmcjGlobalDisconnect.
  MOVE FMC-OK TO retCode.
  GOBACK.

  COPY fmcperf.
```

Figure 69. Sample activity implementation (COBOL, via PERFORM) (Part 2 of 2)

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "EXEC".

DATA DIVISION.

    WORKING-STORAGE SECTION.

        COPY fmcvars.
        COPY fmcrcs.

        01 stringBuffer PIC X(4097).
        01 fieldName PIC X(39) VALUE z"aFieldInTheOutput".

    LINKAGE SECTION.

        01 retCode PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

    DISPLAY "-- COBOL-API Activity Implementation called --"
    CALL "FmcjGlobalConnect".

* retrieve the input container
    CALL "FmcjContainerInContainer"
        USING BY REFERENCE inputValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode.
    DISPLAY "Get Input Container - rc: " retCode.
    IF retCode NOT = FMC-OK
        MOVE 1 TO retCode
        GOBACK
    END-IF

    CALL "SETADDR" USING stringBuffer containerTypeBuffer.
    MOVE 4097 TO bufferLength.
    SET hdlContainer TO inputValue.
    CALL "FmcjContainerType"
        USING BY VALUE hdlContainer
            containerTypeBuffer
            bufferLength
        RETURNING pointerReturnValue.
    DISPLAY "Input Container Name: " stringBuffer.

* retrieve the output container
    CALL "FmcjContainerOutContainer"
        USING BY REFERENCE outputValue
        RETURNING intReturnValue.
    MOVE intReturnValue TO retCode.
    DISPLAY "Get Output Container - rc: " retCode.
    IF retCode NOT = FMC-OK
        MOVE 1 TO retCode
        GOBACK
    END-IF

    SET hdlContainer TO outputValue.
    CALL "FmcjContainerType"
        USING BY VALUE hdlContainer
            containerTypeBuffer
            bufferLength
        RETURNING pointerReturnValue.
    DISPLAY "Output Container Name: " stringBuffer.

* modify output values
    MOVE 42 TO intValue.
    CALL "SETADDR" USING fieldName qualifiedName.

```

Figure 70. Sample activity implementation (COBOL, via CALL) (Part 1 of 2)

Examples

```
CALL "FmcjReadWriteContainerSetLongValue"  
  USING BY VALUE hdContainer  
           qualifiedName  
           intValue  
  RETURNING intReturnValue.  
MOVE intReturnValue TO retCode.  
DISPLAY "Setting long value returns rc: " retCode.  
  
* return the output container  
CALL "FmcjContainerSetOutContainer"  
  USING BY VALUE outputValue  
  RETURNING intReturnValue.  
MOVE intReturnValue TO retCode.  
DISPLAY "Set Output Container - rc: " retCode.  
  
CALL "FmcjGlobalDisconnect".  
MOVE FMC-OK TO retCode.  
GOBACK.  
  
COPY fmcperf.
```

Figure 70. Sample activity implementation (COBOL, via CALL) (Part 2 of 2)

Glossary

This glossary defines terms and abbreviations used in MQSeries Workflow for OS/390 publications. If you do not find the term you are looking for, refer to the index of the appropriate manual or view the *IBM Dictionary of Computing* located at:

<http://www.ibm.com/networking/nsg/nsgmain.htm>

A

administration server. The MQ Workflow component that performs administration functions within an MQ Workflow system. Functions include starting and stopping of the MQ Workflow system, performing error management, and participating in administrative functions for a system group.

activity. One of the steps that make up a process model. This can be a program activity, process activity, or block activity.

activity information member. A predefined data structure member associated with the operating characteristics of an activity.

API. Application Programming Interface.

applet. An application program, written in the Java programming language, that can be retrieved from a Web server and executed by a Web browser. A reference to an applet appears in the markup for a Web page, in the same way that a reference to a graphics file appears; a browser retrieves an applet in the same way that it retrieves a graphics file. For security reasons, an applet's access rights are limited in two ways: the applet cannot access the file system of the client upon which it is executing, and the applet's communication across the network is limited to the server from which it was downloaded. Contrast with *servlet*.

application programming interface. An interface provided by the MQ Workflow workflow manager that enables programs to request services from the MQ Workflow workflow manager.

asynchronous API call. A particular service of the API that allows programs to register functions with the API that are invoked when a defined event is found by MQ Workflow.

audit trail. A relational table in the database that contains an entry for each major event during execution of a process instance.

authorization. The attributes of a user's staff definition that determine the user's level of authority in MQ Workflow. The system administrator is allowed to perform all functions.

B

backward mapping. Conversion of output data created by an OS/390 legacy application into an MQSeries Workflow container. This conversion is performed by the *program execution server's program mapper*.

backward mapping definition. Part of the *MDL* which connects an interface definition and structure definition.

bend point. A point at which a connector starts, ends, or changes direction.

block activity. A composite activity that consists of a group of activities, which can be connected with control and data connectors. A block activity is used to implement a Do-Until loop; all activities within the block activity are processed until the exit condition of the block activity evaluates to true. See also *composite activity*.

Buildtime. An MQ Workflow component with a graphical user interface for creating and maintaining workflow models, administering resources, and the system network definitions.

C

cardinality. (1) An attribute of a relationship that describes the membership quantity. There are four types of cardinality: One-to-one, one-to-many, many-to-many, and many-to-one. (2) The number of rows in a database table or the number of different values in a column of a database table.

child organization. An organization within the hierarchy of administrative units of an enterprise that has a parent organization. Each child organization can have one parent organization and several child organizations. The parent is one level above in the hierarchy. Contrast with *parent organization*.

cleanup server. The MQ Workflow component that physically deletes information in the MQ Workflow Runtime database, which had only been deleted logically.

composite activity. An activity which is composed of other activities. Composite activities are block activities and bundle activities.

container API. An MQ Workflow API that allows programs executing under the control of MQ Workflow to obtain data from the input and output container of the activity and to store data in the output container of the activity.

control connector. Defines the potential flow of control between two nodes in the process. The actual flow of control is determined at run time based on the truth value of the transition conditions associated with the control connector.

coordinator. A predefined role that is automatically assigned to the person designated to coordinate a role.

CPIC. An invocation type that allows the program execution server to run an application synchronously on an IMS service. CPIC is based on IMS/APPC.

D

data connector. Defines the flow of data between containers.

data container. Storage for the input and output data of an activity or process. See *input container* and *output container*.

data mapping. Specifies, for a data connector, which fields from the associated source container are mapped to which fields in the associated target container.

data structure. A named entity that consists of a set of data structure members. Input and output containers are defined by reference to a data structure and adopt the layout of the referenced data structure type.

data structure member. One of the variables of which a data structure is composed.

default control connector. The graphical representation of a standard control connector, shown in the process diagram. Control flows along this connector if no other control path is valid.

document type definition (DTD). The rules, determined by an application, that apply SGML to the markup language of documents of a particular type. SGML provides the syntax for the markup language, and the DTD provides the vocabulary for the markup language.

domain. A set of MQ Workflow system groups which have the same meta-model, share the same staff information, and topology information. Communication between the components in the domain is via message queuing.

dynamic staff assignment. A method of assigning staff to an activity by specifying criteria such as role, organization, or level. When an activity is ready, the users who meet the selection criteria receive the activity to be worked on. See also *level*, *organization*, *process administrator*, and *role*.

E

end activity. An activity that has no outgoing control connector.

EXCI. An invocation type that allows the program execution server to run an application synchronously on a CICS service. EXCI is based on the CICS External CICS Interface provided by CICS Version 4.1 and higher to allow non-CICS applications to call programs running under CICS.

execution server. The MQ Workflow component that performs the processing of process instances at runtime.

exit condition. A logical expression that specifies whether an activity is complete.

export. An MQ Workflow utility program for retrieving information from the MQ Workflow database and making it available in MQ Workflow Definition Language (FDL) or HTML format. Contrast with *import*.

F

fixed member. A predefined data structure member that provides information about the current activity. The value of a fixed member is set by the MQ Workflow workflow manager.

(FDL) MQ Workflow Definition Language. The language used to exchange MQ Workflow information between MQ Workflow system groups. The language is used by the import and export function of MQ Workflow and contains the workflow definitions for staff, programs, data structures, and topology. This allows non-MQ Workflow components to interact with MQ Workflow. See also *export* and *import*.

FlowMark. The predecessor of MQSeries Workflow.

fork activity. An activity that is the source of multiple control connectors.

forward mapping. Conversion of MQSeries Workflow containers into a format accepted by an OS/390 legacy application. This conversion is performed by the *program execution server's program mapper*.

forward mapping definition. Part of the *MDL* which connects a structure definition and interface definition.

fully-qualified name. A qualified name that is complete; that is, one that includes all names in the

hierarchical sequence above the structure member to which the name refers, as well as the name of the member itself.

I

import. An MQ Workflow utility program that accepts information in the MQ Workflow definition language (FDL) format and places it in an MQ Workflow database. Contrast with *export*.

input container. Storage for data used as input to an activity or process. See also *source* and *data mapping*.

interface. The definition of the data structure accepted by an OS/390 CICS or IMS legacy application. This definition is used by the *program mapper* to convert data to (and from) an MQSeries Workflow program's *structure*.

interface definition. Part of the *MDL* which defines the interface used by a legacy application.

interface element. Part of an *interface* definition. An interface element has a name, a type and a cardinality. It is mapped onto a *structure* element by a *mapping rule*.

invocation exit. The DLL specified by the invocation type. The exit is based on an invocation protocol like CICS External CICS Interface, IMS/APPC or the MQSeries CICS and IMS bridges.

invocation protocol. The way the PES connects to a service like CICS or IMS in order to invoke a program on that service.

invocation type. The way the program execution server connects to a service system (like CICS or IMS) in order to invoke a program on that service. The invocation type is part of a program mapping execution request sent to the PES. To invoke a program, the PES loads the appropriate invocation exit as defined for the invocation type. Invocation types include *EXCI* and *CPIC*.

L

level. A number from 0 through 9 that is assigned to each person in an MQ Workflow database. The person who defines staff in Buildtime can assign a meaning to these numbers such as rank and experience. Level is one of the criteria that can be used to dynamically assign activities to people.

local user. Identifies a user during staff resolution whose home server is in the same system group as the originating process.

local subprocess. A subprocess that is processed in the same MQ Workflow system group as the originating process.

logical expression. An expression composed of operators and operands that, when evaluated, gives a result of true, false, or an integer. (Nonzero integers are equivalent to false.) See also *exit condition* and *transition condition*.

M

manager. A predefined role that is automatically assigned to the person who is defined as head of an organization.

mapping definition language. The language used to define *mapping definitions* for the *program mapping exit*.

mapping exit. Used by the PES to convert data between MQSeries Workflow and legacy applications. The exit is defined by a mapping type defined in the *PES directory* and in *Buildtime*. The exit is only called if mapping has been enabled in *Buildtime*.

mapping rules. Part of a *forward mapping* or *backward mapping* definition that defines the mapping between individual *interface elements* and *structure elements*. Mapping rules are defined using the *mapper definition language*.

mapping type. The name used to identify which mapping exit to use. The mapping type is defined in the *PES directory* and must match the *Buildtime* definitions for the legacy application. The mapping type provided with MQSeries Workflow for OS/390 is named **DEFAULT**.

MDL. See *mapping definition language*.

message queuing. A communication technique that uses asynchronous messages for communication between software components.

MQCICS. An invocation type that allows the program execution server to run an application asynchronously on a CICS service. The corresponding invocation exit uses the MQSeries CICS Bridge as invocation protocol.

MQIMS. An invocation type that allows the program execution server to run an application asynchronously on an IMS service. The corresponding invocation exit uses the invocation protocol MQSeries IMS Bridge.

N

navigation. Movement from a completed activity to subsequent activities in a process. The paths followed are determined by control connectors, their associated transition conditions, and by the start conditions of activities. See also *control connector*, *exit condition*, *transition condition*, and *start condition*.

node. (1) The generic name for activities within a process diagram. (2) The operating system image that hosts MQ Workflow systems.

notification. An MQ Workflow facility that can notify a designated person when a process or activity is not completed within the specified time.

notification work item. A work item that represents an activity or process notification.

O

organization. An administrative unit of an enterprise. Organization is one of the criteria that can be used to dynamically assign activities to people. See *child organization* and *parent organization*.

output container. Storage for data produced by an activity or process for use by other activities or for evaluation of conditions. See also *sink*.

P

parent organization. An organization within the hierarchy of administrative units of an enterprise that has one or more child organizations. A child is one level below its parent in the hierarchy. Contrast with child *child organization*.

parent process. A process instance that contains the process activity which started the process as a subprocess.

pattern activity. A single and simple activity in a bundle activity from which multiple instances, called pattern activity instances, are created at run time.

person (pl. people). A member of staff in an enterprise who has been defined in the MQ Workflow database.

PES. See *program execution server*.

PES directory. See *program execution server directory*.

predefined data structure member. A data structure member predefined by MQ Workflow and used for communication between user applications and MQ Workflow Runtime.

process. Synonymously used for a process model and a process instance. The actual meaning is typically derived from the context.

process activity. An activity that is part of a process model. When a process activity is executed, an instance of the process model is created and executed.

process administrator. A person who is the administrator for a particular process instance. The administrator is authorized to perform all operations on a process instance. The administrator is also the target for staff resolution and notification.

process category. An attribute that a process modeler can specify for a process model to limit the set of users who are authorized to perform functions on the appropriate process instances.

process definition. Synonym for *process model*.

process diagram. A graphical representation of a process that shows the properties of a process model.

process instance. An instance of a process to be executed in MQ Workflow Runtime.

process instance list. A set of process instances that are selected and sorted according to user-defined criteria.

process instance monitor. An MQ Workflow client component that shows the state of a particular process instance graphically.

process management. The MQ Workflow Runtime tasks associated with process instances. These consist of creating, starting, suspending, resuming, terminating, restarting, and deleting process instances.

process model. A set of processes represented in a process model. The processes are represented in graphical form in the process diagram. The process model contains the definitions for staff, programs, and data structures associated with the activities of the process. After having translated the process model into a process template, the process template can be executed over and over again. *Workflow model* and *process definition* are synonyms.

process monitor API. An application programming interface that allows applications to implement the functions of a process instance monitor.

process-relevant data. Data that is used to control the sequence of activities in a process instance.

process status. The status of a process instance.

process template. A fixed form of a process model from which process instances can be created. It is the translated form in MQ Workflow Runtime. See also *process instance*.

process template list. A set of process templates that have been selected and sorted according to user-defined criteria.

program. A computer-based application that serves as the implementation of a program activity or as a support tool. Program activities reference executable programs using the logical names associated with the programs in MQ Workflow program registrations. See also *program registration*.

program activity. An activity that is executed by a registered program. Starting this activity invokes the program. Contrast with *process activity*.

program execution agent (PEA). An MQ Workflow component that manages the implementations of program activities for a user in a LAN environment. Each instance of a program execution agent services a single user. MQSeries Workflow for OS/390 does not support program execution agents but rather employs a program execution server.

program execution server (PES). An MQ Workflow component that can manage the implementations of program activities for multiple clients. OS/390 employs a program execution server to implement CICS and IMS programs and to support mapping of data formats between MQSeries Workflow and legacy applications. Multiple instances of a program execution server are possible. In the LAN environment, program execution agents are used instead of program execution servers.

program mapping. Program mapping definitions passed and supported into the mapping database and used by the program mapper at runtime to transform data from legacy applications.

program mapping DB. Database used by the PES exit which contains program mappings imported by the program mapping import tool. Used at runtime by the exit to perform the forward and backward mapping.

program mapping exit. PES exit used to transform MQSeries Workflow for OS/390 containers into a format acceptable by legacy applications and vice versa.

program mapping import tool. Component of the MQSeries Workflow program mapping exit which reads the result of the program mapping parser and inputs the compiled program mapping definitions into the program mapping DB.

program mapping parser. Component of the MQSeries Workflow for OS/390 program mapping exit which parses the MDL and creates an intermediate file which is used by the program mapping import tool.

program registration. Registering a program in MQ Workflow so that sufficient information is available for managing the program when it is executed by MQ Workflow.

R

role. A responsibility that is defined for staff members. Role is one of the criteria that can be used to dynamically assign activities to people.

S

scheduling server. The MQ Workflow component that schedules actions based on time events, such as resuming suspended work items, or detecting overdue processes.

server. The servers that make up an MQ Workflow system are called Execution Server, Administration Server, Scheduling Server, and Cleanup Server.

servlet. An application program, written in the Java programming language, that is executed on a Web server. A reference to a servlet appears in the markup for a Web page, in the same way that a reference to a graphics file appears. The Web server executes the servlet and sends the results of the execution (if there are any) to the Web browser. Contrast with applet.

sink. The symbol that represents the output container of a process or a block activity.

source. The symbol that represents the input container of a process or a block activity.

specific resource assignment. A method of assigning resources to processes or activities by specifying their user IDs.

standard client. The MQ Workflow component, which enables creation and control of process instances, working with worklists and work items, and manipulation of personal data of the logged-on user.

start activity. An activity that has no incoming control connector.

start condition. The condition that determines whether an activity with incoming control connectors can start after all of the incoming control connectors are evaluated.

structure. The definition of the MQSeries Workflow structure passed into or out of an *activity* implementation.

structure definition. Part of the *MDL* which defines the structure used by a program activity.

structure element. Part of an *structure* definition. A structure element has a name, a type and a cardinality. It is mapped onto an *interface* element by a *mapping rule*.

subprocess. A process instance that is started by a process activity.

substitute. The person to whom an activity is automatically transferred when the person to whom the activity was originally assigned is declared as absent.

support tool. A program that end users can start from their worklists in the MQ Workflow Client to help complete an activity.

symbolic reference. A reference to a specific data item, the process name, or activity name in the description text of activities or in the command-line parameters of program registrations. Symbolic references are expressed as pairs of percent signs (%)

that enclose the fully-qualified name of a data item, or either of the keywords `_PROCESS` or `_ACTIVITY`.

system. The smallest MQ Workflow unit within an MQ Workflow domain. It consists of a set of the MQ Workflow servers.

system group. A set of MQ Workflow systems that share the same database.

system administrator. (1) A predefined role that conveys all authorizations and that can be assigned to exactly one person in an MQ Workflow system. (2) The person at a computer installation who designs, controls, and manages the use of the computer system.

T

thin client. A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a PC.

top-level process. A process instance that is not a subprocess and is started from a user's process instance list or from an application program.

transition condition. A logical expression associated with a conditional control connector. If specified, it must be true for control to flow along the associated control connector. See also *control connector*.

translate. The action that converts a process model into a Runtime process template.

trusted program. A program that has been assigned such a characteristic in the FDL (via Buildtime), which enables a PEA or PES to divulge the program ID.

U

Unicode. A system of 16 bit binary codes for text or script characters. Officially called the Unicode Worldwide Character Standard, it is a system for "the interchange, processing, and display of the written texts of the diverse languages of the modern world". Unicode is used by the Java programming language for character representation.

user-defined program execution server (UPES). A program listening on a user-defined MQSeries queue and effectively acting as a program execution server by accepting and implementing program invocation requests received from external programs in the form of XML messages.

user ID. An alphanumeric string that uniquely identifies an MQ Workflow user.

user type definition. Part of the *MDL* which defines the interface used by a user type.

user type interface. A user defined interface type. If you need to map a data type that is not supported by the default mapper type, you can define a user type, and write a type conversion program which handles the conversion of the particular data type. This must use the user type exit.

V

verify. The action that checks a process model for completeness.

W

workflow. The sequence of activities performed in accordance with the business processes of an enterprise.

Workflow Management Coalition (WfMC). A non-profit organization of vendors and users of workflow management systems. The Coalition's mission is to promote workflow standards for workflow management systems to allow interoperability between different implementations.

workflow model. Synonym for *process model*.

work item. Representation of work to be done in the context of an activity in a process instance.

work item set of a user. All work items assigned to a user.

worklist. A list of work items and notifications assigned to a user and retrieved from a workflow management system.

worklist view. List of work items selected from a work item set of a user according to filter criteria which are an attribute of a worklist. It can be sorted according to sort criteria if specified for this worklist.

Bibliography

To order any of the following publications, contact your IBM representative or IBM branch office.

MQSeries Workflow for OS/390 publications

This section lists the publications included in the MQSeries Workflow for OS/390 library.

- *MQSeries Workflow for OS/390: Customization and Administration*, SC33-7030, explains how to customize and administer an MQSeries Workflow for OS/390 system.
- *MQSeries Workflow for OS/390: Programming Guide*, SC33-7031, explains the application programming interfaces (APIs) available on OS/390, including program execution server exits.
- *MQSeries Workflow for OS/390: Messages and Codes*, SC33-7032, explains the MQSeries Workflow for OS/390 system messages.
- *MQSeries Workflow for OS/390: Program Directory*, GI10-0483, explains how to install MQSeries Workflow for OS/390.

MQSeries Workflow publications

This section lists the publications included in the MQSeries Workflow library.

- *IBM MQSeries Workflow: List of Workstation Server Processor Groups*, GH12-6357, lists the processor groups for MQSeries Workflow.
- *IBM MQSeries Workflow: Concepts and Architecture*, GH12-6285, explains the basic concepts of MQSeries Workflow. It also describes the architecture of MQSeries Workflow and how the components fit together.
- *IBM MQSeries Workflow: Getting Started with Buildtime*, SH12-6286, describes how to use Buildtime of MQSeries Workflow.
- *IBM MQSeries Workflow: Getting Started with Runtime*, SH12-6287, describes how to get started with the Client.
- *IBM MQSeries Workflow: Programming Guide*, SH12-6291, explains the application programming interfaces (APIs).

- *IBM MQSeries Workflow: Installation Guide*, SH12-6288, contains information and procedures for installing and customizing MQSeries Workflow.
- *IBM MQSeries Workflow: Administration Guide*, SH12-6289, explains how to administer an MQSeries Workflow system.

Workflow publications

- *IBM Systems Journal*, Vol. 36. No. 1, 1997 by Frank Leymann, Dieter Roller. You can also refer to the Internet:
<http://www.almaden.ibm.com/journal/sj361/leymann.html>
- *Workflow Handbook 1997*, published in association with WfMC. Edited by Peter Lawrence.

MQSeries publications

- *MQSeries System Administration*, SC33-1873, explains administration tasks related to MQSeries.
- *MQSeries Installation*, SH12-6288, discusses the installation of MQSeries.
- *MQSeries System Administration*, SC33-0807, discusses topics related to the application programming interface of MQSeries.

Other useful publications

- *MQSeries Clients*, GC22-1632
- *DB2 for OS/390 Administration Guide*, SC26-8957
- *DB2 for OS/390 SQL Reference*, SC26-8966
- *DB2 for OS/390 Application Programming and SQL Guide*, SC26-8958
- *DB2 for OS/390 Command Reference*, SC26-8960
- *DB2 for OS/390 Utility Guide and Reference*, SC26-8967
- *OS/390 MVS Planning: Workload Management*, GC28-1761
- *OS/390 MVS Programming: Workload Management Services*, GC28-1773
- *W3C Recommendation: Extensible Markup Language (XML) 1.0*, REC-xml-19980210

Licensed books

The licensed books that were declassified in OS/390 Version 2 Release 4 appear on the OS/390 Online Library Collection, SK2T-6700. The remaining licensed books for OS/390 Version 2 appear on the OS/390 Licensed Product library, LK2T-2499, in unencrypted form.

Index

A

- accessor calls
 - API calls 86
 - authorization 85
 - bool 86
 - char 107
 - date/time 87
 - default values 84
 - definition 84
 - enumeration 87
 - error handling 8
 - examples 115, 116, 117
 - integer 111
 - IsNull 110
 - lifetime of values 85
 - long 106, 111, 113
 - multi-valued 107
 - object 112
 - object valued 108, 110
 - return codes 85
 - session requirements 85
 - string 107
 - vector 21
- action calls
 - definition 121
 - error handling 8
- activating program mappings 191
- activity implementation 191
 - API calls 121
 - coding (examples) 558
 - container 124, 131, 136
 - error handling 8
 - input container 289, 293
 - output container 291, 294, 296, 298
 - passthrough 325
 - pseudo code 124, 130, 135
 - remote passthrough 370
 - return code 125, 131, 136
 - XML 153
- activity instance
 - definition 273
 - error reason 240
 - monitor, process instance 273
 - notification 327
 - overview 217
 - subprocess instance, retrieval 275
 - vector API calls 223
- activity instance notification
 - definition 277
 - delete 374
 - description, set 382
 - monitor, process instance 376
 - name, set 384
 - object identifier 277
 - overview 221
 - process instance 378
 - refresh 380
 - retrieve 278
 - start tool 280
 - transfer 386

- ActivityInstance
 - API calls 217
 - ObtainInstanceMonitor() 273
 - ObtainProcessInstanceMonitor() 273
 - SubProcessInstance() 275
- ActivityInstanceNotification
 - API calls 221
 - Delete() 374
 - ObtainProcessInstanceMonitor() 376
 - PersistentObject() 278
 - ProcessInstance() 378
 - Refresh() 380
 - SetDescription() 382
 - SetName() 384
 - StartTool() 280
 - Transfer() 386
- agent
 - API calls 223
 - overview 223
- allocation
 - copy 74
 - declaration 71
 - explicit 11, 132
 - implicit 11, 132
- API calls
 - action 273
 - activity implementation 273
 - categories 70
 - summary by class/object 215
- application
 - activity implementation 3, 7, 124, 135
 - activity implementation, Java 130
 - client 3, 7, 123, 134
 - client, Java 129
- array
 - Java 30
 - query result 20
- assignment 73
- authorization
 - accessor API calls 85
 - definitions 67
 - explicit 67
 - implicit 67
 - process administrator 67
 - system administrator 67
 - XML message 153

B

- backward mapping
 - constants 166, 171
 - definition 164, 166
 - example 167, 170, 171, 172
 - example with constants 172
 - grammar 185
 - non-default backward mapping 170
- Backwardsetting 186
- basic calls
 - definition 70
 - error handling 8
 - examples 78, 79, 80

- basic calls (*continued*)
 - return codes 70
- bibliography 571
- block instance monitor
 - definition 282
 - monitor, block activity 282
 - monitor, process instance 284
 - obtain 66
 - overview 224
 - ownership 67
 - refresh 287
- BlockInstanceMonitor
 - API calls 224
 - ObtainBlockInstanceMonitor() 282
 - ObtainProcessInstanceMonitor() 284
 - Refresh() 287
- bool definition 123

C

- C/C++
 - programming considerations 123
- CharacterInterfaceType 184
- characters 173
- check in 475
- check out 477
- CICS
 - mapping example 192, 193
 - special considerations 163
- COBOL
 - calling requirements 133
 - compiling/linking 136
 - mapping to C data types 137
 - name differences with C 138
 - programming considerations 133
 - string handling 133
 - string handling example 148
- comparison 73
- compile
 - bool, string, vector 123
 - headers 125
 - library files 125
- complete
 - data view 75
 - function 75, 84
- concepts
 - memory management 8, 11
 - object access 8
 - object management 132
 - result object 8
 - session 8
- constructor
 - copy 74
 - declaration 71
- container
 - activity implementation 121, 124, 131, 136
 - analyze structure 37
 - analyzing an element 41
 - array 31
 - array index 31
 - basic data types 30

- container (*continued*)
 - container element 31
 - data member 30
 - data structure 30
 - definition 30
 - element overview 228
 - element vector 230
 - example 31
 - exception 65
 - fixed data members 33
 - fully qualified name 31
 - input, process template 451
 - input, work item 486
 - input container 289, 293
 - leaf 31, 37
 - mapping 163
 - name in dot notation 31
 - output, work item 488
 - output container 291, 294, 296, 298
 - overview 225
 - predefined data members 32
 - read-only 289
 - read/write 289
 - return codes 65
 - structural member 31, 39
 - type 40
 - value 31, 47, 60

Container

- API calls 225
- container element 289
- definition 289
- InContainer() 289
- leaves 289
- OutContainer() 291
- RemoteInContainer() 293
- RemoteOutContainer() 294
- SetOutContainer() 296
- SetRemoteOutContainer() 298

container element

- access 47
- array 42, 46
- definition 31
- exception 65
- leaf 31, 42, 43
- name 41
- return codes 65
- structural member 42, 44
- type 31, 41
- value 54

ContainerElement

- API calls 228

control connector instance

- overview 231
- vector 232

ControlConnectorInstance

- API calls 231

conversion 174

copy

- constructor 74
- function 74

D

data access

- models 16
- pull 16
- push 16

data access (*continued*)

- view 75, 84

date/time

- overview 232

DateAndTime

- API calls 232

deallocation

- declaration 75
- function 22, 75
- vector 22

default values 84

description

- item 382
- persistent list 392
- process instance 419
- process instance list 301
- process template list 307
- worklist 312

destructor

- declaration 75

DLL options

- overview 233

DllOptions

- API calls 233

E

empty

- function 76, 84
- object 76

equal

- comparison 73
- function 73

error

- handling 8
- Java exceptions 9
- mapping errors 171
- overview 240
- reason 240
- result object 12
- return codes 9

exception, Java 241

exceptions

- Java 9

execution data 17

- overview 234

execution service

- definition 300
- log off 319
- log on 320
- overview 4, 235
- passthrough 325
- password, set 467
- process instance list 301
- process template list 306
- query, activity instance notification 327
- query, item 333
- query, process instance 347
- query, process instance list 339
- query, process instance notification 341
- query, process template 354
- query, process template list 352
- query, work item 358
- query, worklist 365
- remote passthrough 370

execution service (*continued*)

- session, begin 320
- session, end 319
- session, passthrough 325
- session, remote passthrough 370
- settings, logged on user 469
- worklist 312
- ExecutionAgent
 - API calls 233
- ExecutionData
 - API calls 234
- ExecutionService
 - API calls 235
 - CreateProcessInstanceList() 301
 - CreateProcessTemplateList() 306
 - CreateWorklist() 312
 - definition 300
 - Logoff() 319
 - Logon() 320
 - Passthrough() 325
 - Query
 - ActivityInstanceNotifications() 327
 - QueryItems() 333
 - QueryProcessInstanceLists() 339
 - QueryProcessInstanceNotifications() 341
 - QueryProcessInstances() 347
 - QueryProcessTemplateLists() 352
 - QueryProcessTemplates() 354
 - QueryWorkitems() 358
 - QueryWorklists() 365
 - Receive() 367
 - Refresh() 466
 - RemotePassthrough() 370
 - SetPassword() 467
 - TerminateReceive() 372
 - UserSettings() 469

ExeOptions

- API calls 237
- overview 237

external service options

- overview 238

ExternalOptions

- API calls 238

F

filter

- activity instance notification 327
- definition 20
- item 333
- persistent list 388, 394
- process instance 347
- process instance list 301
- process instance notification 341
- process template 354
- process template list 307
- work item 358
- worklist 312

finish

- work item 481
- work item, force 483

flat file 165

float numbers 173

float_token 177

FloatInterfaceType 184

FmcjError/FmcError

- API calls 240

- FmcjPEA
 - API calls 233
- FormToMapping 186
- forwardmapping
 - constants 166, 171
 - default forward mapping 169
 - definition 164, 166
 - example 167, 169, 172
 - example with constants 172
 - grammar 186
 - non-default forward mapping 169
- ForwardSetting 186
- fully qualified name 31
- function
 - accessor 84
 - action 121
 - activity implementation 121
 - basic 70
 - categories 70
 - client/server call 121
 - vector accessor 21

G

- Global
 - API calls 241
- global services
 - overview 241
- grammar
 - comments 177
 - example 187
 - interface definition 165
 - interface definitions
 - InterfaceCardinality 182
 - InterfaceDeclaration 181
 - InterfaceSetting 181
 - InterfaceType 182
 - interface types
 - CharacterInterfaceType 184
 - FloatInterfaceType 184
 - IntegerInterfaceType 184
 - PackedAttributeList 182
 - PackedInterfaceType 182
 - UserInterfaceType 185
 - usertype 167
 - ZonedAttributeList 183
 - ZonedInterfaceType 183
 - keywords 180
 - mapping elements 185
 - Backward mapping 185
 - Backwardsetting 186
 - FormToMapping 186
 - Forward mapping 186
 - ForwardSetting 186
 - Mapping 185
 - MappingElement 185
 - MappingRule 186
 - overview 177
 - structure definition 165
 - structure definitions
 - MemberCardinality 181
 - MemberDeclaration 180
 - MemberSetting 181
 - MemberType 181
 - StructureSetting 180
 - tokens 177
 - float_token 177

- grammar (*continued*)
 - tokens 177 (*continued*)
 - hex_digit 178
 - hex_token 178
 - identifier 178
 - int_token 179
 - packed_token 179
 - string_token 179
 - zoned_token 179
 - usertype
 - UserType 186
 - UserTypeDeclaration 187
 - UserTypeLength 187
 - UserTypeSetting 187
 - usertype definition 186

H

- handle
 - object 8
- hex_digit 178
- hex_token 178

I

- identifier 178
- implementation data
 - overview 242
- ImplementationData
 - API calls 242
- IMS
 - special considerations 163
- input container
 - activity implementation 124, 131, 136
 - process instance 407
 - process template 451
 - work item 486
- int_token 179
- integer numbers 173
- IntegerInterfaceType 184
- interface (mapping)
 - definition 164, 165, 174
 - elements 165
 - example 166, 172
 - grammar 181
 - interface element 165
 - interface element size 190
 - interface element types
 - characters 173
 - definition 173
 - float numbers 173
 - integer numbers 173
 - packed numbers 173
 - zoned numbers 173
 - interface elements 166, 168
 - interface types grammar 182
 - InterfaceCardinality 182
 - InterfaceDeclaration 181
 - InterfaceSetting 181
 - InterfaceType 182
- item
 - definition 373
 - delete 374
 - description, set 382
 - filter 333, 358
 - monitor, process instance 376
 - name 384

- item (*continued*)
 - object identifier 374
 - overview 243
 - process instance, retrieval 378
 - properties 382
 - query 333
 - refresh 380
 - sort criteria 336, 362
 - state 471
 - threshold 336, 362
 - transfer 386
 - vector 245
 - worklist 312
- Item
 - API calls 243
 - Delete() 374
 - ObtainProcessInstanceMonitor() 376
 - ProcessInstance() 378
 - Refresh() 380
 - SetDescription() 382
 - SetName() 384

J

- Java
 - API beans 128
 - applet 129
 - communication layer 127
 - CORBA agent 127
 - locator methods 127
 - programming considerations 126
 - servlet 129
- justification 185

K

- keywords 180
- kind
 - function 76

L

- log off 319
- logon
 - absence setting 322
 - default 321
 - present 321
 - session, execution server 320
 - session mode 321

M

- mapping 163, 198
 - activating program mappings 191
 - application examples 192, 193
 - array 166, 168
 - backward mapping 166, 170, 171, 172
 - Buildtime 165
 - CICS example 192, 193
 - constants 166, 168, 171, 172
 - container 164, 165, 168
 - data type mappings 175, 176
 - default mapping 164, 166, 167, 170
 - errors 171, 192
 - example 172, 187, 195, 196, 197
 - examples 194

- mapping 163, 198 (*continued*)
 - explicit mapping 166, 168, 171, 172
 - flat file 165
 - Flowmark definition language (FDL) 165
 - forward mapping 166, 169, 171, 172
 - grammar 177, 187
 - interface 166, 168
 - interface definition grammar 181
 - introduction 163
 - legacy application 163, 165, 166
 - mapper 164
 - mapping algorithm 168
 - mapping database 165
 - mapping definition elements 165
 - mapping definition language (MDL) 165, 169, 191
 - mapping rules 165
 - parser 165
 - PES 163, 165
 - program mapping 163
 - structure 166, 168
 - structure definition grammar 180
 - usertype 167
 - valid conversions 174
 - Workflow API 164
- MappingElement 185
- MappingRule 186
- MemberCardinality 181
- MemberDeclaration 180
- MemberSetting 181
- MemberType 181
- memory
 - management 8, 11
 - ownership 8
 - thread 12, 132
- message
 - overview 245
- Message
 - API calls 245
- message interface 149
- method
 - accessor 84
 - action 121
 - activity implementation 121
 - basic 70
 - categories 70
 - client/server call 121
- modules 1
- monitor 65
 - block 282
 - obtain 66
 - process instance 273, 284, 376, 410
- MQSeries message descriptor 149

N

- name
 - item 384
 - name 461
 - persistent list 388
 - process instance 421, 435, 439, 445
 - process instance list 301, 429
 - process template list 307, 459
 - syntax 384, 421, 435, 439, 445
 - worklist 312, 500

- notification
 - activity instance notification, query 327, 500
 - filter 327, 341
 - item, query 333
 - process instance notification, query 341, 505
 - sort criteria 330, 344
 - threshold 330, 344
 - worklist, create 312

O

- object
 - access 8
 - management 132
 - memory management 8
 - optional property 84
 - persistent 11, 132
 - primary property 84
 - querying (examples) 535
 - secondary property 84
 - transient 11, 132
- object identifier
 - activity instance notification 277
 - item 374
 - process instance 405
 - process instance notification 432
 - process template 434
 - work item 471
- output container
 - activity implementation 124, 131, 136
 - work item 488
- owner
 - block instance monitor 67
 - persistent list 388
 - process instance list 301, 429
 - process instance monitor 67
 - process template list 307, 459
 - transfer, item 386
 - worklist 312, 500

P

- packed numbers 173
- packed_token 179
- PackedAttributeList 182
- PackedInterfaceType 182
- parser 165
- passthrough 325, 370
- password, set 467
- persistent list
 - creating (examples) 511
 - definition 20, 388
 - delete 389
 - description 301, 307, 312
 - description, set 392
 - filter 301, 307, 312, 388
 - filter, set 394
 - name 301, 307, 312, 388
 - overview 4, 246
 - owner 301, 307, 312, 388
 - process instance 301
 - process template list 306
 - query 429, 459
 - query, process instance list 339

- persistent list (*continued*)
 - query, worklist 500, 502, 505, 507
 - querying (examples) 521
 - refresh 390
 - sort criteria 301, 303, 307, 309, 312, 316, 388
 - sort criteria, set 396
 - threshold 301, 307, 312, 388
 - threshold, set 398
 - type 301, 307, 312, 388
 - worklist 312

PersistentList

- API calls 246
- Delete() 389
- Refresh() 390
- SetDescription() 392
- SetFilter() 394
- SetSortCriteria() 396
- SetThreshold() 398

person

- absence 402
- definition 400
- overview 247
- password, set 467
- refresh 400
- settings, logged on user 469
- substitute 403

Person

- API calls 247
- Refresh() 400
- SetAbsence() 402
- SetSubstitute() 403

point

- overview 250
- vector 251

Point

- API calls 250

predefined data members 32

- _ACTIVITY 33
- _ACTIVITY_INFO.CoordinatorOfRole 35
- _ACTIVITY_INFO.Duration 37
- _ACTIVITY_INFO.Duration2 37
- _ACTIVITY_INFO.LowerLevel 36
- _ACTIVITY_INFO.MembersOfRoles 35
- _ACTIVITY_INFO.Organization 35
- _ACTIVITY_INFO.OrganizationType 35
- _ACTIVITY_INFO.People 36
- _ACTIVITY_INFO.PersonToNotify 36
- _ACTIVITY_INFO.Priority 34
- _ACTIVITY_INFO.UpperLevel 36
- _PROCESS 33
- _PROCESS_INFO.Duration 34
- _PROCESS_INFO.Organization 34
- _PROCESS_INFO.Role 33
- _PROCESS_MODEL 33
- _RC 33
- activity information 32, 34
- fixed 32, 33
- process information 32, 33

primary view

- definition 84
- IsComplete() 75

process administrator 67

- process instance
 - create 434, 439
 - definition 405
 - delete 406

- process instance (*continued*)
 - description 419
 - execute 444
 - filter 347
 - input container 407
 - monitor 255, 410
 - name 405, 421, 435, 439, 445
 - notification 341
 - object identifier 405
 - overview 251
 - persistent list, create 301
 - query 347
 - querying (examples) 535
 - refresh 414
 - restart 416
 - resume 417
 - retrieve 412
 - sort criteria 349, 350
 - start 423, 434
 - state 405
 - suspend 425
 - terminate 427
 - threshold 349
 - vector 256
- process instance list
 - creating (examples) 511
 - creation 301
 - delete 389
 - description 301
 - description, set 392
 - filter 301
 - filter, set 394
 - name 301, 429
 - overview 255
 - owner 301, 429
 - query 339, 429
 - refresh 390
 - sort criteria 301, 303
 - sort criteria, set 396
 - threshold 301
 - threshold, set 398
 - type 301, 429
 - vector 255
- process instance monitor
 - monitor, block activity 282
 - monitor, process instance 284
 - overview 65, 255
 - ownership 67
 - refresh 287
- process instance notification
 - definition 432
 - delete 374
 - description, set 382
 - monitor, process instance 376
 - name, set 384
 - object identifier 432
 - overview 256
 - process instance 378
 - refresh 380
 - retrieve 432
 - transfer 386
 - vector 256
- process template
 - create process instance 434, 439
 - definition 434
 - delete 442
 - execute process instance 444
- process template (*continued*)
 - filter 354
 - input container 451
 - name 434
 - object identifier 434
 - overview 257
 - persistent list, create 306
 - program template 455, 461
 - query 354
 - refresh 457
 - retrieve 453
 - sort criteria 356
 - start process instance 434
 - threshold 356
 - valid-from date 434
 - vector 260
- process template list
 - creation 306
 - delete 389
 - description 307
 - description, set 392
 - filter 307
 - filter, set 394
 - name 307, 459
 - overview 259
 - owner 307, 459
 - query 352, 459
 - refresh 390
 - sort criteria 307, 309
 - sort criteria, set 396
 - threshold 307
 - threshold, set 398
 - type 307, 459
 - vector 260
- ProcessInstance
 - API calls 251
 - Delete() 406
 - InContainer() 407
 - ObtainMonitor() 410
 - PersistentObject() 412
 - Refresh() 414
 - Restart() 416
 - Resume() 417
 - SetDescription() 419
 - SetName() 421
 - Start() 423
 - Suspend() 425
 - Terminate() 427
 - Transfer() 386
- ProcessInstanceList
 - API calls 255
 - Delete() 389
 - QueryProcessInstances() 429
 - Refresh() 390
 - SetDescription() 392
 - SetFilter() 394
 - SetSortCriteria() 396
 - SetThreshold() 398
- ProcessInstanceMonitor
 - API calls 255
 - ObtainBlockInstanceMonitor() 282
 - ObtainProcessInstanceMonitor() 284
 - Refresh() 287
- ProcessInstanceNotification
 - API calls 256
 - Delete() 374
 - ObtainProcessInstanceMonitor() 376
- ProcessInstanceNotification (*continued*)
 - PersistentObject() 432
 - ProcessInstance() 378
 - Refresh() 380
 - SetDescription() 382
 - SetName() 384
 - Transfer() 386
- ProcessTemplate
 - API calls 257
 - CreateAndStartInstance() 434
 - CreateInstance() 439
 - Delete() 442
 - ExecuteProcessInstance() 444
 - InitialInContainer() 451
 - PersistentObject() 453
 - ProgramTemplate() 455
 - Refresh() 457
- ProcessTemplateList
 - API calls 259
 - Delete() 389
 - QueryProcessTemplates() 459
 - Refresh() 390
 - SetDescription() 392
 - SetFilter() 394
 - SetSortCriteria() 396
 - SetThreshold() 398
- profile
 - defaults 300
 - user 300
 - workstation 300
- program data
 - overview 260
- program execution agent (PEA)
 - overview 233
- program execution management calls
 - error handling 8
 - program execution server 121
- program execution server (PES)
 - program mapping 163
- program execution server exits
 - common interfaces 199
 - enabling 204, 213
 - program execution server 198
 - program invocation 205
 - program mapping 201
 - special considerations 201
- program template
 - definition 461
 - execute 462
 - overview 261
- ProgramData
 - API calls 260
- programming
 - activity implementation 3, 7
 - client 3, 7
 - concepts 1
 - mapping 163
 - prerequisites 3
- ProgramTemplate
 - API calls 261
 - Execute() 462
- property
 - optional 84
 - primary 84
 - secondary 84
- protocol
 - asynchronous 16

protocol (*continued*)
supported 16
synchronous 16
unsolicited 16, 321

pull data 16

push
data, receive 367
enable 17
kind of information 17
receive 17
session mode 321
terminate receive 372
push data 16

Q

query
activity instance notification 327
array of objects 20
data 19
item 333
process instance 347
process instance list 339
process instance list, process instances 429
process instance notification 341
process template list 352
process template list, process templates 459
vector of objects 20
work item 358
worklist 365, 500
worklist, items 502
worklist, process instance notification 505
worklist, work item 507

R

read-only container
activity implementation, input container 289, 293
definition 289
overview 263
work item, input container 486
read/write container
activity implementation, output container 291, 294, 296, 298
definition 289
overview 263
process instance, input container 407
process template, input container 451
work item, output container 488
ReadOnlyContainer
API calls 263
ReadWriteContainer
API calls 263
receive data 367
remote
terminate, subprocess 427
restart
work item 492
work item, force 485
Result
API calls 265
result object
definition 12

result object (*continued*)
error information 8
information contained 12
overview 265
return code
access API calls 85
action API calls 8
activity implementation 125, 131, 136
basic API calls 70
error handling 8
list of 9

S

secondary view
definition 84
IsComplete() 75
service
execution service 300
overview 265
password, set 467
settings, logged on user 469
Service
API calls 265
definition 466
Refresh() 466
SetPassword() 467
UserSettings() 469
session
absence setting 322
accessor API calls 85
begin 300, 320, 325, 370
default 321
end 300, 319
establish 19
establish, execution server 300
log off 300, 319
log on 300, 320
mode 321
overview 19
passthrough 325
present 321
remote passthrough 370
requirement 8
unified logon 321
sort criteria
activity instance notification 330
definition 20
item 336, 362
persistent list 388, 396
process instance 349, 350
process instance list 301, 303
process instance notification 344
process template 356
process template list 307, 309
work item 362
worklist 312, 316
start
process instance 423, 434
support tool 280
work item 494, 495
state
item 471
process instance 405
work item 471
string
vector 266

string definition 123
string_token 179
StringVector
vector 266
structure (mapping)
definition 164, 165
elements 165
example 165, 172
grammar 180
MemberCardinality 181
MemberDeclaration 180
MemberSetting 181
MemberType 181
structure definitions 165
structure elements 165, 166, 168
StructureSetting 180
subprocess
resume 417
suspend 425
terminate 427
suspension
process instance 425
symbol layout
overview 267
SymbolLayout
API calls 267
synchronous protocol 16
syntax rules
description, item 382
description, persistent list 392
description, process instance 419
name, item 384
name, process instance 421, 435, 439, 445
XML DTD 157
system
execution server 300
system administrator 67
system group
execution server 300

T

thread 12, 132
threshold
activity instance notifications 330
definition 20
items 336, 362
persistent list 388, 398
process instance list 301
process instance notifications 344
process instances 349
process template list 307
process templates 356
worklist 312
transient object 8
type
persistent list 388
private, persistent list 388
private, process instance list 429
private, process template list 459
private, worklist 500
process instance list 301, 429
process template list 307, 459
public, persistent list 388
public, process instance list 429
public, process template list 459

- type (*continued*)
 - public, worklist 500
 - worklist 312, 500

U

- unified logon 321
- unsolicited information 16
- user
 - default values, profile 300
 - password, set 467
 - settings 469
- user-defined program execution server
 - definition 155
- UserInterfaceType 185
- usertype
 - creation of DLL 190
 - definition 167, 174, 190
 - example 167, 190
 - exit interface 189
 - grammar 186
 - introduction 188
 - usertype exit 164, 167
 - UserTypeDeclaration 187
 - UserTypeLength 187
 - UserTypeSetting 187

V

- valid conversions 174
- vector
 - accessor function 21
 - activity instance notifications 222
 - activity instances 223
 - container elements 230
 - control connector instances 232
 - deallocate 22
 - definition 123
 - examples 24, 26
 - first element 22
 - items 245
 - next element 23
 - overview 266
 - points 251
 - process instance lists 255
 - process instance notifications 256
 - process instances 256
 - process template lists 260
 - process templates 260
 - query result 20
 - return codes 21
 - size 23
 - work items 270
 - worklist 271
- view
 - data view 84
 - IsComplete() 75
 - primary 84
 - secondary 84

W

- work item
 - cancel checkout 473
 - check in 475
 - check out 477

- work item (*continued*)
 - definition 471
 - delete 374
 - description, set 382
 - error reason 240
 - finish 481
 - finish, force 483
 - input container 486
 - monitor, process instance 376
 - name, set 384
 - object identifier 471
 - output container 488
 - overview 268
 - persistent list, create 312
 - process instance 378
 - query 333, 358
 - query, worklist 507
 - querying (examples) 548
 - refresh 380
 - restart 492
 - restart, force 485
 - retrieve 490
 - start 494, 495
 - state 471
 - terminate 497
 - transfer 386
 - vector 270

- workflow model 1

- Workitem
 - API calls 268
 - CancelCheckout() 473
 - CheckIn() 475
 - Checkout() 477
 - Delete() 374
 - Finish() 481
 - ForceFinish() 483
 - ForceRestart() 485
 - InContainer() 486
 - ObtainProcessInstanceMonitor() 376
 - OutContainer() 488
 - PersistentObject() 490
 - ProcessInstance() 378
 - Refresh() 380
 - Restart() 492
 - SetDescription() 382
 - SetName() 384
 - Start() 494, 495
 - Terminate() 497
 - Transfer() 386

- worklist
 - creation 312
 - definition 499
 - delete 389
 - description 312
 - description, set 392
 - filter 312
 - filter, set 394
 - name 312, 500
 - overview 270
 - owner 312, 500
 - query 365, 502, 505, 507
 - query, activity instance
 - notification 500
 - querying (examples) 521
 - refresh 390
 - sort criteria 312, 316
 - sort criteria, set 396

- worklist (*continued*)
 - threshold 312
 - threshold, set 398
 - type 312, 500
 - vector 271
- Worklist
 - API calls 270
 - Delete() 389
 - QueryActivityInstance
 - Notifications() 500
 - QueryItems() 502
 - QueryProcessInstance
 - Notifications() 505
 - QueryWorkitems() 507
 - Refresh() 390
 - SetDescription() 392
 - SetFilter() 394
 - SetSortCriteria() 396
 - SetThreshold() 398
- workstation profile
 - default values 300

X

- XML
 - activity implementation 153
 - authentication 153
 - authorization 153, 156
 - code page support 152
 - completion message 155
 - concepts 8
 - container 151
 - correlation 151
 - DTD 157
 - example 156
 - example, container 151
 - example, execute process
 - instance 152
 - input queue 153
 - message content 150
 - message format 149, 157
 - message header 150
 - message interface 149
 - sending requests 152
 - user context data 151
 - user-defined program execution
 - server 155

Z

- zoned numbers 173
- zoned_token 179
- ZonedAttributeList 183
- ZonedInterfaceType 183

Readers' Comments — We'd Like to Hear from You

IBM MQSeries Workflow for OS/390
Programming Guide

Publication No. SC33-7031-02

Overall, how satisfied are you with the information in this book?

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|----------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Overall satisfaction | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

How satisfied are you that the information in this book is:

| | Very Satisfied | Satisfied | Neutral | Dissatisfied | Very Dissatisfied |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Accurate | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Complete | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Easy to find | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Easy to understand | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Well organized | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Applicable to your tasks | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? Yes No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.



Fold and Tape

Please do not staple

Fold and Tape



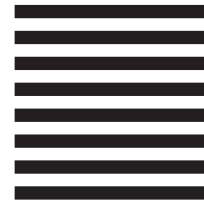
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Deutschland Entwicklung GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany



Fold and Tape

Please do not staple

Fold and Tape



Program Number: 5655-A96



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-7031-02



Spine information:



IBM MQSeries Workflow for
OS/390

MQSeries Workflow for OS/390 Programming Guide

Version 3 Release 2