IBM MQSeries Workflow

Programming Guide

Version 3.3

SH12-6291-06

IBM MQSeries Workflow

Programming Guide

Version 3.3

SH12-6291-06

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix B. Notices" on page 839.

Seventh Edition (March 2001)

This edition applies to version 3, release 3 of IBM MQSeries Workflow (product number 5697-FM3) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SH12-6291-05.

Contents

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. xii
How to read the syntax diagrams xiii

Summary of Changes xv

Summary of deprecated API calls xix

Part 1. Programming concepts . . . 1

Chapter 1. Understanding the programming concept 3
The role of the programmer in modeling a process 3

Chapter 2. Programming interfaces 5

Chapter 3. Prerequisites for programming language API. 7

Chapter 4. Building an MQ Workflow application 9
Overview 9
Concepts of the programming language API 9
Concepts of the XML message interface . . 10
Handling errors. 10
List of return codes 11
List of ActiveX GUI Control exceptions . . 14
Debugging considerations 15
Prerequisites 15
Creating a test database 15
Debugging a client application. 15
Debugging an activity implementation or support tool 16

Chapter 5. Client/server communication and data access models 19
Synchronous client/server communication . . 19
Asynchronous client/server communication 19
The push data access model. 20
Receiving information. 21

Chapter 6. An MQ Workflow session . . . 25

Chapter 7. Using an authentication exit . . 27
Coding an authentication exit 27
Activating an authentication exit 30
Error handling 31

Chapter 8. Querying data 33
Persistent lists 33
Using filters, sort criteria, and thresholds . . 33
Handling collections 34
C-language vectors. 35
Return codes. 35
FmcjXxxVectorDeallocate. 35
FmcjXxxVectorFirstElement 36
FmcjXxxVectorNextElement 36
FmcjXxxVectorSize 37
C-language examples 37
ActiveX arrays 40
Exceptions 40
Add. 40
GetAt 41
GetSize 41
RemoveAll 42
RemoveAt 42
SetAt 42
Events 43
Java arrays 43

Chapter 9. Handling containers 45
Data structure/container type 45
Data member/container element 45
The XML message interface 47
Predefined data members 47
Fixed data members 48
Process information data members . . . 49
Activity information data members . . . 50
Determining the structure of an unknown container 54
Determining the leaves 54
Determining the structural members . . . 55
Determining the type 56
Analyzing a container element 57
Determining the name or type of a container element 57

Determining the structural properties of a container element 58
Determining the leaves of a container element 59
Determining the structural members of a container element 60
Determining the elements of an array . . 61
Accessing a known container element . . . 63
Accessing a value of a container 64
Accessing a value of a container element . . 69
Setting a value of a container 72
Return codes/FmcException 77

Chapter 10. Monitoring a process instance 79
Obtaining an process instance monitor . . . 79
Ownership of monitors 81

Chapter 11. Authorization considerations 83

Chapter 12. Stateless server support . . . 87

Chapter 13. Types of API calls 93
Basic API calls 93
Return codes 93
Allocation 94
Assignment 97
Comparison/equality 97
Conversion 98
Copy 99
Deallocation 99
IsComplete() 100
IsEmpty() 101
Kind() 101
C-language Example: using basic functions 102
C++ Example: using basic methods . . . 104
Accessor/mutator API calls 106
Primary/secondary properties 107
Return codes 107
Accessing a value of type bool 109
Accessing a value of type date/time . . 110
Accessing an enumerated value 111
Accessing a value of type integer . . . 136
Accessing a value of type string 137
Accessing a multi-valued property . . . 138
Accessing an object valued property . . 139
Accessing a pointer valued property . . 140
Determining whether an optional property is set 142
Setting a value of type integer 142
Setting a value of type string 144
Setting an object valued property . . . 145
Updating an object 145
Action API calls 150
Activity implementation API calls 151
Accessing general information 152
Dynamic link libraries 154
Program execution management API calls 155

Part 2. The C and C++ APIs . . . 157

Chapter 14. An MQ Workflow client application 159

Chapter 15. An MQ Workflow activity implementation or support tool 161

Chapter 16. Compiling and linking . . . 163
Supported compilers 165
C++ prerequisite header files 165
Sample compile statements 165

Chapter 17. Memory management . . . 167

Chapter 18. The result object 169

Part 3. ActiveX controls 173

Chapter 19. Component overview. . . . 175
Functional overview 176
Workflow Control overview 176
How to work with an ExecutionService . . 177
How to work with lists 177
ProcessTemplateList Control overview . . . 177
ProcessInstanceList Control overview . . . 177
Worklist Control overview 178
Monitor Control overview 178

Chapter 20. An MQ Workflow client application 179

Chapter 21. An MQ Workflow activity implementation or support tool 181

Part 4. The JAVA API 183

Chapter 22. Threading considerations for the Java CORBA Agent 185

JNDI locator policy 186
OSA, IOR, and COS locator policies. . . . 186
RMI locator policy 187
Microsoft JVM/Internet Explorer V4/V5 and RMI 187

Chapter 23. An MQ Workflow client application 189

Chapter 24. An MQ Workflow activity implementation 191
The Java High Performance Bridge 192
Setup on Windows platforms 193
Setup on UNIX 195
Programming considerations 196

Chapter 25. Compiling 199
JNDI locator policy 199

Chapter 26. How to use the MQ Workflow Java API from within IBM VisualAge for Java 201
Running the MQ Workflow Java CORBA Agent inside the WebSphere Test Environment 201

Chapter 27. Troubleshooting 203

Chapter 28. Object management 205
Garbage Collection when using Java API classes 206

Part 5. The XML message interface 207

Chapter 29. The MQ Workflow XML message 209
Relevant MQSeries Message Descriptor (MQMD) fields 209
The application data 210
The MQ Workflow XML message header 211
Container data 211
Execute process instance example . . . 213
Code page support 214

Chapter 30. Sending requests to MQ Workflow 215
Supported functions 215
XML input queue 216
Authentication and authorization 216
Create and start a process instance example 216

Chapter 31. Invoking an activity implementation 219
User-defined program execution server (UPES) 220
Messages sent to a UPES 221
Authorization 224
Synchronous invocation example. 224
UPES response example. 225

Chapter 32. Error Handling 227
MQ Workflow XML message life cycle . . . 227
General error processing 227
Sending a response 230
Detailed error processing 230
Wrong message format in the MQMD . . 231
Wrong message name or XML document not well formed 232
Message processing errors 232
Errors when returning a response . . . 233
Backout count exceeded. 234
The GeneralError message 234

Chapter 33. The MQ Workflow XML message format 237

Part 6. Using the MQ Workflow APIs 245

Chapter 34. Using the MQ Workflow Runtime API 247
Overview of the Runtime API. 247
API classes 251
API calls per class 255
ActivityInstance 255
ActivityInstanceArray 260
ActivityInstanceNotification 260
ActivityInstanceNotificationArray . . . 263
ActivityInstanceNotificationVector . . . 263
ActivityInstanceVector 263
Agent. 264
Container 266
ContainerArray 270
ContainerElement. 270
ContainerElementArray. 274
ContainerElementVector. 274
ControlConnectorArray 274

ControlConnectorInstance 275
ControlConnectorInstanceVector 276
DateAndTime/ FmcjDateTime/
FmcjCDateTime 276
DllOptions 278
ExecutionAgent/FmcjPEA 279
ExecutionData 280
ExecutionService 281
ExecutionServiceArray 286
ExeOptions 287
ExternalOptions 288
FmcError/FmcjError 291
FmcException 291
Global 292
ImplementationData 293
ImplementationDataVector 294
InstanceMonitor 294
Item 296
ItemVector 299
Message 299
PersistentList 300
Person 301
Point 306
PointArray 306
PointVector 307
ProcessInstance 307
ProcessInstanceList 312
ProcessInstanceListArray 312
ProcessInstanceListVector 313
ProcessInstanceNotification 313
ProcessInstanceNotificationArray . . . 314
ProcessInstanceNotificationVector . . . 315
ProcessInstanceVector 315
ProcessTemplate 315
ProcessTemplateList 318
ProcessTemplateListArray 319
ProcessTemplateListVector 319
ProcessTemplateVector 320
ProgramData 320
ProgramTemplate 322
ReadOnlyContainer 324
ReadOnlyContainerHolder 325
ReadWriteContainer 325
Result 327
Service 328
StringArray 329
StringVector 330
SymbolLayout 331
Workitem 332
WorkitemArray 335
WorkitemVector 335
Worklist 335
WorklistArray 336
WorklistVector 337

Part 7. API action and activity implementation calls 339

Chapter 35. Activity instance actions . . 341
ForceFinish() 341
ForceRestart() 344
InContainer() 347
ObtainProcessMonitor()/ObtainInstanceMonitor() 349
OutContainer() 352
Refresh() 354
SubProcessInstance() 356
Terminate() 359

Chapter 36. Activity instance notification actions 363
ActivityInstance() 363
StartTool() 366

Chapter 37. Container activity implementation API calls. 369
InContainer() 369
OutContainer() 371
RemoteInContainer() 373
RemoteOutContainer() 376
SetOutContainer() 378
SetRemoteOutContainer() 380

Chapter 38. Execution service actions 383
CreateProcessInstanceList() 384
CreateProcessTemplateList() 391
CreateWorklist() 398
Logoff() 407
Logon() 409
Passthrough() 416
PEAShutDown() 418
PEAStartUp() 420
QueryActivityInstanceNotifications() . . . 423
QueryItems() 431
QueryProcessInstanceLists() 437
QueryProcessInstanceNotifications() . . . 440
QueryProcessInstances() 446
QueryProcessTemplateLists() 451
QueryProcessTemplates() 454
QueryWorkitems() 459

QueryWorklists() 466
Receive() 469
RemotePassthrough() 472
SetPersonAbsent() 475
TerminateReceive() 477

Chapter 39. Instance monitor actions . . 481
ObtainInstanceMonitor()/
ObtainBlockMonitor()/
ObtainProcessMonitor() 481
Refresh() 484

Chapter 40. Item actions 489
Delete() 489
ObtainProcessMonitor()/ObtainInstanceMonitor() 492
ProcessInstance() 495
Refresh() 498
SetDescription() 500
SetName() 503
Transfer() 505

Chapter 41. Persistent list actions . . . 509
Delete() 509
Refresh 512
SetDescription() 514
SetFilter() 517
SetSortCriteria() 520
SetThreshold() 522

Chapter 42. Person actions 527
Refresh() 527
SetAbsence() 529
SetSubstitute() 531

Chapter 43. Process instance actions . . 535
Delete() 535
InContainer() 538
ObtainProcessMonitor() 540
OutContainer() 543
Refresh() 545
Restart() 548
Resume() 550
SetDescription() 552
SetName() 555
Start() 557
Suspend() 560
Terminate() 563

Chapter 44. Process instance list actions 567
QueryProcessInstances() 567

Chapter 45. Process template actions . . 571
CreateAndStartInstance() 571
CreateInstance() 579
Delete() 582
ExecuteProcessInstance() 585
InitialInContainer() 595
ProgramTemplate() 597
Refresh() 600

Chapter 46. Process template list actions 603
QueryProcessTemplates() 603

Chapter 47. Program template actions 607
Execute() 607

Chapter 48. Service actions 613
Refresh() 613
SetPassword() 615
UserSettings() 617

Chapter 49. Work item actions 621
ActivityInstance() 624
CancelCheckOut() 627
CheckIn() 629
CheckOut() 632
Finish() 638
ForceFinish() 640
ForceRestart() 643
InContainer() 646
OutContainer() 648
Restart() 650
Start() 652
StartTool() 655
Terminate() 657

Chapter 50. Work listactions 661
QueryActivityInstanceNotifications() . . . 661
QueryItems() 665
QueryProcessInstanceNotifications() . . . 668
QueryWorkitems() 671

Part 8. Working with ActiveX controls 675

Chapter 51. The ExecutionService Control 677

Chapter 52. The list controls 679

Chapter 53. The Monitor Control 681

Chapter 54. Typical scenario of ActiveX Control methods 683

Chapter 55. MQWorkflowCtrl 685
Methods 685
ConfigurationID 685
Connect 685
ContainerArray 685
CurrentDateAndTime 685
DateAndTime 686
Disconnect 686
ExecutionServiceArray 686
NewActivityInstance 687
NewActivityInstanceNotification 687
NewContainer 687
NewExecutionService 687
NewInstanceMonitor 688
NewPerson 688
NewProcessInstance 688
NewProcessInstanceList 688
NewProcessInstanceNotification 689
NewProcessTemplate 689
NewProcessTemplateList 689
NewProgramData 689
NewProgramTemplate 690
NewWorkitem 690
NewWorklist 690
ProgramID 690
RemoteUserID 691
SetConfigurationID 691
StringArray 691
UserID 691

Chapter 56. ContainerCtrl 693
Properties 693
Methods 693
Container 693
ProgramID 693
RemoteUserID 693
UserID 694
Events 694
Error 694

Chapter 57. Methods supported by all GUI controls 695
AboutBox 695
ReadUserSettings 695
RemoveGUI 695
SetHelpFile 696
ShowContextMenu 696
WriteUserSettings 696

Chapter 58. Methods supported by all list controls 699
ConnectGUI 699
ContextMenuDelete 699
ContextMenuListProperties 699
ContextMenuListSettings 700
ContextMenuListRefresh 700
ContextMenuProperties 700
ContextMenuViewIcon 701
ContextMenuViewList 701
ContextMenuViewReport 701
ContextMenuViewSmallIcon 701
FindFirst 701
FindNext 702
GetItemAt 703
GetItemCount 703

Chapter 59. Events triggered by all GUI controls 705
Click 705
DblClick 705
KeyPress 705

Chapter 60. Events triggered by all non-monitor GUI controls 707
Error 707
KeyDown 707
KeyUp 708
MouseDown 708
MouseMove 709
MouseUp 709

Chapter 61. Events triggered by all list controls 711
ViewChanged 711

Chapter 62. ExecutionServiceCtrl 713
Properties 713
Methods 713
ConnectGUI 713
ContextMenuDeleteProcInstList 714
ContextMenuDeleteProcTempList . . . 714
ContextMenuDeleteWorklist 714
ContextMenuLogoff 715
ContextMenuLogon 715
ContextMenuLogonDialog 716
ContextMenuNewProcInstList 716
ContextMenuNewProcTempList 716

ContextMenuNewWorklist 717
ContextMenuProperties 717
ContextMenuRefresh 717
ContextMenuRefreshProcInstLists . . . 717
ContextMenuRefreshProcInstances . . . 717
ContextMenuRefreshProcTempLists . . . 718
ContextMenuRefreshProcTemplates . . . 718
ContextMenuRefreshWorkitems 718
ContextMenuRefreshWorklists 718
ContextMenuUserInformation. 719
Events 719
ItemCollapsed 719
ItemCollapsing. 719
ItemExpanded 720
ItemExpanding 720
SelChanged 721
SelChanging 721

Chapter 63. ProcessTemplateListCtrl . . 723
Properties 723
Methods 725
ContextMenuCreateInstance 725
RefreshProcessTemplateList 725
Events 725

Chapter 64. ProcessInstanceListCtrl. . . 727
Properties 727
Methods 729
ContextMenuRestart 729
ContextMenuResume 730
ContextMenuResumeDeep 730
ContextMenuStart. 730
ContextMenuSuspend 731
ContextMenuSuspendDeep 731
ContextMenuTerminate 731
RefreshProcessInstanceList 731
Events 732

Chapter 65. WorklistCtrl 733
Properties 733
Methods 736
ContextMenuFinish 736
ContextMenuForceFinish 736
ContextMenuForceRestart 737
ContextMenuRestart 737
ContextMenuSelectAll 737
ContextMenuStart. 737
ContextMenuStartTool 738
ContextMenuTransfer 738
PushOption 738
RefreshWorklist 739
SetPushOption 739
Events 740
ActivityInstanceNotificationChanged . . 740
ProcessInstanceNotificationChanged . . 740
WorkitemChanged 741
Starting 741

Chapter 66. MonitorCtrl 743
Properties 743
Methods 743
ActivityProperties() 743
ConnectGUI 743
ControlConnectorProperties 744
OpenMonitor 744
Refresh 744
Events 745
AfterRefreshing 745
BeforeRefreshing 745
BlockActivityClick 745
BlockActivityDoubleClick 746
ControlConnectorClick 746
ControlConnectorDoubleClick. 746
DoActivityEnter 747
DoControlConnectorEnter 747
DoRefresh 748
DoShowContextMenu 748
Error 748
MonitorOpen 749
ProcessActivityClick 749
ProcessActivityDoubleClick 749
ProgramActivityClick 750
ProgramActivityDoubleClick 750

Part 9. Examples and scenarios 753

Chapter 67. Scenarios. 755

Chapter 68. Examples 757

Chapter 69. How to create persistent lists 759
Create a process instance list (ActiveX). . . 759
Create a process instance list (C-language) 760
Create a process instance list (C++) 762
Create a process instance list (Java) 764

Chapter 70. How to query persistent lists 769
Query worklists (ActiveX) 770
Query worklists (C-language) 771

Query worklists (C++) 774
Query worklists (Java) 776

Chapter 71. How to query a set of objects 781
Query process instances from a process instance list (ActiveX) 782
Query process instances (C-language) . . . 783
Query process instances (C++) 784
Query process instances (Java) 786
Query work items from a worklist (ActiveX) 791
Query work items from a worklist (C-language) 792
Query work items from a worklist (C++) . . 794
Query work items from a worklist (Java) . . 796

Chapter 72. An activity implementation 801
Programming an executable (C-language) 801
Programming an executable (C++) 802
Programming an executable (Java) 804

Part 10. Appendixes 825

Appendix A. FlowMark Version 2 compatibility mode 827
Deviations from FlowMark Version 2 . . . 828
FlowMark Version 2 C-language programs 831
Running an existing application program 831
FlowMark Version 2 Visual Basic programs 832
Running an existing application program 832
FlowMark Version 2 C++ programs 832
Running an existing application program 832
Using MQ Workflow Version 3 methods 832

Appendix B. Notices 839
Trademarks 841

Glossary 843

Bibliography 849
MQSeries Workflow publications. 849
Related publications 849

Index 851

About this book

This book describes how to use the IBM MQSeries Workflow Runtime Application Programming Interfaces (hereafter called MQ Workflow APIs) and also how to invoke API requests by passing messages in Extensible Markup Language (XML). The first part of the book describes the concepts underlying the APIs while the remainder of the book provides a reference for the API action calls. The book also describes the MQ Workflow predefined data structures and how to debug applications running under the control of MQ Workflow.

Who should read this book

This book is intended for programmers who design and implement programs using an MQ Workflow API and who may participate in designing a workflow model with IBM MQSeries Workflow. It assumes that readers are experienced programmers and that they understand the concepts of modeling processes.

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 849.

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: swsdid@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

The first part of this book gives an overview of the various MQ Workflow APIs. It describes all concepts common to the Version 3 APIs and introduces the APIs supported.

- "Part 1. Programming concepts" on page 1 describes the concepts underlying all MQ Workflow APIs. It groups the API calls according to their behavior and describes basic and accessor methods in a generic way.
- "Part 2. The C and C++ APIs" on page 157 describes the concepts specific to the C and C++ APIs and states how application programs can be compiled and linked.
- "Part 3. ActiveX controls" on page 173 provides for an overview on the ActiveX Controls.
- "Part 4. The JAVA API" on page 183 provides for an overview on the Java API.
- "Part 5. The XML message interface" on page 207 describes the concepts in order to use XML to request an action from the MQ Workflow execution server. It furthermore describes how a program is invoked by the execution server by means of XML.

"Part 6. Using the MQ Workflow APIs" on page 245 provides for an overview on the functionality supported by the MQ Workflow Runtime. All API calls supported by the MQ Workflow APIs are summarized on a per-object basis.

The next part of this book provides for a reference manual.

- "Part 7. API action and activity implementation calls" on page 339 describes the MQ Workflow APIs 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 MQ Workflow execution service. All action, activity implementation, and program execution management API calls are described on a per-object basis. See "Chapter 13. Types of API calls" on page 93 for the description of the basic and accessor methods.
- "Part 8. Working with ActiveX controls" on page 675 describes the methods and events supported by the ActiveX Controls.

"Part 9. Examples and scenarios" on page 753 provides some examples showing how to use the APIs.

"Appendix A. FlowMark Version 2 compatibility mode" on page 827 describes how to run a FlowMark Version 2 program and states MQ Workflow Version 3 deviations from Version 2. It describes how to change a Version 2 C++ program in order to become a Version 3 program.

Note: The FlowMark Version 2 interface will no longer be supported with the next release or version.

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:

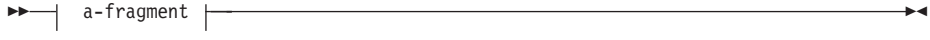
- Read the syntax diagrams from left to right, from top to bottom, following the main path of the line.

  The ▸▸— symbol indicates the beginning of a statement.

  The —▸ symbol indicates that the statement syntax is continued on the next line.

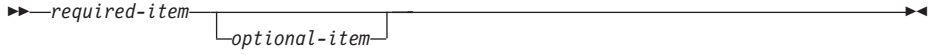
  The ▸— symbol indicates that a statement is continued from the previous line.

  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.



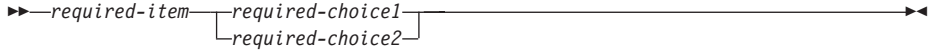
- Required items appear on the horizontal line, the main path.

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

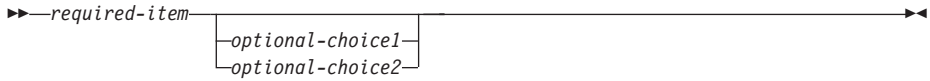


- If you can choose from one or more items, they appear vertically, in a stack.

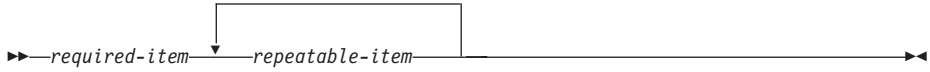
  If you must choose one of the items, one item of the stack appears on the main path.



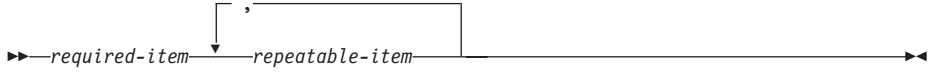
  If choosing one of the items is optional, the entire stack appears below the main path.



- 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.

Summary of Changes

Changes to this document for IBM MQSeries Workflow Version 3.3 are:

- API extensions are added to support stateless server implementations, that is:
  - The execution service exposes new API calls to retrieve a session representation and to recreate sessions, that is, SessionID() and SetSessionContext() are added.
  - New API calls are added so that all major objects can be externally represented by some string, that is, ProcessInstanceListPersistentOid(), ProcessTemplateListPersistentOid(), WorklistPersistentOid(), InstanceMonitorPersistentOid(), ReadOnlyContainerAsStream(), ReadWriteContainerAsStream(), ProgramDataAsStream(), and ProgramTemplateAsStream() are added.
  - The execution service exposes new API calls to recreate objects from their OID or from a stream, that is, PersistentActivityInstance(), PersistentActivityInstanceNotification(), PersistentInstanceMonitor(), PersistentPerson(), PersistentProcessInstance(), PersistentProcessInstanceList(), PersistentProcessInstanceNotification(), PersistentProcessTemplate(), PersistentProcessTemplateList(), PersistentWorkitem(), PersistentWorklist(), ProgramDataFromStream(), ProgramTemplateFromStream(), ReadOnlyContainerFromStream(), and ReadWriteContainerFromStream() are added.
- A new instance monitor is added. The instance monitor state can be passed between applications - see above - and nested monitors can be deleted without caring for the nesting level. Appropriate API calls are added to access the new monitor. Using the existing monitor is deprecated, that is, support will be removed in a later release or version.
- API extensions are added to support process repair actions, that is, ActivityInstanceForceFinish() and ActivityInstanceForceRestart() are added. These API calls allow to pass containers. The appropriate work item actions now allow to pass containers.
- The execution service supports a new way to log on with user credentials, which are passed to a user-provided authentication exit.
- A new activity instance and work item state *Expired* is supported, which can be used in queries for work items. The ExpirationTime() can be queried.
- The activity instance exposes a new API call Refresh().
- The process instance exposes a new API call to retrieve the OID of the associated process template, PersistentOidOfProcessTemplate().
- The container size can be 4 MB.

- Conversion between read/write and read-only containers is added.
- The Java API supports the ExecuteProcessInstance() method.
- Beside performance, the distinction between primary and secondary attributes is removed. That is, an object is automatically refreshed from the server when an attribute not yet available in the API is read.
- The audit mode can be more detailed, which is expressed by the *Filter* enumeration.
- An audit record can be sent to a user-defined MQSeries application. The XML DTD is extended appropriately.
- When a work item is terminated or expires, an XML message is sent to a UPES. The UPES must at least have version 3.3.0.
- Support for Windows Me is added.
- Support for OS/2(R) is removed.
- Support for Windows 95 is removed.

Changes to this document for IBM MQSeries Workflow Version 3.2.2 are:

- The execution service API calls to query work items or to create a worklist allow for the specification of a new filter criterion CREATION_TIME.
- The execution service exposes a new API call SetPersonAbsent().
- The container exposes a new API call SetStringCcsid().
- The activity instance exposes new API calls InContainer(), OutContainer(), PersistentObject(), and Terminate().
- The process instance exposes a new API call OutContainer().
- The work item and notifications expose new API calls PersistentOidOfProcessInstance().
- The work item and activity instance notification expose new API calls PersistentOidOfActivityInstance() and ActivityInstance().
- An activity implementation allows for the retrieval of the associated activity instance object identifier, that is, the program execution agent exposes new API calls PersistentOidOfActivityInstance() and RemotePersistentOidOfActivityInstance().
- The maximum priority of an activity instance and thus the maximum priority of a work item or activity instance notification can be 999.
- Support for Windows 2000 is added.

Changes to this document for IBM MQSeries Workflow Version 3.2.1 are:

- XML message interface support is added to create and start respectively execute a process instance. Additionally, an activity implementation can be started by the MQ Workflow execution server on a user-defined program execution server.

- The process template InContainer() API call has been renamed to InitialInContainer(). The usage of InContainer() is deprecated.
- The process template exposes a new API call ProgramTemplate().
- A new class respectively new functions are added to describe a program template. A program template supports a API call Execute() in a synchronous and an asynchronous flavor.
- Program data exposes new accessor API calls ExecutionMode(), ExecutionUser(), ProgramTrusted().
- The execution service provides for new allocation means which allow to specifiy the system group only. This is to exploit the IBM MQSeries clustering capabilities.
- The execution service API calls to query process instances or to create a process instance list allow for the specification of a new filter attribute START_TIME.

Changes to this document for IBM MQSeries Workflow Version 3.2 are:

- Support for HP-UX and Sun Solaris is now included.
- JAVA support is added.
- ActiveX supports process instance monitoring.
- The specification of a configuration identifier is supported.
- The execution service exposes a new action API call Refresh().
- The process template exposes a new action API call ExecuteProcessInstance() in a synchronous and an asynchronous flavour. This also means that an asynchronous communication protocol is added.
- The process instance exposes a new action API call Restart().
- The work item and the activity instance notification expose new action API calls StartTool(). The work item exposes a new API call CancelCheckOut().
- Restrictions are removed from the work item Restart(), ForceFinish(), and ForceRestart() API calls; a work item implemented by a *process* can now also be restarted or finished.

Changes to this document for IBM MQSeries Workflow Version 3.1.2 are:

- The item object exposes a new action API call Delete().
- Item changes are pushed to a present client. This function applies to work items, activity instance notifications, and process instance notifications, and is supported in the C-language and C++ APIs.
- Process instance monitor support is added in the C-language and C++ APIs.

Changes to this document for IBM MQSeries Workflow Version 3.1.1 are:

- Support for OS/2(R) is now included.

- The person object exposes new action API calls Refresh(), SetAbsence(), and SetSubstitute().
- The process template exposes a new action API call Delete().
- The IsTerminatedOnError() accessor API call on the process template as well as on the process instance is removed.
- A new object FmcjError is added to describe the reason why a work item is in state InError. The work item as well as an activity instance notification return the error reason.
- FmcjWorkitem::Checkout() returns a new program definition, the definition of an external service. A new object FmcjExternalOptions is added to allow querying the properties of an external service.
- The work item exposes a new action API call Terminate().
- ActiveX support is added.
- Version 2 REXX support for OS/2 is added.
- Version 2 Lotus Notes support is added.
- Version 2 C++ Logon() now offers as an option to consider using the Version 3 session mode. Refer to "Deviations from FlowMark Version 2" on page 828 for more information.

Summary of deprecated API calls

- Using the FmcjBlockInstanceMonitor and the FmcjProcessInstanceMonitor respectively functions starting with FmcjBlockInstanceMonitor or FmcjProcessInstanceMonitor is deprecated. Use the FmcjInstanceMonitor instead.
- Using the COS, OSA, RMI, and IOR policies in the Java API is deprecated.

The following table states the API calls, which are deprecated, and the new API calls to be used instead. Deprecated API calls should not be used; they will be removed in a future release or version of the API.

Deprecated API call Class/Function::method/function	**API call to be used Class/Function::method/function**
ActivityInstance:: ObtainProcessInstanceMonitor()	ActivityInstance::ObtainProcessMonitor()
ActivityInstance::PersistentObject()	ExecutionService::PersistentActivityInstance()
ActivityInstanceNotification::Expired()	There is a notification and the work item state is not *Ready*
ActivityInstanceNotification:: PersistentObject()	ExecutionService:: PersistentActivityInstanceNotification()
ActivityInstanceNotification:: StartOverdue()	There is a notification and the work item state is *Ready*
ActivityInstanceNotification:: ObtainProcessInstanceMonitor()	ActivityInstanceNotification:: ObtainProcessMonitor()
Item::ObtainProcessInstanceMonitor()	Item::ObtainProcessMonitor()
ProcessInstance::ObtainMonitor()	ProcessInstance::ObtainProcessMonitor()
ProcessInstance::PersistentObject()	ExecutionService::PersistentProcessInstance
ProcessInstanceNotification:: ObtainProcessInstanceMonitor()	ProcessInstanceNotification:: ObtainProcessMonitor()
ProcessInstanceNotification:: PersistentObject()	ExecutionService:: PersistentProcessInstanceNotification()
ProcessTemplate::InContainer()	ProcessTemplate::InitialInContainer()
ProcessTemplate::PersistentObject()	ExecutionService:: PersistentProcessTemplate()
Workitem:: ObtainProcessInstanceMonitor()	Workitem::ObtainProcessMonitor()
Workitem::PersistentObject()	ExecutionService::PersistentWorkitem()

Part 1. Programming concepts

This part provides you with a general introduction to the programming concepts of MQ Workflow.

Chapter 1. Understanding the programming concept

This chapter introduces the concept of workflow modeling as it relates to the design of application programs for use with IBM MQSeries Workflow, hereafter referred to as MQ Workflow.

MQ 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.

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

Chapter 2. Programming interfaces

MQ 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 issues 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 some server.

The MQ 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:
- MQ Workflow C-language API
- MQ Workflow C++ language API
- MQ Workflow ActiveX Controls and OLE Objects
- MQ Workflow Java API
- MQ Workflow XML message interface

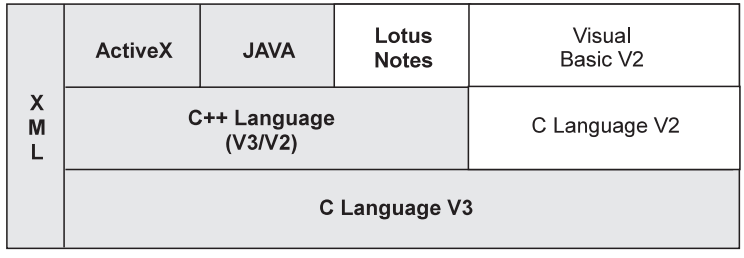


Figure 1. MQ Workflow Client APIs

The basic interfaces for requesting Runtime services from MQ Workflow are a C-language API and an XML message interface. Access can be gained to the C-language functions from all languages that support C calls - see "Chapter 16. Compiling and linking" on page 163 for more information. A C++ language API is provided on top of the C-language API. Since the C++ API is a small layer of inline methods, that is, delivered as source code, access

can be gained from all popular C++ compilers. The ActiveX and Java APIs are implemented on top of the C++ layer. MQ Workflow uses the XML 1.0 standard as document description language. Besides the Version 3 APIs, FlowMark Version 2 C-language, C++, VisualBasic, and REXX APIs are supported.

Note: The FlowMark Version 2 API is no longer supported in the next release.

The MQ Workflow APIs provide 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 MQ Workflow server
- To process container data associated with an activity implementation

Chapter 3. Prerequisites for programming language API

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

- The MQ Workflow Development Kit must be installed on the machine where you are developing your applications.
- A compiler of one of the supported languages is installed and configured.

Refer "Part 2. The C and C++ APIs" on page 157, "Part 3. ActiveX controls" on page 173, and "Part 4. The JAVA API" on page 183 for more information.

Chapter 4. Building an MQ Workflow application

Overview

There are essentially two different tasks which you can address by using the MQ Workflow application programming interface (API):

- You can write your own client application instead of using the MQ Workflow provided GUIs (Graphical User Interfaces) or command line interfaces. For example, you may want to:
  - Control the MQ Workflow functionality provided to your user.
  - Present the MQ Workflow functionality in a way that your user is accustomed to.
  - Run selected MQ Workflow tasks without user intervention.
- You can write a program that implements an activity or support tool in your workflow process model.

These two kinds of programs usually contain specific parts which are discussed in chapters "An MQ Workflow client application" and "An MQ Workflow activity implementation or support tool". See the respective chapters per language.

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

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 when they were queried from a server. In the C-language API, 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 MQ 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 programming language) 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 MQ Workflow.

In C and C++, 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 chapter "Chapter 18. The result object" on page 169 for detailed information on the result object.

Objects are managed by the application programmer but object memory is owned by the MQ Workflow API. The application programmer determines the lifetime of transient objects by using allocate, or query, and deallocate mechanisms. The MQ 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 MQ Workflow and the execution server response.

Handling errors

All action, activity implementation, program execution management API calls, or messages show whether or not the call has been successfully executed by returning a so-called *return code* as their return value. 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 11). 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 now or in future. That is, if you are not only asking whether the return code is different from FMC_OK, but, if you are checking return codes explicitly, then you should always care for unexpected errors. For example, if you are coding an *if* statement, then you should also code an *else* statement. If you are coding a *switch* statement, then you should also code the *default* case.

In addition to the return code, a so-called *result object* can be accessed in C and C++ which describes the result of the call in more detail - see "Chapter 18. The result object" on page 169.

Although the result object is set by each API call, querying its contents can be of special importance for basic and accessor API calls. Basic and accessor API calls do not return any value or return the value queried as their return value. It can happen during application development that a wrong handle or a buffer too small to hold a character value is specified. Additional errors can occur when a not yet available attribute is automatically read from the server. 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 or exceptions that are issued by the MQ Workflow APIs; it is strongly adviced to use the symbolic names instead of the integer values.

Note: When the result object states an FMC_ERROR_COMMUNICATION, that error is accompanied by three parameters: the failing action, the reason code for the failure, and the failing object.

Table 1. List of return codes

Numeric value	Symbolic value
0	FMC_OK
1	FMC_ERROR
10	FMC_ERROR_USERID_UNKNOWN
11	FMC_ERROR_ALREADY_LOGGED_ON
12	FMC_ERROR_PASSWORD
13	FMC_ERROR_COMMUNICATION
14	FMC_ERROR_TIMEOUT
15	FMC_ERROR_INVALID_CODE_PAGE
16	FMC_ERROR_INVALID_CHAR
100	FMC_ERROR_INTERNAL
101	FMC_ERROR_SERVER
102	FMC_ERROR_UNKNOWN
103	FMC_ERROR_MESSAGE_FORMAT
104	FMC_ERROR_MESSAGE_DATA
105	FMC_ERROR_RESOURCE
106	FMC_ERROR_NOT_LOGGED_ON
107	FMC_ERROR_NEW_OWNER_NOT_FOUND
108	FMC_ERROR_NO_OLD_OWNER
109	FMC_ERROR_OLD_OWNER_ABSENT
110	FMC_ERROR_NEW_OWNER_ABSENT
111	FMC_ERROR_ALREADY_STARTED
112	FMC_ERROR_MEMBER_NOT_FOUND

Table 1. List of return codes (continued)

Numeric value	Symbolic value
113	FMC_ERROR_MEMBER_NOT_SET
114	FMC_ERROR_WRONG_TYPE
115	FMC_ERROR_MEMBER_CANNOT_BE_SET
116	FMC_ERROR_MEMBER_INVALID
117	FMC_ERROR_FORMAT
118	FMC_ERROR_DOES_NOT_EXIST
119	FMC_ERROR_NOT_AUTHORIZED
120	FMC_ERROR_WRONG_STATE
121	FMC_ERROR_NOT_UNIQUE
122	FMC_ERROR_EMPTY
123	FMC_ERROR_NO_MANUAL_EXIT
124	FMC_ERROR_PROFILE
125	FMC_ERROR_INVALID_FILTER
126	FMC_ERROR_PROGRAM_EXECUTION
127	FMC_ERROR_PROTOCOL
128	FMC_ERROR_TOOL_FUNCTION
129	FMC_ERROR_INVALID_TOOL
130	FMC_ERROR_INVALID_HANDLE
131	FMC_ERROR_NOT_EMPTY
132	FMC_ERROR_INVALID_USER
133	FMC_ERROR_OWNER_ALREADY_ASSIGNED
134	FMC_ERROR_INVALID_NAME
135	FMC_ERROR_INVALID_PROGRAMID
136	FMC_ERROR_SIZE_EXCEEDED
137	FMC_ERROR_INVALID_TEMPLATE_NAME
138	FMC_ERROR_INFINITE_RECURSION
139	FMC_ERROR_SUB_PROC_MEMBER_NOT_SET
140	FMC_ERROR_PROCESS_TEMPLATE_NOT_FOUND
406	FMC_ERROR_WRONG_ACT_IMPL_KIND
500	FMC_ERROR_NON_LOCAL_USER
501	FMC_ERROR_WRONG_KIND
502	FMC_ERROR_INVALID_ACTIVITY
503	FMC_ERROR_CHECKOUT_NOT_POSSIBLE
504	FMC_ERROR_BACK_LEVEL_VERSION
505	FMC_ERROR_NEWER_VERSION
506	FMC_ERROR_INVALID_CORRELATION_ID
507	FMC_ERROR_NOT_ALLOWED
508	FMC_ERROR_BACK_LEVEL_OBJECT
509	FMC_ERROR_INVALID_CONTAINER
510	FMC_ERROR_UNEXPECTED_CONTAINER
511	FMC_ERROR_NO_PROGRAM_FOR_PLATFORM
512	FMC_ERROR_LOGON_DENIED
513	FMC_ERROR_AUTHENTICATION

Table 1. List of return codes (continued)

Numeric value	Symbolic value
800	FMC_ERROR_BUFFER
801	FMC_ERROR_INVALID_SESSION
802	FMC_ERROR_INVALID_TIME
804	FMC_ERROR_NO_MORE_DATA
805	FMC_ERROR_INVALID_OID
807	FMC_ERROR_INVALID_THRESHOLD
808	FMC_ERROR_INVALID_SORT
810	FMC_ERROR_INVALID_DESCRIPTION
811	FMC_ERROR_INVALID_INVOCATION_TYPE
812	FMC_ERROR_OWNER_NOT_FOUND
813	FMC_ERROR_INVALID_LIST_TYPE
814	FMC_ERROR_INVALID_RESULT_HANDLE
815	FMC_ERROR_MESSAGE_CATALOG
816	FMC_ERROR_INVALID_SPECIFICATION
817	FMC_ERROR_QRY_RESULT_TOO_LARGE
818	FMC_ERROR_NO_VERSION_2_FILTER
819	FMC_ERROR_INVALID_USER_CONTEXT
820	FMC_ERROR_MESSAGE_STRING
821	FMC_ERROR_MESSAGE_SIZE_EXCEEDED
822	FMC_ERROR_INVALID_STREAM
900	FMC_ERROR_NO_SYS_ADMIN
901	FMC_ERROR_INVALID_SESSION_MODE
902	FMC_ERROR_PROGRAM_UNDEFINED
904	FMC_ERROR_PEA_NOT_LOCAL
905	FMC_ERROR_INVALID_ABSENCE_SPEC
1000	FMC_ERROR_NOT_SUPPORTED
1012	FMC_ERROR_PROGRAM_NOT_DEFINED
1014	FMC_ERROR_PEA_NOT_REACHABLE
1015	FMC_ERROR_INVALID_PEA_FROM_CTNR
1016	FMC_ERROR_INVALID_PEA_FROM_MODEL
1017	FMC_ERROR_INVALID_SYSTEM_FROM_CTNR
1018	FMC_ERROR_INVALID_SYSTEM_FROM_MODEL
1019	FMC_ERROR_SUB_PROC_TERMINATED_BY_ERROR
1020	FMC_ERROR_NO_PEA_FOUND_FOR_AUTO_START
1021	FMC_ERROR_NO_CTNR_ACCESS
1022	FMC_ERROR_INVALID_CONFIGURATION_ID
1023	FMC_ERROR_MIGRATION_OF_RUNNING_PROGRAM
1024	FMC_ERROR_MIGRATION_OF_CHECKEDOUT_SUSPENDED
1025	FMC_ERROR_MIGRATION_NO_SUBPROCESS
1100	FMC_ERROR_XML_DOCUMENT_INVALID
1101	FMC_ERROR_NO_MQSWF_DOCUMENT
1102	FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED
1103	FMC_ERROR_XML_WRONG_DATA_STRUCTURE

Table 1. List of return codes (continued)

Numeric value	Symbolic value
1104	FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND
1105	FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE
1106	FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED
1107	FMC_ERROR_XML_DOCUMENT_FORMAT
1108	FMC_ERROR_XML_PARAMETER_INCORRECT
1109	FMC_ERROR_XML_PARAMETER_SIGNATURE_INCORRECT
1110	FMC_ERROR_XML_INVALID_ELEMENT
1111	FMC_ERROR_XML_INCORRECT_PARAMETER
1150	FMC_ERROR_XML_PARSER_NOT_INSTALLED
2000	FMC_ERROR_INVALID_QUEUE_SCOPE
32013	FMC_ERROR_USER_SUPPORT_MISMATCH
32014	FMC_ERROR_SUPPORT_MODE_MISMATCH
32015	FMC_ERROR_IMPLEMENTATION_SUPPORT_MISMATCH
32202	FMC_ERROR_USER_NOT_AUTHORIZED
32203	FMC_ERROR_LOCAL_USER_REQUIRED
32204	FMC_ERROR_EXIT_ERROR

List of ActiveX GUI Control exceptions

The following list shows the numeric values of exceptions that are issued by the MQ Workflow ActiveX GUI Controls; it is strongly adviced to use the symbolic names instead of the integer values:

Table 2. List of ActiveX exceptions

Numeric value	Symbolic value
1500	FMC_METHOD_EXCEPTION
1501	FMC_WRONG_INDEX
1502	FMC_MEMORY_EXCEPTION
1503	FMC_ERROR_PARAMETER
1504	FMC_OLE_EXCEPTION
1505	FMC_OLE_DISPATCH_EXCEPTION
1506	FMC_USER_EXCEPTION
1507	FMC_OBJECT_NOT_VALID
1508	FMC_OBJECT_STILL_VALID
1509	FMC_GUI_ALREADY_CONNECTED
1510	FMC_GUI_NOT_CONNECTED
1511	FMC_WRONG_CONTAINER_TYPE
1512	FMC_UNKNOWN_ITEM
1513	FMC_SET_CONTAINER_VALUE
1514	FMC_RECURSION_ERROR

Debugging considerations

Prerequisites

Debugging an MQ Workflow application that uses the programming language interface assumes that the appropriate environment is established. This means that:

- MQ Workflow DLLs are accessible. This is automatically guaranteed by a standard MQ Workflow installation.
- A test database has been created that reflects your debugging requirements (see "Creating a test database").
- The MQ Workflow servers are running on the server machine so that tests can be executed.
- You are able to connect to the required server. This can be checked with the MQ Workflow provided configuration checker *fmczchk* (refer to *IBM MQSeries Workflow: Installation Guide*).
- If you want to debug activity implementations, then the MQ Workflow Program Execution Agent must have been started for the user who gets assigned the work item.

  Note: Programs that implement activities of a process model must be registered for use with the MQ Workflow workflow manager. Ensure that the program you want to debug is registered for the selected operating system and that it can be found with the registered name and path information.

Creating a test database

In order to create a test database, you do not only need to create the database as such but you also need to:

- Add topology data (see the *IBM MQSeries Workflow: Installation Guide* on how to bootstrap your database).
- Add test data (see the *IBM MQSeries Workflow: Getting Started with Buildtime* and the chapter "Using the Runtime export and import utility").

Debugging a client application

To test your client application, start it under the control of your favorite debugger. If your application is multi-threaded, it is your responsibility to synchronize the threads properly.

Note: You can also consider to use MQ Workflow's tracing facility to get detailed information on MQ Workflow actions or the configuration checker tool for problem determination.

Debugging an activity implementation or support tool

Activity implementations and support tools run under the control of the MQ Workflow program execution agent and therefore need some special attention so that debugging becomes possible.

As with FlowMark Version 2, there is the option to change your FDL and register your debugger as the program implementation. This is the option you can use for Java.

For C, C++, and ActiveX, MQ Workflow supports using an unchanged FDL. It provides for two environment variables to enable debugging. *FMC_PEA_DEBUGGER_NAME* serves to specify the name of your debugger. You can either specify the full path and file name of your debugger or make the debugger accessible through your PATH statement. If you then set *FMC_PEA_DEBUG_ACT_IMPL* to "YES", the program execution agent starts the named debugger instead of the activity implementation. For example:

FMC_PEA_DEBUGGER_NAME = IDEBUG.EXE

FMC_PEA_DEBUG_ACT_IMPL = YES

The program execution agent starts your debugger in a separate operating system process with an appropriate environment. Since the process environment of the debugger process is set by the program execution agent and inherited by the activity implementation, your activity implementation is still known to the program execution agent and authorized to issue API requests.

When an executable is to be debugged, the program execution agent provides the name of the activity implementation and its parameters to the called debugger.

When a dynamic link library is to be debugged, the program execution agent provides the name of a program that loads your DLL and the activity implementation parameters to the called debugger.

Note: Your DLL must have been registered to run in **fenced mode**.

The MQ Workflow program that loads your DLL in fenced mode is *FMCXDLL.EXE*. It provides you with two functions, *FmcDebugDllV2* and *FmcDebugDllV3*. *FmcDebugDllV3* serves to debug MQ Workflow Version 3 DLLs and *FmcDebugDllV2* serves to debug FlowMark Version 2 compatible DLLs.

They call the entry point of your Version 2 or Version 3 DLL. If you set a breakpoint on these functions, the debugger stops before the entry point of your DLL is called and you can step into your activity implementation.

Note: The Microsoft debugger msdev.exe cannot process these entry points. Nevertheless, if you must debug your DLL, add the following to your code

```
#if defined(_MSC_VER) && defined(_DEBUG)
DebugBreak();
#endif
```

When the program execution agent starts msdev.exe, then run FMCXDLL.EXE in the msdev window. The DebugBreak() statement in your code enables the debugger to start debugging your DLL. Note that DebugBreak() only works under the control of msdev.exe and creates an unhandled exception otherwise.

When debugging a fenced DLL, be aware of the following:

1. There can exist specific problems that only apply to unfenced DLLs and that do not show up during debugging of fenced DLLs. For instance, consider the case that you run multiple instances of your (reentrant) DLL in parallel. In unfenced mode, your DLL is loaded only once and runs in the context of the program execution agent in multiple threads. In fenced mode, your DLL runs in the context of multiple FMCXDLL processes. As the data segment of a DLL is unique per process but shared between threads of a single process, you may not encounter effects seen without debugger.
2. If your DLL or the entry point in your DLL cannot be found, the debugger window will not show up and the state of your activity implementation will become *InError*. You can use MQ Workflow's trace facility to determine such problems.
3. Ensure that your DLL uses the supported (standard) calling convention and signature; MQ Workflow has defined the *FMC_APIENTRY* calling convention (see file fmcjcglo.h). If your DLL is registered as a FLowMark Version 2 compatible DLL, it gets passed two parameters, the execution session identification (called program identification in Version 3) and a pointer to additional parameters. MQ Workflow Version 3 DLLs only receive the additional parameters argument since the Version 3 program execution agent can determine the program identification on its own.

Chapter 5. Client/server communication and data access models

When you request actions from an MQ 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.
- Use an asynchronous protocol to ask for an action and to view the result at a later point in time.

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 MQ 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 way 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 MQ 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 asynchronously issue an action, then you do, however, receive an acknowledgement telling whether MQ Workflow accepted the request or not. You can also receive a correlation identification which you can use in order to receive a specific response. You can specifiy 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 get informed whether the request is accepted.
- When you specified 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 way of communication is only supported in C++ and the C-language. All message-based requests are executed asynchronously.

The push data access model

Receiving unsolicited information pushed by an MQ 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 and the XML message interface.

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

1. The server must be asked for sending data. This means that:
   - The settings of the considered 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 MQ 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/mutator API calls" on page 106 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 on 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 MQ Workflow execution server sends informations to 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 MQ Workflow execution server sends informations to 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 MQ Workflow execution server sends an information to user N when he/she is currently logged on as present. The owner of the work item as the requester of the action does not get any additional information.

Notes:

1. Filtering and sorting is left to the application. No indication about affected worklists is pushed to the client.
2. The ActiveX API provides for a worklist *Push* processing option that controls whether pushed information is to be placed on that list. If set, any item information is put on the list whether it respects filtering or not. It is put at the begin of the list, that is, does not respect sorting.

Receiving information

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

Note: Receiving any asynchronous response, that is, not waiting for a specific response identified by its correlation ID, or receiving pushed data, becomes possible in a different application process when you retrieve the session ID and attach to that session in the other process (SetSessionContext()).

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. 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 inform 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 stops, however, pushing information when logoff has been executed. When you did not send a TerminateReceive() to the receiving application thread, then 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(R) expiration mechanism applies in order to get rid of such "dead" messages. The expiration time of client messages can be configured for MQ Workflow.

When receiving data, a correlation identification can be specified to indicate which information is to be read. If it is not specified or pointing to FMCJ_NO_CORRELID, then any asynchronous response or pushed data arriving for the session is received; note that the correlation identification is set as the result of a successful asynchronous request.

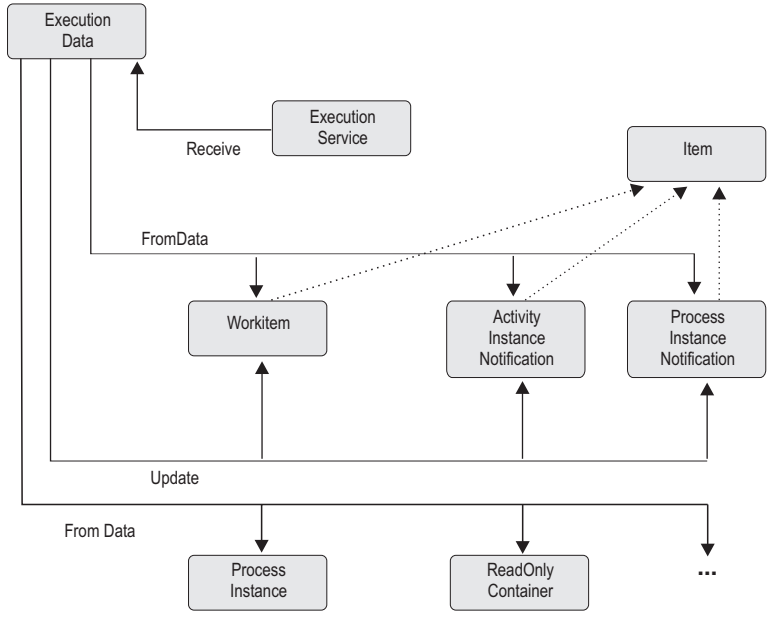


Figure 2. Handling data sent by an MQ Workflow server. Legend: --➔ Inheritance (C++); ⟶ provides for access

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 occurrs, the error description can be examined.

Note: ActiveX uses the event mechanism in order to inform an application that data has been changed. See "Events" on page 740.

Chapter 6. An MQ Workflow session

In order to communicate with an MQ Workflow server, a session must have been established between the user and that server. The server is either identified explicitly (system group or 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 is up and running on the selected system because the administration server manages sessions and checks the authentication of the user. It additionally cares for any severe errors to be 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.

Although threads are not explicitly supported by MQ Workflow (objects are not threadsafe), MQ Workflow does not prevent you from using threads. A session can span multiple threads. You have to care, however, for object synchronization. And, in all languages except Java, you should use the Connect() and Disconnect() API calls on each thread so that API resources are managed correctly.

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 a present session mode, activity instances which are started automatically

can be scheduled on your behalf and you can receive information pushed by an MQ 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: MQ 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.

Chapter 7. Using an authentication exit

Especially in Web-based environments, product-independent infrastructures handling authentication and authorization of users become more common-place. Not only user directories like LDAP are exploited but also public-key infrastructures are used more widely. Especially in EJB environments credential-based authentication is the rule, not the exception.

To support such environments, MQ Workflow provides a means to authenticate users by some third-party authentication scheme instead of by MQ Workflow itself. Third-party authentication is supported by a so-called *authentication exit*, which has to be provided by the user and called by MQ Workflow. An authentication exit can use any authentication service like a simple file, a database, directory services, and so on.

An MQ Workflow client initiates third-party authentication by calling a variant of the Logon() API that allows to specify credentials and optionally a user name. When called, the MQ Workflow administration server invokes the authentication exit passing the information received. It is then the responsibility of the authentication exit to verify authentication based on the information passed, and to map the information to an MQ Workflow user ID. The MQ Workflow user ID has to be returned to the administration server, which checks whether the user ID is a registered MQ Workflow user ID; any other user ID will be rejected. If the user ID is valid, the administration server grants a session for that user without any further verification. The authentication exit and the MQ Workflow administration server form a trusted environment.

By implementing an authentication exit, you are able to use your existing user authentication service. You can use your established user IDs for authentication and map them to MQ Workflow user IDs.

Check the *MQ Workflow API Programming Examples* for authentication exit examples.

Coding an authentication exit

Authentication exits are supported in Java and the C-language environments.

Note: HP-UX supports authentication exits written in the C-language only.

An authentication exit has a defined interface, to which every user-written exit must conform. For the C-language, the interface is described by the header file *fmcaexit.h* installed in the *Api* subdirectory of your MQ Workflow installation.

The C-language authentication exit must provide an Init() function, which is called when the exit is needed the first time, that is, during MQ Workflow administration server startup and a DeInit() function, which is called when the exit is unloaded, that is, when the administration server is shut down.

Usually the Init() function does all the initialization needed for the exit. When initialization is not needed, provide an empty implementation, that is, return FMC_EXIT_OK, that is, 0.

DeInit() normally deallocates and frees resources allocated during Init(). If DeInit() is not needed, provide an empty implementation, that is, return FMC_EXIT_OK, that is, 0.

Actual authentication is done by the C-language Authenticate() function or Java authenticate() method, which have to be implemented to make logging on based on user credentials possible. Consider that user authentication can be performance critical since there can be phases when many users log on in parallel, for example, in the morning or after lunch. The MQ Workflow administration server handles one logon request at a time.

Notes:

1. Whenever you modify your authentication exit, you must shut down and restart the administration server in order to make your changes effective.
2. An authentication exit is called in the context of an MQ Workflow transaction. Consequently, commit and rollback calls must never be issued to prevent prematurely ending that transaction.

C-language signatures

```
long FMC_APIENTRY Init(   void ** exitHandle,
                          char *  initializationParameter,
                          long    initializationParameterLength,
                          char *  errorIdBuffer,
                          long *  errorIdBufferLength,
                          char *  descriptionBuffer,
                          long *  descriptionBufferLength )

long FMC_APIENTRY DeInit( void ** exitHandle,
                          char *  errorIdBuffer,
                          long *  errorIdBufferLength,
                          char *  descriptionBuffer,
                          long *  descriptionBufferLength )

long FMC_APIENTRY Authenticate(
                          FmcjServerAuthExitAuthenticate * exitParms )
```

Java signatures

```
public abstract int authenticate( Hashtable exitParms )
```

Parameters

All parameters but the *exitParms* are reserved for future use. The *exitParms* specify information exchanged between the MQ Workflow administration server and the authentication exit. See the fmcaexit.h header file or the Java authentication exit example for more detailed descriptions.

exitParameterListEyecatcher
Input. An eyecatcher (*FMCAXHRP*) to identify the list of parameters.

exitParameterListVersion
Input. The version of the parameter list.

exitParameterListLength
Input. In the C-language, denotes the size of the parameter list structure.

version Input. The version number of the MQ Workflow client.

release Input. The release number of the MQ Workflow client.

modlevel Input. The modification level of the MQ Workflow client.

userCredentials
Input/Output. The buffer containing the user credentials.

userCredentialsSize
Input/Output. In the C-language, denotes the size of the user credentials contained in the user credentials buffer.

userCredentialsBufferSize
Input. In the C-language, denotes the size of the buffer containing the user credentials.

exitCorrelID Input/Output. A correlation ID which must be returned but not changed.

phaseNumber Input/Output. A number maintained by the MQ Workflow administration server; must be returned but not changed. Reserved for future use.

exitResult Output. The result of authentication: either LogonAccepted, LogonDenied, or Error. Other values are reserved for future use.

reasonCode Output. Specifies a user-defined reason of an Error result.

userName Input/Output. A user name known by the authentication exit.

mqwfUserID Output. The MQ Workflow user ID returned by a successful authentication. Remember that an MQ Workflow user ID must contain only uppercase characters.

Return type
long/int The result of calling this API call.

- FMC_EXIT_OK, that is, 0, signals that the function executed successfully.
- FMC_EXIT_RECOVERABLE_ERROR, that is, a 1, signals that the function executed unsuccessfully but recoverable. The logon request returns FMC_ERROR_NOT_AUTHORIZED.
- FMC_EXIT_NONRECOVERABLE_ERROR, that is, a 2, signals that the function executed unsuccessfully and nonrecoverable. When the C-language Authenticate() or the Java authenticate() returns that error, the adminstration server shuts down. A C-language Init() or DeInit() returning that error is treated similar to a recoverable error.

Activating an authentication exit

Set the *RTAuthenticationExitTypeServer* variable in the configuration profile of the server to either "C" or "JAVA", according to your implementation.

For example, if you want to set in your configuration *FMC* that an authentication exit written in the C-language is to be loaded, issue:

```
fmczchk -c inst:m,RTAuthenticationExitTypeServer,C -y FMC
```

For example, if you want to set in your configuration *FMC* that an authentication exit written in Java is to be loaded, issue:

```
fmczchk -c inst:m,RTAuthenticationExitTypeServer,JAVA -y FMC
```

Depending on the *RTAuthenticationExitTypeServer* setting, the appropriate authentication exit is loaded when the MQ Workflow administration server starts. In the C-language, the authentication exit Init() function is called.

The authentication exit is called by the administration server whenever you issue a logon with credentials. Note that a logon request specifying an MQ Workflow user ID (and a password) is executed by the administration server without involving the authentication exit.

When the MQ Workflow administration server shuts down, the authentication exit is unloaded. In the C-language, the authentication exit DeInit() function is called before the authentication exit is unloaded.

To ensure adequate performance, the C-language authentication exit must be a load library so that it can be loaded into the address space of the administration server. Depending on your operating system environment, it has to be named *fmcaexit.dll* or *libfmcaexit.so* and to be placed in the PATH/LIBPATH visible by the MQ Workflow administration server.

To adequately support Java, a Java Virtual Machine (JVM) is instantiated in the address space of the administration server, which is then reused for all logon requests passing credentials. The Java class implementing the authentication exit has to be named com.ibm.workflow.java.exit.Authentication and to be placed into the CLASSPATH visible by the MQ Workflow administration server.

Take special care that no broken load library DLL or Java class is put into the administration server's path or CLASSPATH. A broken authentication exit can cause a shutdown of the administration server and a wrong authentication exit can undermine your security concept.

Error handling

Logging on with user credentials can be rejected because:

- The authentication exit is not found; the required functions (entry points in the DLL) are not found; the called functions do not return FMC_EXIT_OK[1]. In all these situations, FMC_ERROR_NOT_AUTHORIZED is returned by the logon with credentials request.
- The user ID returned by the authentication exit is not a registered MQ Workflow user ID; results in FMC_ERROR_USERID_UNKNOWN.
- The authentication exit denies authentication; results in FMC_ERROR_LOGON_DENIED.
- The authentication exit reports an error because of the stated reason; results in FMC_ERROR_AUTHENTICATION and the first parameter (of the result object) set to the reason code.

1. FMC_EXIT_NONRECEOVERABLE_ERROR shuts down the administration server.

Chapter 8. Querying data

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

- A query via a service object, which returns all objects authorized for. The number of objects returned to the client can be restricted by a filter and a threshold. (Not supported in ActiveX.)
- A query using a persistent list definition, which returns all objects qualifying through the list definition.
- 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 especially 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 specifying criteria which 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"
```

Note that 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 the C or C++ language or an array of objects in the ActiveX and Java language.

A vector is provided by the caller and filled by the MQ 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 the C-language API.[2] 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 the C-language in order to access a vector assumed to already exist.

In the C-language, the result of the query is the vector handle initialized to the set of objects, if a 0 handle had been passed, respectively the existing vector extended by 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

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

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 the C++ language, 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 MQ Workflow API. The ownership of the array elements, the objects, stays with the array.

C-language vectors

Vector accessor functions are described below. This is because all these functions are similar looking 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-language 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 MQ Workflow internal error has occurred. Contact your IBM representative.

Vector accessor functions allow for the operations listed below; **'Xxx' denotes some 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 C-language handle is set to 0 so that it can no longer be used.

C-language signature

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

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-language signature

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

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-language signature

```
FmcjXxxHandle FMC_APIENTRY FmcjXxxVectorNextElement(
                                   FmcjXxxVectorHandle hdlVector )
```

Parameters
hdlVector 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-language signature

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

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

Return type
unsigned long
The number of elements in the vector.

C-language examples

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

Using First/NextElement() calls

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

Figure 3. Reading a vector in C (using First/NextElement() calls) (Part 1 of 5)

```
FmcjGlobalConnect();

FmcjExecutionServiceAllocate(&service);
rc = FmcjExecutionServiceLogon( service,
                                "ADMIN", "PASSWORD",
                                Fmc_SM_Default, Fmc_SA_Reset
                               );
if ( rc != FMC_OK )
  return rc;
printf("Logged on\n");
```

Figure 3. Reading a vector in C (using First/NextElement() calls) (Part 2 of 5)

```
rc= FmcjExecutionServiceQueryProcessInstances(
                              service,
                              FmcjNoFilter,
                              FmcjNoSortCriteria,
                              FmcjNoThreshold,
                              &hdlVector  );
if ( rc != FMC_OK )
   return rc;
printf("Queried process instances\n");
```

Figure 3. Reading a vector in C (using First/NextElement() calls) (Part 3 of 5)

```
hdlInstance= FmcjProcessInstanceVectorFirstElement(hdlVector);
numElements= FmcjProcessInstanceVectorSize(hdlVector);

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

FmcjProcessInstanceVectorDeallocate(&hdlVecor);
```

Figure 3. Reading a vector in C (using First/NextElement() calls) (Part 4 of 5)

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

   FmcjGlobalDisconnect();
   return FMC_OK;
}
```

Figure 3. Reading a vector in C (using First/NextElement() calls) (Part 5 of 5)

Using NextElement() call only

```
#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]="";
```

Figure 4. Reading a vector in C (using NextElement() only) (Part 1 of 5)

```
  FmcjGlobalConnect();

  FmcjExecutionServiceAllocate(&service);
  rc = FmcjExecutionServiceLogon( service,
                                  "ADMIN", "PASSWORD",
                                  Fmc_SM_Default, Fmc_SA_Reset
                                );
  if ( rc != FMC_OK )
    return rc;
  printf("Logged on\n");
```

Figure 4. Reading a vector in C (using NextElement() only) (Part 2 of 5)

```
  rc= FmcjExecutionServiceQueryProcessInstances(
                                service,
                                FmcjNoFilter,
                                FmcjNoSortCriteria,
                                FmcjNoThreshold,
                                &hdlVector  );
  if ( rc != FMC_OK )
    return rc;
  printf("Queried process instances\n");
```

Figure 4. Reading a vector in C (using NextElement() only) (Part 3 of 5)

```
  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) );
```

Figure 4. Reading a vector in C (using NextElement() only) (Part 4 of 5)

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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

Figure 4. Reading a vector in C (using NextElement() only) (Part 5 of 5)

ActiveX arrays

In ActiveX, the result of a query for a set of objects is stored in arrays. The arrays are provided by the respective ActiveX Controls. You cannot allocate or delete an array.

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

All arrays provide for the same methods to query the number of objects contained and the objects themselves.

All array indexes start with 0 (zero). That is, valid index numbers are 0 to GetSize()-1. Note that you should not remember the index number of an object because the object can have a different index after each query, depending on the sort criteria and the number of objects returned.

Exceptions

Following exceptions can be thrown:

FMC_WRONG_INDEX(1501)
The index is out of the range of the array.

Add

Adds a new object to the ContainerArray or to the ContainerElementArray.

Signature

```
long Add()
```

Adds a new execution service to the ExecutionServiceArray.

Signature

```
long Add           ( BSTR system, BSTR systemGroup)
long AddDefault    ()
long AddSystemGroup( BSTR systemGroup )
```

Adds the specified string to the StringArray.

Signature

```
long Add ( BSTR string )
```

Parameters

string Input. The string to be added to the StringArray.

system Input. The system where the execution server runs.

systemGroup Input. The system group where the system resides.

Return type

long The index of the added object in the array.

GetAt

Returns the object at the specified index.

Signature

```
Object GetAt ( long index )
```

Parameters

index Input. The index of the object to be retrieved.

Return type

Object An object of the type contained in the array.

GetSize

Returns the number of elements in the array.

ActiveX signature

```
long GetSize()
```

Return type

long The cardinality of the array.

RemoveAll

Removes all objects from the StringArray.

Signature

```
void RemoveAll ( )
```

RemoveAt

Removes the object at the specified index; can be called on the ExecutionServiceArray, ContainerArray, ContainerElementArray, and StringArray.

Signature

```
void RemoveAt ( long index )
```

Parameters

index Index. The index of the object to be removed.

SetAt

Sets the value of a StringArray element at the specified index.

Signature

```
void SetAt( long index, BSTR string )
```

Parameters

index Input. The index of the array value to be set.

string Input. The value to be set.

Events

NewObject

Indicates that a new execution service has been added to the ExecutionServiceArray or that a new list object has been added to the ProcessInstanceListArray, the ProcessTemplateListArray, or the Worklist array.

Signature

```
void NewObject( long index )
```

Parameters

index Input. The index of new element in the array.

ObjectRemove

Indicates that an execution service has been removed from the ExecutionServiceArray or that a list object has been removed from the ProcessInstanceListArray, the ProcessTemplateListArray, or the Worklist array.

Signature

```
void ObjectRemove( long index )
```

Parameters

index Input. The index of new element in the array.

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]. Note that you should not remember the index number of an object because the object can have a different index after each query, depending on the sort criteria and the number of objects returned.

Chapter 9. Handling containers

A container represents input or output data of a process template, process instance, work item, activity implementation, or support tool 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 4 MB 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 MQ 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.

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[3] 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.

3. A fully qualified name in XML is represented by the nesting hierarchy.

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.

For detailed examples see “Part 9. Examples and scenarios” on page 753.

The XML message interface

In the XML message interface, data members (container elements) are represented as follows:

- The data member name is represented by an XML element name.
- Nested data structures are decomposed into XML child elements according to their structure, that is, there is no dot notation for fully qualified names.
- Arrays are depicted as a sequence of elements.
- The data member type is not part of the XML element content.

For example:

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

For more information refer to "Container data" on page 211.

Predefined data members

All containers automatically specify data members predefined by MQ 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 or support tools.
- 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 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 API call. An exception is the _RC data member which should be set only if the program cannot otherwise specify 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 API calls. Their values *can be read* using API container API calls. Fixed data member _RC is available in output containers but should only be set when your compiler does not support a program exit code.

_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 when the implementation is a program. Typically it is used to evaluate exit and transition conditions. It cannot be read from input containers and, unless it has not been set explicitly, it is automatically set to the exit code of the activity implementation when that program ends. If set explicitly, then that value stays.

In cases where your compiler does not support an exit code, you can use the Container API to set its value.

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 MQ 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

_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 MQ 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. MQ Workflow does not deduce any meaning from this value; it is just used for client purposes. Any integer value between 0 and 999 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.

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: STRING

_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.

Notes:

1. ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.
2. 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 MQ Workflow predefined data members.

ActiveX signatures

```
long LeafCount()

void Leaves( ContainerElementArray * leaves )

long AllLeafCount()

void AllLeaves( ContainerElementArray * leaves )
```

C-language signatures

```
unsigned long FmcjContainerLeafCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerLeaves( FmcjContainerHandle handle )

unsigned long FmcjContainerAllLeafCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerAllLeaves( FmcjContainerHandle handle )
```

C++ language signatures

```
unsigned long LeafCount()

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

unsigned long AllLeafCount()

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

Java signatures

```
public abstract int leafCount() throws FmcException

public abstract ContainerElement[] leaves() throws FmcException

public abstract int allLeafCount() throws FmcException

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

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.

long/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.

ActiveX signatures

```
long MemberCount()

void StructMembers( ContainerElementArray * members )
```

C-language signatures

```
unsigned long FmcjContainerMemberCount( FmcjContainerHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerStructMembers( FmcjContainerHandle handle )
```

C++ language signatures

```
unsigned long MemberCount()

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

Java signatures

```
public abstract int memberCount() throws FmcException

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

Parameters

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.

long/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.

ActiveX signature

```
BSTR Type()
```

C-language signature

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

C++ language signature

```
string Type()
```

Java signature

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

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

BSTR/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.
Functions/methods you have seen on 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.

ActiveX signatures

```
BSTR Name()

BSTR FullName()

BSTR Type()
```

C-language signatures

```
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++ language signatures

```
string Name() const

string FullName() const

string Type() const
```

Java signatures

```
public abstract String name() throws FmcException

public abstract String fullName() throws FmcException

public abstract String type() throws FmcException
```

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

BSTR/char*/string/String

The name or type of the container element.

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.

ActiveX signatures

```
boolean IsArray()

boolean IsLeaf()

boolean IsStruct()
```

C-language signatures

```
bool FmcjContainerElementIsArray ( FmcjContainerElementHandle handle )

bool FmcjContainerElementIsLeaf  ( FmcjContainerElementHandle handle )

bool FmcjContainerElementIsStruct( FmcjContainerElementHandle handle )
```

C++ language signatures

```
bool IsArray () const

bool IsLeaf  () const

bool IsStruct() const
```

Java signatures

```
public abstract boolean isArray () throws FmcException

public abstract boolean isLeaf  () throws FmcException

public abstract boolean isStruct() throws FmcException
```

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.

Note: When these API calls are called on a leaf itself, the LeafCount() returns 1 because the container element obviously is a leaf, but no further leaves are returned when Leaves() are queried.

ActiveX signatures

```
long LeafCount()

void Leaves( ContainerElementArray * leaves )
```

C-language signatures

```
unsigned long
FmcjContainerElementLeafCount( FmcjContainerElementHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerElementLeaves( FmcjContainerElementHandle handle )
```

C++ language signatures

```
unsigned long LeafCount()

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

Java signatures

```
public abstract int leafCount() throws FmcException

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

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.

long/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.

ActiveX signatures

```
long MemberCount()

void StructMembers( ContainerElementArray * members )
```

C-language signatures

```
unsigned long
FmcjContainerElementMemberCount( FmcjContainerElementHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerElementStructMembers( FmcjContainerElementHandle handle )
```

C++ language signatures

```
unsigned long MemberCount()

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

Java signatures

```
public abstract int memberCount() throws FmcException

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

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.

long/unsigned long/int
The number of structural members.

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.

ActiveX signatures

```
long Cardinality()

void ArrayElements( ContainerElementArray * elements )
```

C-language signatures

```
unsigned long
FmcjContainerElementCardinality( FmcjContainerElementHandle handle )

FmcjContainerElementVectorHandle
FmcjContainerElementArrayElements( FmcjContainerElementHandle handle )
```

C++ language signatures

```
unsigned long Cardinality() const

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

Java signatures

```
public abstract int cardinality() throws FmcException

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

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.

long/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 or container element structure.

Notes:

1. A qualified name must start with a letter and cannot start with brackets or parentheses. In other words, if you want to access an element of a container element which is an array, then you need to call the ArrayElements() API call.
2. ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signature

```
long GetElement( BSTR              qualifiedName,
                 ContainerElement * element      )
```

C-language signature

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

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

C++ language signature

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

Java signature

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

Parameters

element Output. The container element.

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

qualifiedName
Input. The fully qualified name of the container element.

Return type
long/APIRET The return code of calling this API call - see return codes.

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.

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.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signatures

```
long GetValueDbl( BSTR      qualifiedName,
                  double *  value,
                  boolean   isArray,
                  long      index        )

long GetValueLng( BSTR      qualifiedName,
                  long *    value,
                  boolean   isArray,
                  long      index        )

long GetValueStr( BSTR      qualifiedName,
                  BSTR *    value,
                  boolean   isArray,
                  long      index       )
```

C-language signatures

```
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-language signatures

```
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-language signatures

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

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

C-language signatures

```
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          bufferLength )

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

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

C++ language signatures

```
unsigned long BinaryLength( unsigned long index )

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

unsigned long BinaryLength()
```

C++ language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

Java signatures

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

public abstract
byte[] getBuffer(   String qualifiedName ) throws FmcException
```

Java signatures

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

public abstract
double getDouble(   String qualifiedName ) throws FmcException
```

Java signatures

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

public abstract
int getLong(        String qualifiedName ) throws FmcException
```

Java signatures

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

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.

Return type

byte[]/double/int/String
The leaf value.

unsigned long
The minimum required buffer length for reading the value.

long/APIRET The return code of calling this API call - see return codes.

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.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signatures

```
long GetValueDbl( double *  value,
                  long      index )

long GetValueLng( long *    value,
                  long      index )

long GetValueStr( BSTR *    value,
                  long      index )
```

C-language signatures

```
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-language signatures

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

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

C-language signatures

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

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

C-language signatures

```
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 FmcjContainerElementArrayStringLength(
                     FmcjContainerElementHandle handle    )

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

C++ language signatures

```
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++ language signatures

```
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
```

Java signatures

```
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
```

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. In ActiveX, the index is ignored for a container element which is no array.

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.

long/APIRET The return code of calling this API call - see return codes.

Setting a value of a container

The following API calls allow to set the value of a container leaf in a read/write container.

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.

Setting a container value changes the value in the API cache only; the execution server is not contacted. The container can then be used as the input container for a process instance (Start(), CreateAndStart(), Execute()), as the output container of a work item (CheckIn(), SetOutContainer()), or as a corrective container when calling ForceFinish() or ForceRestart().

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signatures

```
long SetValueDbl( BSTR      qualifiedName,
                  double    value,
                  boolean   isArray,
                  long      index        )

long SetValueLng( BSTR      qualifiedName,
                  long      value,
                  boolean   isArray,
                  long      index        )

long SetValueStr( BSTR      qualifiedName,
                  BSTR      value,
                  boolean   isArray,
                  long      index       )
```

C-language signatures

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

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

C-language signatures

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

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

C-language signatures

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

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

C-language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

Java signatures

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

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

Java signatures

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

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

Java signatures

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

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

Java signatures

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

Parameters

dataLength Input. The length of the binary value.

handle Input. The handle of the container to be set.

index Input. When the leaf is an array, the index of the array element to be set.

isArray Input. If set to *True*, an array element is to be set and the index is used.

qualifiedName
Input. The fully qualified name of the leaf within the container.

value 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

long/APIRET The return code of calling this API call - see return codes.

Return codes/FmcException

The following return codes can be returned or can be described by the result object or following exceptions can be thrown, the number in parentheses shows their 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 MQ 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.

Chapter 10. Monitoring a process instance

MQ 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.

Obtaining an process instance monitor

Once a process instance[4] has been accessed, an **instance monitor** for a process instance can be obtained (ObtainProcessMonitor()). The transient 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.

4. or activity instance or a (work) item

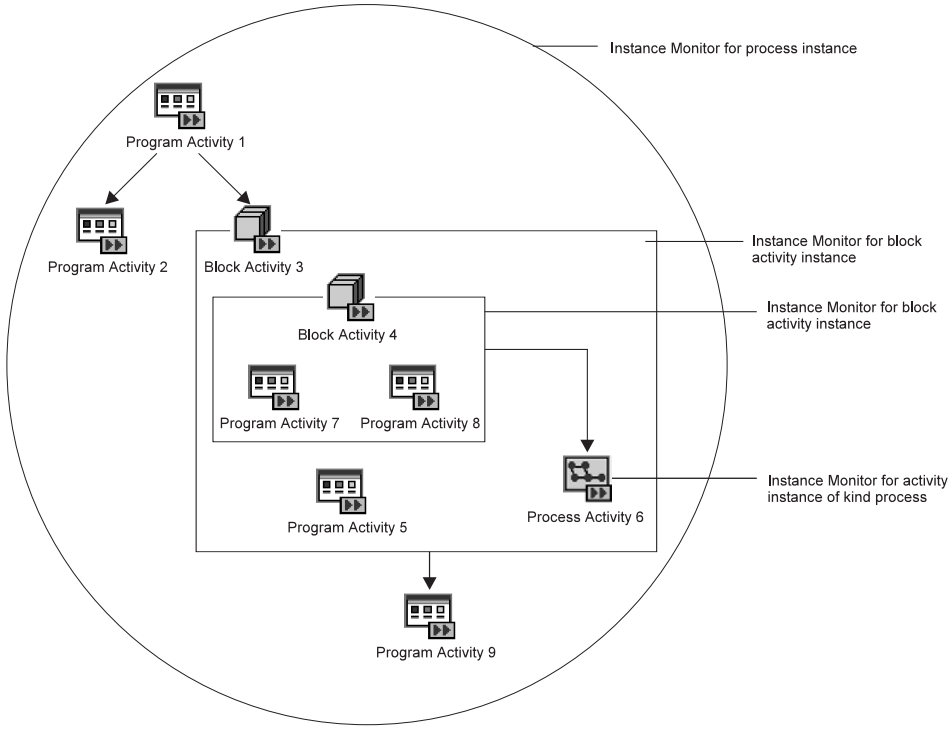


Figure 5. Instance monitors for process instances and activity instances of type Block

For example, the illustrated 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 instance monitor object 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 instance of type *Block* is encountered, it is possible to obtain its instance monitor (ObtainBlockMonitor()). Similar to an process instance monitor, an instance monitor for an activity instance of type *Block* 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 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*.

When an activity instance of type *Process* is encountered, it is possible to obtain its instance monitor, either via the embracing instance monitor object (ObtainProcessMonitor()) or by retrieving the implementing (sub)process instance of the activity instance and then obtaining the associated instance monitor.

When obtaining an process instance monitor, it is possible to use the *deep option* in order to specify that *all* monitors for activities of type *Block* are to be returned from the MQ Workflow execution server in the same step. The instance monitors for the block activity instances then all show the state of the process instance at this retrieval time. This means, when an block 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 an block 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 currently ignored.

Ownership of monitors

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

Chapter 11. 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 MQ Workflow action; performing that action can itself request some specific authority.

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.

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 49 for details).

The following table shows the authorizations and the MQ 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 authenticated himself to MQ 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 3. 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 3. Authorization for persons (continued)

Name	E/I	Authorized Functions
Process authorization	E	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: • Create • Start • Create and start • Set process instance name • Query • Refresh 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: • Query • Refresh The appropriate FDL authorization keyword is PROCESS_CATEGORY.
Process administration authorization	E	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: • Delete • Restart • Resume • Suspend • Terminate 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: • Force-finish • Force-restart The appropriate FDL authorization keyword is PROCESS_CATEGORY AS ADMINISTRATOR.

Table 3. Authorization for persons (continued)

Name	E/I	Authorized Functions
Process administrator	I	Has process administration authority for the appropriate process instance.
Process creator	I	Can perform the following process instance functions: • Set process instance name • Delete, if not yet started • Query • Refresh • Start
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: • Query • Refresh • Transfer The appropriate FDL authorization keyword is WORKITEMS_OF.
Workitem owner	I	Can perform all functions on the assigned (work) item except: • Force Finish • Force Restart

Chapter 12. Stateless server support

In clustered application server environments, like Web server farms, client requests are sent for scalability and fail-over to a number of different Web servers via routing components. Routing is done fully dynamically so that there exists no 1-1 client/server affinity.

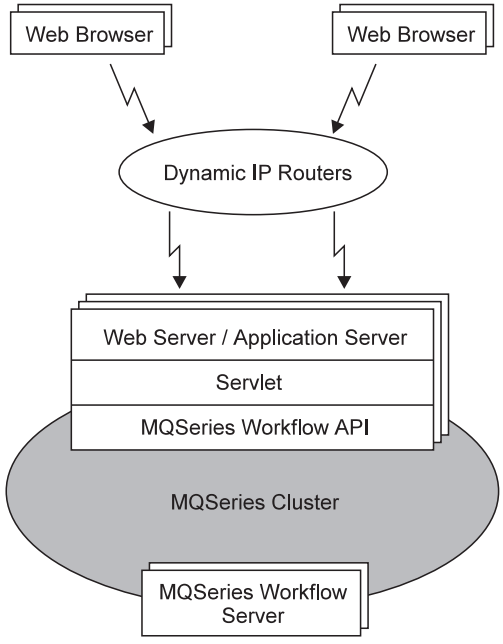


Figure 6. Stateless server support

From a Web server point of view, client requests arrive independently of each other. The Web server has no prior knowledge of the client. Such a server configuration is called ”stateless”.

Implementation of *stateless server*s is supported by the MQ Workflow APIs.

Since the applications using the MQ Workflow API, for example, the Web servers, have no knowledge beside the client request itself, the MQ Workflow API allows to pass all state from one application process to the other. The MQ Workflow objects are serializable.

- All identity-based objects allow for the retrieval of their unique object ID (OID) and support creation from an OID.

- All value-based objects allow for being streamed and support creation from a stream.

Identity-based objects are:

- Sessions, that is, execution service objects
- Process templates
- Process instances
- Activity instances
- Work items
- Activity instance notifications and process instance notifications
- Process template lists, process instance lists, and worklists
- Instance monitors
- Persons (identified by their user ID)

These objects expose API calls:

- PersistentOID() to retrieve a string representation of their OIDs.

  A session's OID is retrieved by the SessionID() API call.
- Persistent*Object*() to recreate the object.

  (*Object* in Persistent*Object*() stands for the object created, for example, PersistentWorkitem() creates a work item from its OID).

  The session is recreated by the SetSessionContext() API call.

Recreation means that the object is created in the API cache, that is, a transient object representing the persistent object identified by the OID is created. An MQ Workflow server is not contacted. As with other objects, the caller becomes responsible for the object, that is, the caller needs to deallocate the object when no longer needed.

Value-based objects are:

- Containers
- Program data
- Program templates

These objects expose API calls:

- AsStream() to retrieve a serialized representation of the object.
- FromStream() API calls to recreate the object from the stream.

Again, recreation means that the object is created in the API cache. An MQ Workflow server is not contacted.

For example, consider the following simple scenario, which concentrates on the extensions for the stateless server support only; code snippets are sketched in C++:

1. Application process A receives an MQ Workflow logon request for user *JIM*. It allocates an execution service object and logs on to MQ Workflow specifying the user ID *JIM*.

   Application process A retrieves the ID of the session established and returns. Note that the session exists until Logoff() is called or until the session expires. The session is not removed from the MQ Workflow server when the application program ends.

   ```
   //
   // Establish a session
   //
     FmcjExecutionService service;
     APIRET rc = service.Logon("JIM", "password");
     if ( rc != FMC_OK )
     {
       cout << "Logon failed, - rc: " << rc << endl;
       return;
     }

     // retrieve session relevant information
     string sessionID= service.SessionID();
   ```

2. Application process B receives the request for a process template and its container together with the information on the already existing session, the session ID, and the user ID for whom the session has been established.

   It allocates an execution service object and attaches itself to the specified session.

   ```
   //
   // Attach to an existing session
   //
     FmcjExecutionService service;
     APIRET rc = service.SetSessionContext(userID,sessionID);
     if ( rc != FMC_OK )
     {
       cout << "Reestablish session failed, - rc: " << rc << endl;
       return;
     }
     ...
   ```

   Application process B queries the requested process template and its associated input container from the MQ Workflow server. It then retrieves the OID of the process template

   ```
   //
   // Retrieve the OID of an object identified by obj
   //
   string oid= obj.PersistentOID();
   ```

and the stream format of the container.

```
//
// Retrieve a value-based object in stream format, for example,
// a read/write container identified by container
//
unsigned long bufferLength   = container.StreamLength();
char          containerBuffer= new char[bufferLength];

if ( 0 == container.AsStream(containerBuffer, bufferLength) )
{
  cout << "Retrieval of stream failed" << endl;
  return;
}
  ...
```

Since the process template and its container are the objects which are potentially acted on by the next client request, application B returns the process template OID and the container stream to the caller so that the next request can be handled by a different application process; even on a different operating system platform.

3. Application process C receives a request to create and start a process instance together with all information needed, the session ID, the user ID for whom the session has been established, the process template OID, and the container stream.

   It allocates an execution service object and attaches itself to the specified session - see 2 on page 89.

   It recreates the process template from its OID

```
//
// Recreate an object from an OID, for example, a process template
// The object constructed needs to be deleted/deallocated when no longer needed
//
FmcjProcessTemplate *pProcess= service.PersistentProcessTemplate(ptOid);

if ( 0 == pProcess )
{
  cout << "Creation of process template failed" << endl;
  // determine reason of failure from result object
  return;
}
  ...
```

   and the container from the stream passed.

```
//
// Recreate an object from a stream, for example, a read/write container
// The object constructed needs to be deleted/deallocated when no longer needed
//
FmcjReadWriteContainer *pContainer=
    service.ReadWriteContainerFromStream(containerBuffer,bufferLength);

if ( 0 == pContainer )
```

```
{
  cout << "Creation of container failed" << endl;
  // determine reason of failure from result object
  return;
}
  ...
```

Application process C issues the CreateAndStartProcessInstance() action method. On successful return from the MQ Workflow server, it retrieves the OID of the process instance created and returns that information to the caller - see 2 on page 89.

4. Application process D receives a request to terminate the process instance together with all information needed, the session ID, the user ID for whom the session has been established, and the process instance OID.

   It allocates an execution service object and attaches itself to the specified session - see 2 on page 89.

   It recreates the process instance from its OID - see 3 on page 90.

   It issues the Terminate() action method.

5. Depending on the client request, issue MQ Workflow API calls similar to steps 2 on page 89, 3 on page 90, or 4.

Chapter 13. Types of API calls

MQ 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 or support tool
Program execution management	Handle program execution agents

Basic and accessor API calls are described in more 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 "Part 7. API action and activity implementation calls" on page 339. These API calls require client/server communication or communication with the program execution agent.

Basic API calls

Basic API calls are essentially provided so that transient objects can 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 MQ Workflow API. Refer also to "Chapter 17. Memory management" on page 167.

Basic API calls are only provided in the various APIs as far as needed. For example, the Java language does only support 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-language calls and the MQ 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 0-pointer or it does not conform to the syntax rules.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative. Also, check the MQ Workflow trace for any exceptions encountered.

Basic API calls allow for the basic operations listed below; **Xxx denotes some class or scope**, for example, FmcjXxxEqual() can stand for FmcjProcessInstanceEqual().

Allocation

Following API calls allow the application to set up the respective object. This is needed for supporting objects like string vectors or, in ActiveX, for objects to be initialized by a persistent one. Transient objects representing persistent objects are allocated implicitly by the MQ Workflow API when persistent objects are created or queried from an MQ 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.

ActiveX signatures

```
ActivityInstance *              NewActivityInstance()
ActivityInstanceNotification *  NewActivityInstanceNotification()
Container *                     NewContainer()
ExecutionService *              NewExecutionService()
InstanceMonitor *               NewInstanceMonitor()
Person *                        NewPerson()
ProcessInstance *               NewProcessInstance()
ProcessInstanceList *           NewProcessInstanceList()
ProcessInstanceNotification *   NewProcessInstanceNotification()
ProcessTemplate *               NewProcessTemplate()
ProcessTemplateList *           NewProcessTemplateList()
ProgramData *                   NewProgramData()
ProgramTemplate *               NewProgramTemplate()
Workitem *                      NewWorkitem()
Worklist *                      NewWorklist()
```

C-language signatures

```
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++ language signature

```
FmcjXxx()

FmcjDateTime( bool initWithCurrentDateTime= false )

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

FmcjExecutionService( string const & systemGroup )

FmcjExecutionService( string const & system,
                      string const & systemGroup )
```

Java signatures

```
Agent()

ReadOnlyContainerHolder()
ReadOnlyContainerHolder( ReadOnlyContainer value )
```

Parameters

service
Input/Output. The address of the handle to the object to be set when the object has been constructed. Care 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. Only specifying 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.

value Input. A read-only container which initializes the holder object.

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++ language signature

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

Parameters

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

Comparison/equality

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 the C-language, 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.

ActiveX signature

```
boolean IsEqual( IDispatch * anObject )
```

C-language signature

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

C++ language signature

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

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.

Conversion

Following API calls allow the application to convert a read-only container into a read/write container and vice versa. Note that a copy of the original container is created in Java and the C-language.

C-language signatures

```
FmcjReadWriteContainerHandle FMC_APIENTRY
    FmcjReadOnlyContainerAsReadWriteContainer(
                          FmcjReadOnlyContainerHandle handle )

FmcjReadOnlyContainerHandle FMC_APIENTRY
    FmcjReadWriteContainerAsReadOnlyContainer(
                           FmcjReadWriteContainerHandle handle )
```

C++ language signatures

```
operator FmcjReadWriteContainer();

operator FmcjReadOnlyContainer();
```

Java signatures

```
public abstract
ReadWriteContainer asReadWriteContainer() throws FmcException

public abstract
ReadOnlyContainer  asReadOnlyContainer () throws FmcException
```

Parameters

handle Input. The handle of the read/write or read-only container to be converted.

Copy

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-language signature

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

C++ language signature

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

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. Care 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

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-language 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. In ActiveX, setting the object to Nothing decreases its use count so that it can become available for destruction.

C-language signature

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

C++ language signature

```
virtual ¬FmcjXxx()
```

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 MQ Workflow server, that is, both primary and secondary properties are available (see also "Accessor/mutator API calls" on page 106).

ActiveX signature

```
boolean IsComplete()
```

C-language signature

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

C++ language signature

```
bool IsComplete()
```

Java signature

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

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.

ActiveX signature

```
boolean IsEmpty()
```

C++ language signature

```
bool IsEmpty()
```

Java signature

```
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.

ActiveX signature

```
Enum Kind()
```

C-language signature

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

C++ language signature

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

Java signature

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

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 111.

C-language 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;
```

Figure 7. C example using basic functions (Part 1 of 8)

```
  FmcjGlobalConnect();

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

Figure 7. C example using basic functions (Part 2 of 8)

```
/*  Query Workitems */
rc= FmcjExecutionServiceQueryWorkitems( service,
                                        FmcjNoFilter,
                                        FmcjNoSortCriteria,
                                        FmcjNoThreshold,
                                        &wList );
printf( "\nQuery workitems returns rc : %u\n", rc );
fflush(stdout);
```

Figure 7. C example using basic functions (Part 3 of 8)

```
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);
```

Figure 7. C example using basic functions (Part 4 of 8)

```
                                   /* 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);
```

Figure 7. C example using basic functions (Part 5 of 8)

```
                                    /* 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);
```

Figure 7. C example using basic functions (Part 6 of 8)

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

Figure 7. C example using basic functions (Part 7 of 8)

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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

Figure 7. C example using basic functions (Part 8 of 8)

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");
```

Figure 8. C++ example using basic methods (Part 1 of 7)

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

Figure 8. C++ example using basic methods (Part 2 of 7)

```
// Query Workitems
vector<FmcjWorkitem>        wList;
rc= service.QueryWorkitems( FmcjNoFilter,
                            FmcjNoSortCriteria,
                            FmcjNoThreshold,
                            wList );
cout << "Query workitems returns rc : " << rc << endl ;
```

Figure 8. C++ example using basic methods (Part 3 of 7)

```
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;
```

Figure 8. C++ example using basic methods (Part 4 of 7)

```
  FmcjWorkitem workitem2= wList[1];  // access next element
  if ( workitem1 == workitem2 )
    cout << "Surprise - workitems are equal" << endl;
  else
    cout << "Workitems represent different objects" << endl;
```

Figure 8. C++ example using basic methods (Part 5 of 7)

```
                                     // 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
```

Figure 8. C++ example using basic methods (Part 6 of 7)

```
  // logoff
  rc = service.Logoff();

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

Figure 8. C++ example using basic methods (Part 7 of 7)

Accessor/mutator API calls

Accessor/mutator 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 automatically done from an MQ Workflow server when the property is accessed and not yet available in the API cache or explicitly done by using the appropriate create, query, or refresh API calls. Creation or update can be done on the client when the MQ Workflow server sends new information (pushes information).

Default values are provided to you as long as the transient object is *empty* 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++, ActiveX, 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.

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 *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 cannot have been set to a 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.

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.

Primary/secondary properties

By default, the MQ Workflow API provides for two views on 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 provided by the server when objects are queried and can be used in filter expressions. Secondary properties, and a refresh of the primary properties, are only provided when the secondary property is accessed (the API automatically refreshes the object when a property is queried and not yet available) or 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.

This means for an API program that there is no general need to distinguish between primary and secondary properties from an access point of view. You might, however, want to consider that an object is automatically refreshed when a not yet available property is read. Or, from a performance point of view, you might want to prevent unnecessarily accessing properties not yet available.

Return codes

Accessor/mutator 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 the C++ or C-language, you can query the MQ Workflow result object for any errors encountered. Java throws an FmcException. The following codes can occur, 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_EMPTY(122)
The object has not yet been read from the database, that is, default values are returned.

FMC_ERROR_BUFFER(800)
The buffer provided is too small to hold the largest possible value. See file fmcmxcon.h for required lengths.

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; can be returned when the API automatically refreshes the object.

FMC_ERROR_DOES_NOT_EXIST(118)
The object does not or no longer exist. For example, the message is not found in the message catalog.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative. Also, check the MQ Workflow trace for any exceptions encountered.

FMC_ERROR_INVALID_CONFIGURATION_ID(1022)
The configuration provided is invalid; it is 0 or it does not conform to its 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_INVALID_RESULT_HANDLE(814)
The handle of the result object provided is invalid; it is 0 or it 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_MESSAGE_FORMAT(103)
An internal message format error; can be returned when the API automatically refreshes the object. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain; can be returned when the API automatically refreshes the object.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call; can be returned when the API automatically refreshes the object.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on; can be returned when the API automatically refreshes the object.

FMC_ERROR_PROFILE(124)
The profile cannot be found or opened.

FMC_ERROR_PROGRAM_EXECUTION(126)
The API call cannot be called from within an activity implementation, for example, SetConfiguration().

FMC_ERROR_TIMEOUT(14)
Timeout has occurred; can be returned when the API automatically refreshes the object.

FMC_ERROR_WRONG_STATE(120)
The API call cannot be executed because the object is in a wrong state. For example, the configuration cannot be changed after logon.

Accessor API calls allow for the operations listed below; **Xxx denotes some 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.

ActiveX signature

```
boolean Property()
```

C-language signature

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

C++ language signature

```
bool Property() const
```

Java signature

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

Parameters

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

Return type

bool/ boolean The property value.

Declaration examples

ActiveX boolean ManualStartMode();

C-language 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. Date/time values are expressed in local time.

ActiveX signature

```
void Property( DateAndTime * time )
```

C-language signature

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

C++ language signature

```
FmcjDateTime Property() const
```

Java signature

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

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

ActiveX void EndTime(DateAndTime * time);

C-language 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.

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

Note: The Java language does not support enumeration types. For that reason, enumerations are implemented by final classes with constants.

ActiveX signature

```
Enum Property()
```

C-language signature

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

C++ language signature

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

Java signature

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

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

ActiveX AssignReason ReceivedAs();

C-language FmcjItemAssignReason FMC_APIENTRY FmcjWorkitemReceivedAs(FmcjWorkitemHandle handle);

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

Java public abstract AssignReason receivedAs() throws FmcException;

Following enumeration types and constants are defined; types are listed in the order ActiveX, C-language, C++, Java. Numbers in parantheses are the corresponding integer values. You are stronlgy adviced to use the symbolic names only.

Assign reason

This enumeration is named AssignReason in ActiveX, FmcjItemAssignReason in the C-language, FmcjItem::AssignReason in C++ and com.ibm.workflow.api.ItemPackage.AssignReason in Java. Possible values are:

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

- **ActiveX** AssignReason_NotSpecified
- **C-language** Fmc_IR_NotSet
- **C++** FmcjItem::NotSpecified
- **Java** AssignReason.NOT_SPECIFIED

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

- **ActiveX** AssignReason_Normal
- **C-language** Fmc_IR_Normal
- **C++** FmcjItem::Normal
- **Java** AssignReason.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.

- **ActiveX** AssignReason_Substitute
- **C-language** Fmc_IR_Substitute
- **C++** FmcjItem::Substitute
- **Java** AssignReason.Substitute

ProcessAdministrator(3) Indicates that the work item or notification has been assigned because the user is the process administrator.

- **ActiveX** AssignReason_ProcessAdministrator
- **C-language** Fmc_IR_ProcessAdministrator
- **C++** FmcjItem::ProcessAdministrator

Java AssignReason.PROCESS_ADMINISTRATOR

SystemAdministrator(4)
Indicates that the work item or notification has been assigned because the user is the system administrator.

ActiveX AssignReason_SystemAdministrator

C-language Fmc_IR_SystemAdministrator

C++ FmcjItem::SystemAdministrator

Java AssignReason.SYSTEM_ADMINISTRATOR

ByTransfer(5) Indicates that the work item or notification has been transferred to the user.

ActiveX AssignReason_ByTransfer

C-language Fmc_IR_ByTransfer

C++ FmcjItem::ByTransfer

Java AssignReason.BY_TRANSFER

Audit setting

This enumeration is named AuditSetting in ActiveX, FmcjProcessTemplateAuditSetting in the C-language, FmcjProcessTemplate::AuditSetting in C++ and com.ibm.workflow.api.ProcessTemplatePackage.AuditSetting in Java. Possible values are:

NotSet(0) Indicates that nothing is known about the audit setting.

ActiveX Audit_NotSet

C-language Fmc_TA_NotSet

C++ FmcjProcessTemplate::NotSet

Java AuditSetting.NOT_SET

NoAudit(1) Indicates that auditing is not to be performed.

ActiveX Audit_NoAudit

C-language Fmc_TA_NoAudit

C++ FmcjProcessTemplate::NoAudit

Java AuditSetting.NO_AUDIT

Condensed(2) Indicates that condensed auditing is to be performed.

ActiveX Audit_Condensed

C-language Fmc_TA_Condensed

C++ FmcjProcessTemplate::Condensed

Java AuditSetting.CONDENSED

Full(3) Indicates that full auditing is to be performed.

ActiveX Audit_Full

C-language Fmc_TA_Full

C++ FmcjProcessTemplate::Full

Java AuditSetting.FULL

Filter(4) Indicates that fine-grained, filtered auditing is to be performed.

ActiveX Audit_Filter

C-language Fmc_TA_Filter

C++ FmcjProcessTemplate::Filter

Java AuditSetting.FILTER

Activity instance escalation

This enumeration is named AIEscalation in ActiveX, FmcjActivityInstanceEscalation in the C-language, FmcjActivityInstance::Escalation in C++ and com.ibm.workflow.api.ActivityInstancePackage.Escalation in Java. Possible values are:

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

ActiveX AIEscalation_NotSpecified

C-language Fmc_AE_NotSet

C++ FmcjActivityInstance::NotSpecified

Java Escalation.NOT_SPECIFIED

NoNotification(1)
Indicates that no notification occurred so far on the activity instance.

ActiveX AIEscalation_NoNotification

C-language Fmc_AE_NoNotification

C++ FmcjActivityInstance::NoNotification

Java Escalation.NO_NOTIFICATION

FirstNotification
Indicates that the first notification occurred.

ActiveX(4) AIEscalation_FirstNotification

C-language(4) Fmc_AE_FirstNotification

C++(4) FmcjActivityInstance::FirstNotification

Java(2) Escalation.FIRST_NOTIFICATION

SecondNotification
Indicates that the second notification occurred.

ActiveX(5) AIEscalation_SecondNotification

C-language(5) Fmc_AE_SecondNotification

C++(5) FmcjActivityInstance::SecondNotification

Java(3) Escalation.SECOND_NOTIFICATION

Activity instance state

This enumeration is named ActivityInstanceState in ActiveX, FmcjActivityInstanceStateValue in the C-language, FmcjActivityInstance::state in C++ and com.ibm.workflow.api.ActivityInstancePackage.ExecutionState in Java. Possible values are:

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

ActiveX AIState_Undefined

C-language Fmc_AS_NotSet

C++ FmcjActivityInstance::undefined

Java ExecutionState.UNDEFINED

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

ActiveX AIState_Ready

C-language Fmc_AS_Ready

C++ FmcjActivityInstance::ready

Java ExecutionState.READY

Running(2) Indicates that the activity instance is in the running state.

ActiveX AIState_Running

C-language Fmc_AS_Running

C++ FmcjActivityInstance::running

Java ExecutionState.RUNNING

Finished Indicates that the activity instance is in the finished state.

ActiveX(4) AIState_Finished

C-language(4) Fmc_AS_Finished

C++(4) FmcjActivityInstance::finished

Java(3) ExecutionState.FINISHED

Terminated Indicates that the activity instance is in the terminated state.

ActiveX(8) AIState_Terminated

C-language(8) Fmc_AS_Terminated

C++(8) FmcjActivityInstance::terminated

Java(4) ExecutionState.TERMINATED

Suspended Indicates that the activity instance is in the suspended state.

ActiveX(16) AIState_Suspended

C-language(16) Fmc_AS_Suspended

C++(16) FmcjActivityInstance::suspended

Java(5) ExecutionState.SUSPENDED

Inactive Indicates that the activity instance is still inactive.

ActiveX(32) AIState_Inactive

C-language(32) Fmc_AS_Inactive

C++(32) FmcjActivityInstance::inactive

Java(6) ExecutionState.INACTIVE

CheckedOut Indicates that the activity instance has been checked out.

ActiveX(64) AIState_CheckedOut

C-language(64) Fmc_AS_CheckedOut

C++(64) FmcjActivityInstance::checkedOut

Java(7) ExecutionState.CHECKED_OUT

InError Indicates that the activity instance has not been executed successfully.

ActiveX(128) AIState_InError

C-language(128) Fmc_AS_InError

C++(128) FmcjActivityInstance::inError

Java(8) ExecutionState.IN_ERROR

Executed Indicates that the activity instance has been executed.

ActiveX(256) AIState_Executed

C-language(256) Fmc_AS_Executed

C++(256) FmcjActivityInstance::executed

Java(9) ExecutionState.EXECUTED

Planning Indicates that the activity instance is in the planning state.

ActiveX(512) AIState_Planning

C-language(512) Fmc_AS_Planning

C++(512) FmcjActivityInstance::planning

Java(10) ExecutionState.PLANNING

ForceFinished Indicates that the activity instance is in the force-finished state.

ActiveX(1024) AIState_ForceFinished

C-language(1024) Fmc_AS_ForceFinished

C++(1024) FmcjActivityInstance::forceFinished

Java(11) ExecutionState.FORCE_FINISHED

Skipped Indicates that the activity instance has not been executed but skipped.

ActiveX(2048) AIState_Skipped

C-language(2048) Fmc_AS_Skipped

C++(2048) FmcjActivityInstance::skipped

Java(12) ExecutionState.SKIPPED

Deleted Indicates that the activity instance has been deleted.

ActiveX(4096) AIState_Deleted

C-language(4096) Fmc_AS_Deleted

C++(4096) FmcjActivityInstance::deleted

Java(13) ExecutionState.DELETED

Terminating Indicates that the activity instance is in the terminating state.

ActiveX(8192) AIState_Teminating

C-language(8192)
Fmc_AS_Terminating

C++(8192) FmcjActivityInstance::terminating

Java(14) ExecutionState.TERMINATING

Suspending Indicates that the activity instance is in the suspending state.

ActiveX(16384)
AIState_Suspending

C-language(16384)
Fmc_AS_Suspending

C++(16384) FmcjActivityInstance::suspending

Java(15) ExecutionState.SUSPENDING

Expired Indicates that the activity instance is in the expired state.

ActiveX(32768)
AIState_Expired

C-language(32768)
Fmc_AS_Expired

C++(32768) FmcjActivityInstance::expired

Java(16) ExecutionState.EXPIRED

Activity instance type

This enumeration is named ActivityInstanceType in ActiveX, FmcjActivityInstanceType in the C-language, FmcjActivityInstance::Type in C++ and com.ibm.workflow.api.ActivityInstancePackage.Type in Java. Possible values are:

NotSet(0) Indicates that nothing is known about the type of the activity instance.

ActiveX AIType_NotSet

C-language Fmc_AT_NotSet

C++ FmcjActivityInstance::NotSet

Java Type.NOT_SET

Process(1) Indicates that the activity instance is implemented by a process.

ActiveX AIType_Process

	C-language	Fmc_AT_Process
	C++	FmcjActivityInstance::Process
	Java	Type.PROCESS
Program(2)	Indicates that the activity instance is implemented by a program.	
	ActiveX	AIType_Program
	C-language	Fmc_AT_Program
	C++	FmcjActivityInstance::Program
	Java	Type.PROGRAM
Block	Indicates that the activity instance is implemented by a block.	
	ActiveX(16)	AIType_Block
	C-language(16)	Fmc_AT_Block
	C++(16)	FmcjActivityInstance::Block
	Java(3)	Type.BLOCK

Connector state

This enumeration is named ConnectorState in ActiveX, FmcjControlConnectorInstanceStateValue in the C-language, FmcjControlConnectorInstance::state in C++ and com.ibm.workflow.api.ControlConnectorInstancePackage.EvaluationState in Java. Possible values are:

False(0)	Indicates that evaluation of the control connector resulted in False.	
	ActiveX	ConnectorState_False
	C-language	Fmc_CS_False
	C++	FmcjControlConnectorInstance::False
	Java	EvaluationState.IS_FALSE
True(1)	Indicates that evaluation of the control connector resulted in True.	
	ActiveX	ConnectorState_True
	C-language	Fmc_CS_True
	C++	FmcjControlConnectorInstance::True
	Java	EvaluationState.IS_TRUE

NotEvaluated(2)
Indicates that the control connector has not yet been evaluated.

ActiveX	ConnectorState_NotEvaluated
C-language	Fmc_CS_NotEvaluated
C++	FmcjControlConnectorInstance::NotEvaluated
Java	EvaluationState.NOT_EVALUATED

NotSet(3) Indicates that nothing is known about the evaluation of the control connector.

ActiveX	ConnectorState_NotSet
C-language	Fmc_CS_NotSet
C++	FmcjControlConnectorInstance::NotSet
Java	EvaluationState.NOT_SET

Connector type

This enumeration is named ConnectorType in ActiveX, FmcjControlConnectorInstanceType in the C-language, FmcjControlConnectorInstance::Type in C++ and com.ibm.workflow.api.ControlConnectorInstancePackage.Type in Java. Possible values are:

NotSet(0) Indicates that nothing is known about the type of the control connector instance.

ActiveX	ConnectorType_Undefined
C-language	Fmc_CT_NotSet
C++	FmcjControlConnectorInstance::Undefined
Java	Type.UNDEFINED

Condition(1) Indicates that the control connector instance is a connector which can have a transition condition.

ActiveX	ConnectorType_Condition
C-language	Fmc_CT_Condition
C++	FmcjControlConnectorInstance::Condition
Java	Type.CONDITION

Otherwise(2) Indicates that the control connector instance is the “otherwise” connector.

ActiveX	ConnectorType_Otherwise

C-language Fmc_CT_Otherwise

C++ FmcjControlConnectorInstance::Otherwise

Java Type.OTHERWISE

Execution data kind

This enumeration is called FmcjExecutionDataKindEnum in the C-language and FmcjExecutionData::KindEnum in C++. Possible values are:

NotSet(0) Indicates that nothing is known about the type of the execution data.

ActiveX not applicable

C-language Fmc_DART_NotSet

C++ FmcjExecutionData::NotSet

Java not supported

Terminate(2) Indicates that receiving execution data can end.

ActiveX not applicable

C-language Fmc_DART_Terminate

C++ FmcjExecutionData::Terminate

Java not supported

ItemDeleted(1000)
Indicates that the execution data describes the deletion of a work item or notification.

ActiveX not applicable

C-language Fmc_DART_ItemDeleted

C++ FmcjExecutionData::ItemDeleted

Java not supported

Workitem(1002)
Indicates that the execution data describes the creation or update of a work item.

ActiveX not applicable

C-language Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

Java not supported

ActivityInstanceNotification(1003)
Indicates that the execution data describes the creation or update of an activity instance notification.

ActiveX not applicable

C-language Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

Java not supported

ProcessInstanceNotification(1004)
Indicates that the execution data describes the creation or update of a process instance notification.

ActiveX not applicable

C-language Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

Java not supported

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.

ActiveX not applicable

C-language Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

Java not supported

ExecuteProgramResponse(1101)
Indicates that the execution data describes the answer to an asynchronous request which asked for the execution of a program.

ActiveX not applicable

C-language Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

Java not supported

Execution mode

This enumeration is named ExeMode in ActiveX, FmcjProgramTemplateExeMode in the C-language and FmcjProgramTemplate::ExeMode in C++.

NotSet(0) Indicates that nothing is known about the execution mode.

ActiveX ExeMode_NotSet

C-language Fmc_GM_NotSet

C++ FmcjProgramTemplate::NotSet

Normal(1) Indicates that the program does not participate in global transactions.

ActiveX ExeMode_Normal

C-language Fmc_GM_Normal

C++ FmcjProgramTemplate::Normal

Safe(2) Indicates that the program participates in global transactions.

ActiveX ExeMode_Safe

C-language Fmc_GM_Safe

C++ FmcjProgramTemplate::Safe

Execution user

This enumeration is named ExeUser in ActiveX, FmcjProgramTemplateExeUser in the C-language and FmcjProgramTemplate::ExeUser in C++. Possible values are:

NotSet(0) Indicates that nothing is known about the execution user.

ActiveX ExeUser_NotSet

C-language Fmc_GU_NotSet

C++ FmcjProgramTemplate::NotSpecified

Agent(1) Indicates that the program executes under the identifier of the program execution agent.

ActiveX ExeUser_Agent

C-language Fmc_GU_Agent

C++ FmcjProgramTemplate::Agent

Starter(2) Indicates that the program executes under the user ID of the starter of the program.

ActiveX ExeUser_Starter

C-language Fmc_GU_Starter

C++ FmcjProgramTemplate::Starter

EXE options style

This enumeration is named ExeOptionsStyle in ActiveX, FmcjExeOptionsStyle in the C-language, FmcjExeOptions::Style in C++ and com.ibm.workflow.api.ProgramDataPackage.Style in Java. Possible values are:

NotSet(0) Indicates that nothing is known about the style of the EXE.

	ActiveX	EOStyle_NotSet
	C-language	Fmc_EO_NotSet
	C++	FmcjExeOptions::NotSet
	Java	Style.NOT_SET
Visible(1)	Indicates that the EXE should start visibly.	
	ActiveX	EOStyle_Visible
	C-language	Fmc_EO_Visible
	C++	FmcjExeOptions::Visible
	Java	Style.VISIBLE
Invisible(2)	Indicates that the EXE should start invisibly.	
	ActiveX	EOStyle_Invisible
	C-language	Fmc_EO_Invisible
	C++	FmcjExeOptions::Invisible
	Java	Style.INVISIBLE
Minimized(3)	Indicates that the EXE should start minimized.	
	ActiveX	EOStyle_Minimized
	C-language	Fmc_EO_Minimized
	C++	FmcjExeOptions::Minimized
	Java	Style.MINIMIZED
Maximized(4)	Indicates that the EXE should start maximized.	
	ActiveX	EOStyle_Maximized
	C-language	Fmc_EO_Maximized
	C++	FmcjExeOptions::Maximized
	Java	Style.MAXIMIZED

External service options time period

This enumeration is named ExternalOptionsTimePeriod in ActiveX, FmcjExternalOptionsTimePeriod in the C-language, FmcjExternalOptions::TimePeriod in C++ and com.ibm.workflow.api.ProgramDataPackage.TimePeriod in Java. Possible values are:

NotSet(0)	Indicates that nothing is known about an external service timeout.	
	ActiveX	TimePeriod_NotSet

C-language Fmc_EX_NotSet

C++ FmcjExternalOptions::NotSet

Java TimePeriod.NOT_SET

TimeInterval(1)
Indicates that the program execution agent should wait a specified time interval for the answer of the started external service.

ActiveX TimePeriod_TimeInterval

C-language Fmc_EX_TimeInterval

C++ FmcjExternalOptions::TimeInterval

Java TimePeriod.TIME_INTERVAL

Forever(2) Indicates that the program execution agent should wait forever for the answer of the started external service, that is, whatever time it takes.

ActiveX TimePeriod_Forever

C-language Fmc_EX_Forever

C++ FmcjExternalOptions::Forever

Java TimePeriod.FOREVER

Never(3) Indicates that the program execution agent should not wait for an answer of the started external service.

ActiveX TimePeriod_Never

C-language Fmc_EX_Never

C++ FmcjExternalOptions::Never

Java TimePeriod.NEVER

Implementation data basis

This enumeration is called ImplementationDataBasis in ActiveX, FmcjImplementationDataBasis in the C-language, FmcjImplementationData::Basis in C++ and com.ibm.workflow.api.ProgramDataPackage.Basis in Java. Possible values are:

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

ActiveX Basis_NotSpecified

C-language Fmc_DP_NotSet

C++ FmcjImplementationData::NotSpecified

	Java	Basis.NOT_SET
OS2(1)	Indicates that the program is an OS/2 program.	
	ActiveX	Basis_OS2
	C-language	Fmc_DP_OS2
	C++	FmcjImplementationData::OS2
	Java	Basis.OS2
AIX(2)	Indicates that the program is an AIX program.	
	ActiveX	Basis_AIX
	C-language	Fmc_DP_AIX
	C++	FmcjImplementationData::AIX
	Java	Basis.AIX
HPUX(3)	Indicates that the program is an HP-UX program.	
	ActiveX	Basis_HPUX
	C-language	Fmc_DP_HPUX
	C++	FmcjImplementationData::HPUX
	Java	Basis.HPUX
Windows95(4)	Indicates that the program is a Windows 95, Windows 98, or Windows Me program.	
	ActiveX	Basis_Windows95
	C-language	Fmc_DP_Windows95
	C++	FmcjImplementationData::Windows95
	Java	Basis.WINDOWS_95
WindowsNT(5)	Indicates that the program is a Windows NT or Windows 2000 program.	
	ActiveX	Basis_WindowsNT
	C-language	Fmc_DP_WindowsNT
	C++	FmcjImplementationData::WindowsNT
	Java	Basis.WINDOWS_NT
OS/390(6)	Indicates that the program is an OS/390 (R) program.	
	ActiveX	Basis_OS390
	C-language	Fmc_DP_OS390

	C++	FmcjImplementationData::OS390
	Java	Basis.OS390
Solaris(7)	Indicates that the program is a Solaris program.	
	ActiveX	Basis_Solaris
	C-language	Fmc_DP_Solaris
	C++	FmcjImplementationData::Solaris
	Java	Basis.SOLARIS

Implementation data type

This enumeration is called ImplementationDataProgramType in ActiveX, FmcjImplementationDataType in the C-language, FmcjImplementationData::Type in C++ and com.ibm.workflow.api.ProgramDataPackage.Type in Java. Possible values are:

NotSet(0)	Indicates that nothing is known about the implementation.	
	ActiveX	IOProgramType_NotSet
	C-language	Fmc_DT_NotSet
	C++	FmcjImplementationData::NotSet
	Java	ImplementationData.NOT_SET
EXE(1)	Indicates that the program is an executable.	
	ActiveX	IOProgramType_EXE
	C-language	Fmc_DT_EXE
	C++	FmcjImplementationData::EXE
	Java	ImplementationData.EXE
DLL(2)	Indicates that the program is implemented by a dynamic link library.	
	ActiveX	IOProgramType_DLL
	C-language	Fmc_DT_DLL
	C++	FmcjImplementationData::DLL
	Java	ImplementationData.DLL
External	Indicates that the program is some external service.	
	ActiveX(4)	IOProgramType_External
	C-language(4)	Fmc_DT_External
	C++(4)	FmcjImplementationData::External

Java(3) ImplementationData.EXTERNAL

Item state

This enumeration is called State in ActiveX, FmcjItemStateValue in the C-language, FmcjItem::state in C++ and com.ibm.workflow.api.ItemPackage.ExecutionState in Java. Possible values are:

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

- **ActiveX** ItemState_Undefined
- **C-language** Fmc_IS_NotSet
- **C++** FmcjItem::undefined
- **Java** ExecutionState.UNDEFINED

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

- **ActiveX** ItemState_Ready
- **C-language** Fmc_IS_Ready
- **C++** FmcjItem::ready
- **Java** ExecutionState.READY

Running(2) Indicates that the item is in the running state.

- **ActiveX** ItemState_Running
- **C-language** Fmc_IS_Running
- **C++** FmcjItem::running
- **Java** ExecutionState.RUNNING

Finished Indicates that the item is in the finished state.

- **ActiveX(4)** ItemState_Finished
- **C-language(4)** Fmc_IS_Finished
- **C++(4)** FmcjItem::finished
- **Java(3)** ExecutionState.FINISHED

Terminated Indicates that the item is in the terminated state.

- **ActiveX(8)** ItemState_Terminated
- **C-language(8)** Fmc_IS_Terminated
- **C++(8)** FmcjItem::terminated
- **Java(4)** ExecutionState.TERMINATED

Suspended Indicates that the item is in the suspended state.

- **ActiveX(16)** ItemState_Suspended

C-language(16) Fmc_IS_Suspended

C++(16) FmcjItem::suspended

Java(5) ExecutionState.SUSPENDED

Disabled Indicates that the item is disabled.

ActiveX(32) ItemState_Disabled

C-language(32) Fmc_IS_Disabled

C++(32) FmcjItem::disabled

Java(6) ExecutionState.DISABLED

CheckedOut Indicates that the item is checked out.

ActiveX(64) ItemState_CheckedOut

C-language(64) Fmc_IS_CheckedOut

C++(64) FmcjItem::checkedOut

Java(7) ExecutionState.CHECKED_OUT

InError Indicates that the item is in the InError state.

ActiveX(128) ItemState_InError

C-language(128) Fmc_IS_InError

C++(128) FmcjItem::inError

Java(8) ExecutionState.IN_ERROR

Executed Indicates that the item has been executed.

ActiveX(256) ItemState_Executed

C-language(256) Fmc_IS_Executed

C++(256) FmcjItem::Executed

Java(9) ExecutionState.EXECUTED

Planning Indicates that the item is in the planning state.

ActiveX(512) ItemState_Planning

C-language(512) Fmc_IS_Planning

C++(512) FmcjItem::Planning

Java(10) ExecutionState.PLANNING

ForceFinished Indicates that the item has been force-finished.

ActiveX(1024) ItemState_ForceFinished

C-language(1024) Fmc_IS_ForceFinished

C++(1024) FmcjItem::ForceFinished

Java(11) ExecutionState.FORCE_FINISHED

Deleted Indicates that the item has been deleted.

ActiveX(4096) ItemState_Deleted

C-language(4096) Fmc_IS_Deleted

C++(4096) FmcjItem::Deleted

Java(12) ExecutionState.DELETED

Terminating Indicates that the item is in the terminating state.

ActiveX(8192) ItemState_Terminating

C-language(8192) Fmc_IS_Terminating

C++(8192) FmcjItem::Terminating

Java(13) ExecutionState.TERMINATING

Suspending Indicates that the item is in the suspending state.

ActiveX(16384) ItemState_Suspending

C-language(16384) Fmc_IS_Suspending

C++(16384) FmcjItem::Suspending

Java(14) ExecutionState.SUSPENDING

Expired Indicates that the item is in the expired state.

ActiveX(32768) ItemState_Expired

C-language(32768) Fmc_IS_Expired

C++(32768) FmcjItem::Expired

Java(15) ExecutionState.EXPIRED

Item type

This enumeration is called Kind in ActiveX, FmcjItemType in the C-language, FmcjItem::ItemType in C++ and com.ibm.workflow.api.ItemPackage.ItemType in Java. Possible values are:

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

ActiveX	Kind_Unknown
C-language	Fmc_IT_NotSet
C++	FmcjItem::unknown
Java	ItemType.UNKNOWN

Workitem(1) Indicates that the item is a work item.

ActiveX	Kind_Workitem
C-language	Fmc_IT_Workitem
C++	FmcjItem::Workitem
Java	ItemType.WORK_ITEM

ProcessInstanceNotification
Indicates that the item is a process instance notification.

ActiveX(3)	Kind_ProcessInstanceNotification
C-language(3)	Fmc_IT_ProcessInstanceNotification
C++(3)	FmcjItem::ProcessInstanceNotification
Java(2)	ItemType.PROCESS_INSTANCE_NOTIFICATION

FirstActivityInstanceNotification
Indicates that the item is the first activity instance notification.

ActiveX(4)	Kind_FirstActivityInstanceNotification
C-language(4)	Fmc_IT_FirstActivityInstanceNotification
C++(4)	FmcjItem::FirstActivityInstanceNotification
Java(3)	ItemType.FIRST_ACTIVITY_INSTANCE_NOTIFICATION

SecondActivityInstanceNotification
Indicates that the item is the second activity instance notification.

ActiveX(5)	Kind_SecondActivityInstanceNotification
C-language(5)	Fmc_IT_SecondActivityInstanceNotification
C++(45)	FmcjItem::SecondActivityInstanceNotification

	Java(4)	ItemType.SECOND_ACTIVITY_INSTANCE_NOTIFICATION

Persistent list type

This enumeration is named TypeOfList in ActiveX, FmcjPersistentListTypeOfList in the C-language, FmcjPersistentList::TypeOfList in C++ and com.ibm.workflow.api.PersistentListPackage.TypeOfList in Java. Possible values are:

NotSet(0)	Indicates that nothing is known about the list type.	
	ActiveX	TypeOfList_NotSet
	C-language	Fmc_LT_NotSet
	C++	FmcjPersistentList::NotSet
	Java	TypeOfList.NOT_SET
Public(1)	Indicates that the list definition is for public usage.	
	ActiveX	TypeOfList_Public
	C-language	Fmc_LT_Public
	C++	FmcjPersistentList::Public
	Java	TypeOfList.PUBLIC
Private	Indicates that the list definition is for private usage.	
	ActiveX(3)	TypeOfList_Private
	C-language(3)	Fmc_LT_Private
	C++(3)	FmcjPersistentList::Private
	Java(2)	TypeOfList.PRIVATE

Process instance escalation

This enumeration is called PIEscalation in ActiveX, FmcjProcessInstanceEscalation in the C-language, FmcjProcessInstance::Escalation in C++ and com.ibm.workflow.api.ProcessInstancePackage.Escalation in Java. Possible values are:

NotSet(0)	Indicates that it is not known whether there is a notification on the process instance.	
	ActiveX	PIEscalation_NotSet
	C-language	Fmc_PE_NotSet
	C++	FmcjProcessInstance::NotSet
	Java	Escalation.NOT_SET

NoNotification(1)
Indicates that no notification occurred so far on the process instance.

ActiveX	PIEscalation_NoNotification
C-language	Fmc_PE_NoNotification
C++	FmcjProcessInstance::NoNotification
Java	Escalation.NO_NOTIFICATION

ProcessInstanceNotification
Indicates that a process instance notification occurred.

ActiveX(3)	PIEscalation_ProcessNotification
C-language(3)	Fmc_PE_ProcessNotification
C++(3)	FmcjProcessInstance::ProcessNotification
Java(2)	Escalation.PROCESS_NOTIFICATION

Process instance state

This enumeration is called ProcInstanceState in ActiveX, FmcjProcessInstanceStateValue in the C-language, FmcjProcessInstance::state in C++ and com.ibm.workflow.api.ProcessInstancePackage.ExecutionState in Java. Possible values are:

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

ActiveX	State_Undefined
C-language	Fmc_PS_NotSet
C++	FmcjProcessInstance::undefined
Java	ExecutionState.UNDEFINED

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

ActiveX	State_Ready
C-language	Fmc_PS_Ready
C++	FmcjProcessInstance::ready
Java	ExecutionState.READY

Running(2) Indicates that the process instance is in the running state.

ActiveX	State_Running
C-language	Fmc_PS_Running
C++	FmcjProcessInstance::running

	Java	ExecutionState.RUNNING
Finished	Indicates that the process instance is in the finished state.	
	ActiveX(4)	State_Finished
	C-language(4)	Fmc_PS_Finished
	C++(4)	FmcjProcessInstance::finished
	Java(3)	ExecutionState.FINISHED
Terminated	Indicates that the process instance is in the terminated state.	
	ActiveX(8)	State_Terminated
	C-language(8)	Fmc_PS_Terminated
	C++(8)	FmcjProcessInstance::terminated
	Java(4)	ExecutionState.TERMINATED
Suspended	Indicates that the process instance is in the suspended state.	
	ActiveX(16)	State_Suspended
	C-language(16)	Fmc_PS_Suspended
	C++(16)	FmcjProcessInstance::suspended
	Java(5)	ExecutionState.SUSPENDED
Terminating	Indicates that the process instance is in the terminating state.	
	ActiveX(32)	State_Terminating
	C-language(32)	Fmc_PS_Terminating
	C++(32)	FmcjProcessInstance::terminating
	Java(6)	ExecutionState.TERMINATING
Suspending	Indicates that the process instance is in the suspending state.	
	ActiveX(64)	State_Suspending
	C-language(64)	Fmc_PS_Suspending
	C++(64)	FmcjProcessInstance::suspending
	Java(7)	ExecutionState.SUSPENDING
Deleted	Indicates that the process instance is in the deleted state.	
	ActiveX(128)	State_Deleted

C-language(128)
Fmc_PS_Deleted

C++(128) FmcjProcessInstance::deleted

Java(8) ExecutionState.DELETED

Work item program retrieval

This enumeration is named WorkitemProgramRetrieval in ActiveX, FmcjWorkitemProgramRetrieval in the C-language, FmcjWorkitem::ProgramRetrieval in C++ and com.ibm.workflow.api.WorkItemPackage.ProgramRetrieval in Java. Possible values are:

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

ActiveX WIProgramRetrieval_NotSet

C-language Fmc_WS_NotSet

C++ FmcjWorkitem::NotSet

Java ProgramRetrieval.NOT_SET

CommonDataOnly(1)
Indicates that the common parts of program definitions are to be retrieved.

ActiveX WIProgramRetrieval_CommonDataOnly

C-language Fmc_WS_CommonDataOnly

C++ FmcjWorkitem::CommonDataOnly

Java ProgramRetrieval.COMMON_DATA_ONLY

SpecifiedDefinitions(2)
Indicates that the specified program definitions are to be retrieved.

ActiveX WIProgramRetrieval_SpecifiedDefinitions

C-language Fmc_WS_SpecifiedDefinitions

C++ FmcjWorkitem::SpecifiedDefinitions

Java ProgramRetrieval.SPECIFIED_DEFINITIONS

AllDefinitions
Indicates that all program definitions are to be retrieved.

ActiveX(4) WIProgramRetrieval_AllDefinitions

C-language(4) Fmc_WS_AllDefinitions

C++(4) FmcjWorkitem::AllDefinitions

Java(3) ProgramRetrieval.ALL_DEFINITIONS

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. In the Java language, an Integer object is returned for optional integer values.

ActiveX signature

```
long Property()
```

C-language signature

```
long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )

unsigned long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ language signature

```
long Property() const

unsigned long Property() const
```

Java signature

```
public abstract int property() throws FmcException

public abstract Integer property() throws FmcException
```

Parameters

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

Return type

long/unsigned long/int
The property value.

Declaration examples

ActiveX long Priority();

C-language unsigned long FMC_APIENTRY FmcjWorkitemPriority(FmcjWorkitemHandle handle);

C++ unsigned long Priority() const;

Java public int priority() throws FmcException;

public java.lang.Integer threshold() 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.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signature

```
BSTR Property()
```

C-language signature

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

C++ language signature

```
string Property() const
```

Java signature

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

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

BSTR/char*/string/String

The property value.

Declaration examples

ActiveX	BSTR Description();
C-language	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 the C++ and C-language, and as an array in ActiveX and 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-language vectors" on page 35).

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

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

ActiveX signature

```
void Property( ValueTypeArray * value )
```

C-language signature

```
FmcjValueTypeVectorHandle
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ language signature

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

Java signature

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

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

ActiveX	void Staff(StringArray * staff);
C-language	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.

ActiveX signature

```
Object  Property()
```

C-language signature

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

C++ language signature

```
FmcjObject Property() const
```

Java signature

```
public abstract Object property() throws FmcException

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

public abstract
ExecutionAgent getExecutionAgent() throws FmcException

public
ReadOnlyContainer value() throws FmcException
```

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.

ReadOnlyContainer
The read-only container described by the holder object.

Declaration examples

ActiveX	fmcError ErrorReason();
C-language	FmcjErrorHandle FMC_APIENTRY FmcjWorkitemErrorReason(FmcjWorkitemHandle handle);
C++	FmcjError ErrorReason() const;
Java	public abstract FmcError errorReason() throws FmcException;

Accessing a pointer valued property

Returns the value of a property which is a pointer to some object.

ActiveX signature

```
Object * Property()
```

C-language signature

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

C++ language signature

```
FmcjObject * Property() const
```

Java signature

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

Parameters

handle

Input. The handle of the object to be queried.

Return type

Object*/Handle/FmcjObject*

A pointer or handle to the object respectively the object itself.

Declaration examples

ActiveX	Container * InContainer();
C-language	FmcjReadOnlyContainerHandle FMC_APIENTRY FmcjProgramDataInContainer(FmcjProgramDataHandle handle);
	FmcjProcessInstanceHandle FMC_APIENTRY FmcjExecutionServicePersistentProcessInstance(FmcjExecutionService service, char const * oid);
C++	FmcjReadOnlyContainer * InContainer() const;
	FmcjProcessInstance * PersistentProcessInstance(string const & oid) const;
Java	public abstract ReadOnlyContainer inContainer() throws FmcException;

```
public abstract ProcessInstance persistentProcessInstance(
String oid ) 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.

ActiveX signature

```
boolean PropertyIsNull()
```

C-language signature

```
bool FMC_APIENTRY FmcjXxxPropertyIsNull( FmcjXxxHandle handle )
```

C++ language signature

```
bool PropertyIsNull() const
```

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

ActiveX boolean DescriptionIsNull();

C-language 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. In the Java language, an Integer object is to be passed for optional integer values.

ActiveX signature

```
void SetProperty( long newValue )
```

C-language signature

```
void FMC_APIENTRY FmcjXxxSetProperty( FmcjXxxHandle  handle,
                                      long           newValue );
```

C++ language signature

```
void SetProperty( long newValue );
```

Java signature

```
public abstract void setProperty( int newValue ) throws FmcException

public abstract void setProperty(Integer newValue) throws FmcException
```

Parameters

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

newValue Input. The new value of the property.

Declaration examples

ActiveX void SetTimeout(long newValue);

C-language void FMC_APIENTRY FmcjExecutionServiceSetTimeout(FmcjExecutionServiceHandle handle, long newValue);

C++ void SetTimeout(long newValue) const;

Java public void SetTimeout(int newValue) throws FmcException;

public void setThreshold(Integer threshold) throws FmcException;

An example is the FmcjService::SetTimeout API call which sets the timeout value for requests issued by the client to an MQ 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 wanted. 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.

Note: It is possible that, even though FMC_ERROR_TIMEOUT is returned when you issue a client-server call, the MQ 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 a value of type string

This API call sets the specified property to the specified value.

ActiveX signature

```
long SetProperty( BSTR newValue )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjXxxSetProperty( FmcjXxxHandle  handle,
                                        char const *   newValue );
```

C++ language signature

```
APIRET SetProperty( string const & newValue );
```

Java signature

```
public abstract void setProperty( String newValue ) throws FmcException
```

Parameters

handle Input. The handle of the object to be changed.
newValue Input. The new value of the property.

Declaration examples

ActiveX long SetSessionContext(BSTR userID, BSTR sessionID);

C-language

```
APIRET FMC_APIENTRY FmcjExecutionServiceSetSessionContext(
                    FmcjExecutionServiceHandle handle,
                    char const *               userID,
                    char const *               sessionID );
```

C++ APIRET SetSessionContext(string const & userID, string const & sessionID);

Java public void setSessionContext(String userID, String sessionID) throws FmcException;

Setting an object valued property

This API call sets the specified property to the specified object.

Java signature

```
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 communication protocol, providing this information is necessary.

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 MQ 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-language signature

```
APIRET FMC_APIENTRY FmcjXxxUpdate( FmcjXxxHandle          handle,
                                   FmcjExecutionDataHandle data );
```

C++ language signature

```
APIRET Update( FmcjExecutionData const & data );
```

Parameters

handle Input. The handle of the object to be updated.
data Input. The data which is to be used for the update.

Return codes

The C-language functions and the MQ 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

C-language 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;
```

Figure 9. Accessing values in C (Part 1 of 9)

```
 FmcjGlobalConnect();
```

Figure 9. Accessing values in C (Part 2 of 9)

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

Figure 9. Accessing values in C (Part 3 of 9)

```
                                       /* set the timeout for requests */
  FmcjExecutionServiceSetTimeout( service, 60000 );
```

Figure 9. Accessing values in C (Part 4 of 9)

```
/* 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 );
 }
```

Figure 9. Accessing values in C (Part 5 of 9)

```
                                    /* access a date/time value     */
 startTime= FmcjWorkitemStartTime( workitem );
 printf( "Start time : %s\n",
   FmcjDateTimeAsString(&startTime, generalBuffer, 200) );
```

Figure 9. Accessing values in C (Part 6 of 9)

```
                                    /* access a value of type long  */
 priority = FmcjWorkitemPriority( workitem );
 printf( "Priority   : %u\n", priority );
```

Figure 9. Accessing values in C (Part 7 of 9)

```
                                   /* 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) );
  }
```

Figure 9. Accessing values in C (Part 8 of 9)

```
  /* logoff */
  FmcjExecutionServiceLogoff(service);
  FmcjExecutionServiceDeallocate(&service);
  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

Figure 9. Accessing values in C (Part 9 of 9)

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 );
```

Figure 10. Accessing values in C++ (Part 1 of 6)

```
  // 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;
  }
```

Figure 10. Accessing values in C++ (Part 2 of 6)

```
                                        // access a value of type date/time
  cout << "Start time : " << workitem.StartTime()<< endl;
```

Figure 10. Accessing values in C++ (Part 3 of 6)

```
                                        // access a value of type long
  cout << "Priority   : " << workitem.Priority()<< endl;
```

Figure 10. Accessing values in C++ (Part 4 of 6)

```
                                       // access an enumerated value
  FmcjItem::AssignReason reason= workitem.ReceivedAs();
  cout << "Received as: " <<
          ((reason == FmcjItem::Normal) ? "normal user" : "...")
       << endl;
```

Figure 10. Accessing values in C++ (Part 5 of 6)

```
  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 10. Accessing values in C++ (Part 6 of 6)

Action API calls

Action API calls are client-server calls, involving communication with an MQ Workflow server. As such, they require 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 MQ 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 MQ Workflow server.

All action API calls are described separately in "Part 7. API action and activity implementation calls" on page 339. You can find examples in "Part 9. Examples and scenarios" on page 753.

Activity implementation API calls

An activity or support tool can be implemented by a program which uses the MQ Workflow API. In this case, the activity implementation API calls provide access to information like the program identification by which the running program is known to the program execution agent or to the input and output containers of the activity instance or work item or of the input container of the support tool. They also allow the program implementing an activity to return the updated output container to MQ Workflow so that navigation can continue on the basis of those values.

A program implementing an activity or support tool is usually executed under the control of an MQ Workflow program execution agent on request from some MQ Workflow execution server. When an MQ Workflow execution server receives a request to start a work item or support tool, it determines the implementing program to be started and sends an appropriate request together with the input and output containers, if needed, to the logged-on user's MQ Workflow program execution agent. Since containers are sent to the program execution agent, input and output containers are requested from and returned to an MQ Workflow program execution agent by the implementing program. You do *not* have to create an execution service object and log on to an MQ Workflow execution server to handle containers from within an activity implementation or support tool.

However, if you want to access not only containers, for example, if you want to query information about the process instance the work item is a part of, you have to log on to the MQ Workflow execution server that requested to start your program. 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. 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.

An MQ Workflow program execution agent can run more than one program at a time. When a container is requested, it determines the calling program and provides the container sent by the server for this program's usage.

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 agent cannot know the calling program. For that purpose, the program calling the program execution agent must provide the program identification of the actual 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 agent only provides the program identification to *trusted* programs.

Accessing general information

As said, an activity implementation or support tool can retrieve the program identification by which it is known to the program execution agent (class name FmcjPEA or ExecutionAgent in Java). Furthermore, an activity implementation or a program who knows the program identification can query the user on whose behalf the activity implementation was started or the persistent object identification of the associated activity instance.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

ActiveX signatures

```
BSTR ProgramID()

BSTR PersistentOidOfActivityInstance()
BSTR RemotePersistentOidOfActivityInstance( BSTR programID )

BSTR RemoteUserID( BSTR programID )
BSTR UserID()
```

C-language signatures

```
char * FMC_APIENTRY FmcjPEAProgramID(
                           char *         buffer,
                           unsigned long  bufferLength )

char * FMC_APIENTRY FmcjPEAPersistentOidOfActivityInstance(
                           char *         buffer,
                           unsigned long  bufferLength )

char * FMC_APIENTRY FmcjPEARemotePersistentOidOfActivityInstance(
                           char const *   programID,
                           char *         buffer,
                           unsigned long  bufferLength )

char * FMC_APIENTRY FmcjPEAUserID(
                           char *         buffer,
                           unsigned long  bufferLength )

char * FMC_APIENTRY FmcjPEARemoteUserID(
                           char const *   programID,
                           char *         buffer,
                           unsigned long  bufferLength )
```

C++ language signatures

```
static string ProgramID()

static string PersistentOidOfActivityInstance()
static string RemotePersistentOidOfActivityInstance(
                               string const & programID )

static string RemoteUserID(   string const & programID )
static string UserID()
```

Java signatures

```
public abstract
String programID() throws FmcException

public abstract
String persistentOidOfActivityInstance() throws FmcException

public abstract
String remotePersistentOidOfActivityInstance( String programID )
throws FmcException

public abstract
String userID() throws FmcException

public abstract
String remoteUserID( String programID ) throws FmcException
```

Parameters

buffer Input/Output. A pointer to a buffer to contain the value to be retrieved.

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.

programID Input. The program ID by which the activity implementation is known to the program execution agent.

Return type

BSTR/char*/string/String
The value.

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 buffer is expected, but 0 is passed.

FMC_ERROR_BUFFER(800)
The buffer provided is too small to hold the largest possible value. See file fmcmxcon.h for required lengths.

FMC_ERROR_COMMUNICATION(13)
The program execution agent cannot be contacted.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative. Also, check the MQ Workflow trace for any exceptions encountered.

FMC_ERROR_INVALID_PROGRAMID(135)
The specified program ID is invalid or unknown.

FMC_ERROR_PROGRAM_EXECUTION(126)
The API call is not called from within an activity implementation or support tool.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call. For example, the activity implementation is not trusted and thus cannot retrieve its program ID.

FMC_ERROR_TOOL_FUNCTION(128)
The API call is not supported from within a support tool.

Dynamic link libraries

Beside being an Executable, the activity implementation or support tool program can also be a dynamic link library (DLL) or shared library, named DLL in the following discussion.

If it is a DLL, it can execute in fenced or non-fenced mode. If fenced, the DLL is executed in an operating system process different from the program execution agent process. If non-fenced, the DLL is executed in the program execution agent's own operating system process.

A DLL signature looks as follows; in C++ use the *extern "C"* construct:

C-language signature

```
int FMC_APIENTRY entryPoint( char const * arguments )
```

Parameters
arguments
Input. The arguments to be passed to the program.

In the FlowMark Version 2 compatibility mode, a DLL signature looks as follows:

C-language signature

```
int FMC_APIENTRY entryPoint( char const * programID,
                             char const * arguments )
```

Parameters

programID
Input. The program ID (formerly called session ID) by which the program is known to the program execution agent.

arguments
Input. The arguments to be passed to the program.

A DLL can also specify a DLL initialization and/or a DLL termination routine. Immediately after the program execution agent loads a DLL, it calls the DLL initialization entry point, if available. And immediately before the program execution agent unloads a DLL, it calls the DLL termination entry point, if available.

C-language Signature

```
void FmcDllInit()
```

C-language Signature

```
void FmcDllTerm()
```

For example, consider a non-fenced DLL which is kept loaded. Then initialization could acquire resources which are held through the life time of the DLL until they are freed by the termination routine. Examples of objects you might want to acquire only once are sessions to resource managers or open file handles.

See “Chapter 72. An activity implementation” on page 801 or the ″API Programming Examples″ support pack for activity implementation examples.

Program execution management API calls

Program execution management API calls provide for the management of MQ Workflow program execution agents. They allow for a user-associated program execution agent to be started and to be stopped (shutdown).

Part 2. The C and C++ APIs

This part provides an overview of the concepts which are specific for the MQ Workflow C-language and C++ APIs.

Chapter 14. An MQ Workflow client application

An MQ Workflow C or C++ client application typically contains the following parts (which may not be delimited this clearly, however):

```
                #include <MQ Workflow Api-prerequisites (C++)>
                #include <MQ Workflow API>
                int main()
                {
Setup             Declare objects
                        :
                  Connect
                  Allocate service object
                   Logon ()

Actions           MQ Workflow API calls

                  Logoff()
                  Deallocate service object
Cleanup           Disconnect
                  return 0;
                }
```

To set up your program, you typically declare the program variables or objects you are going to use and you include the MQ 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 MQ Workflow API headers.

You should then initialize the MQ Workflow API by calling the Connect() API call so that resources held by the API are allocated correctly. Connect() - and Disconnect() - are to be called at the begin respectively end of each thread.

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 MQ 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 MQ Workflow API so that resources held by the API are deallocated correctly.

Chapter 15. An MQ Workflow activity implementation or support tool

An MQ Workflow C or C++ activity implementation or support tool implementation typically contains the following parts.

```
                #include <MQ Workflow Api-prerequisites (C++)>
                #include <MQ Workflow API>
                int main()
                {
Setup           {   Declare objects
                          :
                    Connect
                    InContainer()
                     OutContainer() if no support tool

                    read values
Actions         {   set values if no support tool

                    SetOutContainer() if no support tool
                    Deallocate object
Cleanup         {   Disconnect
                    return rc;
                }
```

To set up your program, you typically declare the program variables or objects you are going to use and you include the MQ 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 MQ Workflow API headers.

You should then initialize the MQ Workflow API by calling the Connect() API call so that resources held by the API are allocated correctly. Connect() - and Disconnect() - are to be called at the begin respectively end of each thread.

An activity implementation can then retrieve the activity's input and output containers from the MQ Workflow program execution agent that started this program. A support tool can retrieve the activity's input container only.

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 MQ Workflow program execution agent. Any resources held by your program are deallocated. The return value of your program tells the program execution agent about the overall outcome of your program.

The output container as well as the return code of your program are passed back to the MQ 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 MQ Workflow navigation. [5]

As a last step, you disconnect from the MQ Workflow API so that resources held by the API are deallocated correctly.

Your activity implementation can as well behave like a client application (see "Chapter 14. An MQ Workflow client application" on page 159) and request services from an MQ 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.

5. 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.

Chapter 16. Compiling and linking

All C++ and C-language programs developed for use with MQ Workflow must include header files provided by MQ Workflow and link the corresponding library files. These files have been installed on your system, if you selected to install the MQ Workflow Development Kit. They are installed by default:

- For AIX(R), the header files in the /usr/lpp/fmc/api directory; the shared library files in the /usr/lpp/fmc/lib directory.
- For HP-UX, the header files in the /opt/fmc/api directory; the shared library files in the /opt/fmc/lib directory. The shared libraries are linked to /usr/lib.
- For Solaris, the header files in the /opt/fmc/api directory; the shared library files in the /opt/fmc/lib directory. The shared libraries are linked to /usr/lib.
- For the Windows platforms, in the \Program Files\MQSeries Workflow\Api directory on your selected drive.

When using the MQ Workflow C++ API, definitions for bool, string, and vector must be provided. If your compiler supports these definitions, use the definitions of your compiler. Include the appropriate files before the MQ Workflow C++ API headers. In case that your compiler does not support any of these definitions, MQ Workflow delivers some files to be included in the stl subdirectory of the API: *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. See "C++ prerequisite header files" on page 165 which definitions must be included for the supported compilers.

Note that bool.h and vector.h are part of the Standard Template Library delivered with MQ Workflow and copyrighted[6] 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.

6. Copyright (c) 1994

Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided ″as is″ without express or implied warranty.

Note: In the Windows environments, MQ Workflow interprets any input in the ANSI code page.This means that there can be differences when MQ Workflow tests for a printable character and, for example, when an application uses a function like isprint() to test for a printable character.

The MQ 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-API Header	C++ Header
Runtime client	fmcjcrun.h	fmcjprun.hxx
Runtime activity implementation:		
- container access only	fmcjccon.h	fmcjpcon.hxx
- container and server access	fmcjcrun.h	fmcjprun.hxx
Runtime support tool		
- container access only	fmcjccon.h	fmcjpcon.hxx
- container and server access	fmcjcrun.h	fmcjprun.hxx

The MQ Workflow dynamic link libraries have been split accordingly.

fmcjdcom contains common functionality and must always be linked
fmcjdcbr contains templates and persistent lists
fmcjdcon contains container functionality
fmcjdrun contains Runtime functionality only, that is, deals with process instances, work items, notifications, and instance monitors

Such, the following libraries must be linked.

Feature	fmcjdcom	fmcjdcbr	fmcjdcon	fmcjdrun
Runtime client	x	x	x	x
Runtime activity implementation:				
- container access only	x		x	
- container and server access	x	x	x	x
Runtime support tool:				
- container access only	x		x	
- container and server access	x	x	x	x

All popular compilers can be used to compile and link your applications accessing the C++ and C-language MQ Workflow APIs. Your compile and link options must ensure that the MQ Workflow APIs are 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 should automatically be applied when you use the header files provided by MQ Workflow. You **must** use the multi-threaded libraries.

Access can be gained to C-language functions using calls from all languages that support C calls. Access can be gained to the C++ API from all popular C++ compilers since the C++ API is delivered as source code (inline methods).

Supported compilers

Supported in terms of maintenance are, however, only those compilers and environments listed below.

- For AIX:
  - IBM VisualAge C++ Professional for AIX, Version 5.0
  - IBM C for AIX, Version 5.0
- For HP-UX, HP aC++ Compiler S700 Version A.01.15.01
- For Solaris:
  - Sun WorkShop Compiler, Versions 4.2 and 5.0
  - For C++ only, Kuck&Associates Inc. KAI C++ Version 3.3
- For the Windows platforms:
  - IBM VisualAge for C++, Versions 3.5 and 4.0
  - Microsoft Visual C++, Versions 5.0 and 6.0

C++ prerequisite header files

The following table indicates for the C++ API whether definitions for bool, string, and vector are supplied by the supported compilers (compiler type or compiler provided include) or whether the MQ Workflow provided definitions have to be used:

Platform	bool	vector/string
AIX IBM VA 5.0	compiler type	compiler include
HP-UX	compiler type	compiler include
Solaris 4.2	compiler type	MQ Workflow
Solaris 5.0/KAI	compiler type	compiler include
Windows IBM VA 3.5	MQ Workflow	MQ Workflow
Windows IBM VA 4.0	compiler type	compiler include
Windows MSVC	compiler type	compiler include

Sample compile statements

Sample compile statements are:

- For AIX and IBM VisualAge C++ Professional for AIX 5.0:

```
xlC_r -o <executable> -I/usr/lpp/fmc/api -L/usr/lpp/fmc/lib
                              -l<MQ Workflow libs> <source file>
```

- For AIX and IBM C for AIX, Version 5.0:

```
Compile your program:
xlc_r -c -O -I/usr/lpp/fmc/api <source file>

Link your program using the shell script linkxlC_r provided in /usr/lpp/fmc/api/Csuppor
linkxlC_r -o <executable> <object file> -L/usr/lpp/fmc/lib
                                -l<MQ Workflow libs> -L/usr/lpp/xlC/lib -lC_r
```

- For HP-UX:

```
aCC -D_THREAD_SAFE -DRWSTD_MULTI_THREAD -D_REENTRANT
    -o <executable> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
```

- For Solaris and the C-language:

```
cc -o <executable> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
```

- For Solaris and the Sun Workshop C++ Compilers:

```
CC -mt -o <executable> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
```

- For Solaris and the KAI C++ Compiler:

```
KCC --thread_safe
    -o <executable> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
```

- For the Windows platforms and Microsoft Visual C++ 5.0 or 6.0:

```
cl -MD <optional parameters> <source file>
```

- For the Windows platforms and IBM VisualAge for C++ 3.5:

```
icc /GM+ /Su4 <optional parameters> <source file>
```

- For the IBM VisualAge for C++ 4.0 compiler, care that the multi-thread libraries are used and that an enum size of 4 bytes is specified.

Chapter 17. Memory management

Workflow process models, their instances, and resulting work items are all objects persistently stored in an MQ 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-language) or destructed (C++). Transient objects are, however, no longer available when your application program ends.

Some transient objects are *explicitly allocated* by you. These are supporting objects, which do not reflect persistent ones. Examples are the FmcjStringVector when you specify a set of persons to stand in for (C-language) 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 life time of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQ Workflow API. The MQ Workflow API provides a handle (C-language) 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 MQ Workflow to lazy read a collection of objects passed from the server and thus increases performance.

The MQ 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**.

As all resource memory is finally owned by the application process itself, you can access all objects from different threads within that process. MQ Workflow does not hinder you from using threads; it is coded reentrantly. On the other hand, MQ 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 on that object. Objects are **not** thread-safe.

Chapter 18. The result object

In general, a result object states the result of the last MQ Workflow API request (in the considered 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.

Although MQ Workflow does **not** explicitly support threads in that it manages the synchronization of objects (you have to care for that), MQ Workflow does not prohibit to use threads. That is why it provides for result objects on a per thread basis.

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 respectively the FmcjResult::ObjectOfCurrentThread() method. The result object is automatically updated with each request.

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

For example, in the C-language, you can access and use a result object in the following way:

```
#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 11. Accessing a result object in the C-language

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

For example, in the C++ language, you can access and use a result object the following way:

```
#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() ;
  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 12. 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.

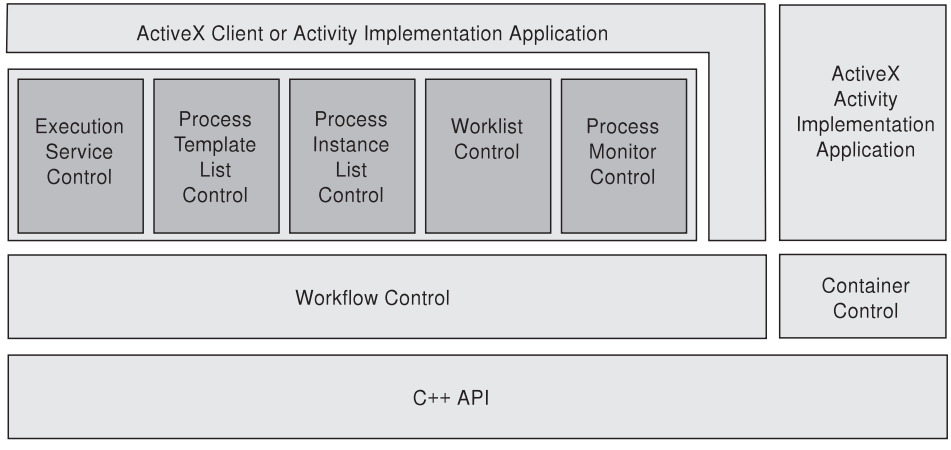
Part 3. ActiveX controls

This part provides for an overview on the MQ Workflow ActiveX controls.

Chapter 19. Component overview

MQ Workflow delivers several ActiveX controls which can be used to write client application programs or activity implementations and support tools. Following controls are provided:

- IBM MQSeries Workflow Control 3.1
- IBM MQSeries ExecutionService Control 3.1
- IBM MQSeries ProcessTemplateList Control 3.1
- IBM MQSeries ProcessInstanceList Control 3.1
- IBM MQSeries Worklist Control 3.1
- IBM MQSeries ProcessMonitor Control 3.1
- IBM MQSeries Container Control 3.1



The ActiveX API is implemented on top of the C++ API and serves as an access layer for the ActiveX controls to an execution server. The Workflow Control and the Container Control are the OLE interface to the C++ API layer. On top of the Workflow Control, you find all controls except the Container Control. All controls except the Container Control contain a Design-time GUI as well as a Runtime GUI. The Container control can be used by activity implementations and support tools just accessing containers. Note that the MQ Workflow Standard Runtime Client itself is implemented using the provided ActiveX controls - see also “Part 9. Examples and scenarios” on page 753.

Functional overview

The Workflow Control operates within a Visual Basic user application as follows:

- The Visual Basic user application usually contains one (non-visual) Workflow Control.
- The Workflow Control contains one ExecutionServiceArray.
- The ExecutionServiceArray can contain multiple ExecutionServices, and each ExecutionService is connected to one MQ Workflow execution server.
- Each ExecutionService contains one array for each of the following list types: ProcessTemplateList, ProcessInstanceList, and Worklist.
- Each of these arrays can contain multiple objects, that is, the WorklistArray can contain multiple worklists, each of which can be connected to a Worklist Control.
- You can have one or more (visual) ExecutionService Controls, connected to the Workflow Control, showing specific information of available execution services.
- You can have one or more (visual) controls, connected to the Workflow Control, showing the objects of specific lists.

Workflow Control overview

The Workflow Control contains several unique objects. The objects are directly maintained by the control. The ExecutionServiceArray object can create and maintain any number of ExecutionService objects. Each ExecutionService object handles a reference to an MQ Workflow execution server.

Each **ExecutionService** contains a **ProcessTemplateListArray**, a **ProcessInstanceListArray**, and a **WorklistArray**. For each of the arrays there are methods for adding, retrieving, and deleting array elements as well as determining the number of entries in the array.

Furthermore, a **StringArray** object is maintained by the Workflow Control. Objects maintained herein are to be used, for example, when support tools are queried for a workitem as in **Workitem::SupportTools** or when a list of Person IDs is queried as within **Workitem::Staff**.

There are also several enumeration types: AssignReason, Kind, or State. For example, State contains entries that correspond to the current state of an item (for example, Ready, Running, Disabled, or Suspended).

How to work with an ExecutionService

To work with an ExecutionService object, your program must have access to the WorkFlow Control OCX. In a Visual Basic programming environment this is accomplished by imbedding the specific OCX into one of the available forms. The Workflow Control allows you to access the ExecutionServiceArray. By using the methods **Add** or **AddDefault** you can create a new ExecutionService. You get access to the newly created ExecutionService object via the **GetAt** method.

Having access to a new ExecutionService object you can issue all methods provided by this object. There is no GUI involved in this process.

How to work with lists

To work with a list control, you must have created an ExecutionService object. To access, for example, all worklists you are authorized to see, you must fill the WorklistArray managed by the ExecutionService object. This is accomplished by the calling the **QueryWorklists()** method. Having done this, you must use the **WorklistArray()** method to get access to the object that contains the Worklist objects. To get access to an individual worklist object you can use the **GetAt()** method of the WorklistArray. All other lists are handled in the same way. See "ActiveX arrays" on page 40 for detailed information. There is no GUI involved in this process.

ProcessTemplateList Control overview

The ProcessTemplateList Control maintains the ProcessTemplate objects which can be viewed through the particular list. You can fill the array by using the **QueryProcessTemplates()** method of the ProcessTemplateList class. Using **GetSize()**, you can obtain the number of items within the list and to work with a particular ProcessTemplate object, you can use the **GetAt()** method.

ProcessInstanceList Control overview

The ProcessInstanceList Control maintains the ProcessInstance objects which can be viewed through the particular list. You can fill the array by using the **QueryProcessInstances()** method of the ProcessInstanceList class. Using **GetSize()**, you can obtain the number of items within the list and to work with a particular ProcessInstance object, you can use the **GetAt()** method.

Worklist Control overview

The Worklist Control maintains objects which can be viewed through the particular list, namely work items, activity instance notifications, or process instance notifications. It maintains an ActivityInstanceNotifArray, a ProcessInstanceNotifArray, and a WorkitemArray.

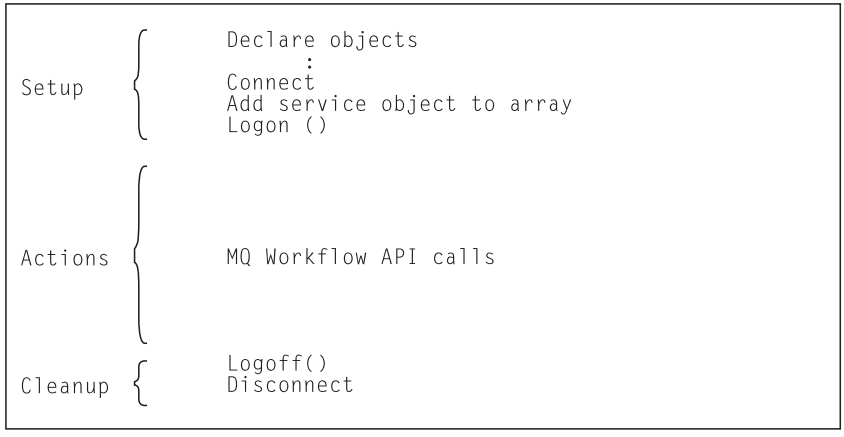
The WorkitemArray object, for example, can create and maintain any number of Workitem objects. There are methods for adding, retrieving, and deleting array elements as well as determining the number of entries in the array.

Monitor Control overview

The Monitor Control represents the monitor for a process instance or an activity instance. You can use the **ObtainMonitor()** methods in order to access a monitor.

Chapter 20. An MQ Workflow client application

An MQ Workflow ActiveX client application typically contains the following parts, not necessarily divided that clearly.



To set up your program, your MQ Workflow Control must be on the VisualBasic form. You then typically declare the program variables or objects you are going to use.

You should then initialize the MQ Workflow API by calling the Connect() method so that resources held by the API are allocated correctly. Connect() and Disconnect() are to be called at the begin respectively end of each thread.

You then need to allocate a service object which represents the server you are going to ask services from. You do this by adding the object to the execution service array provided for that purpose. 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.

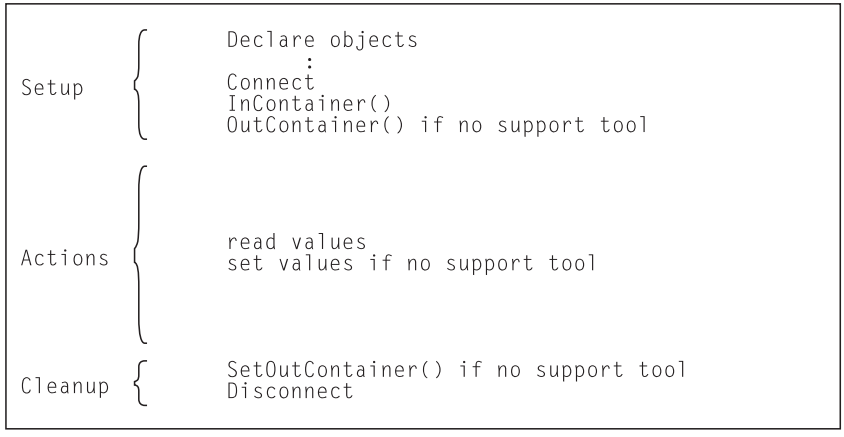
After a successful logon, you can issue action or program execution management methods in order to query or manage MQ Workflow objects you are authorized for.

At the end of your program, you log off in order to close the session to the server.

As a last step, you disconnect from the MQ Workflow API so that resources held by the API are deallocated correctly.

Chapter 21. An MQ Workflow activity implementation or support tool

An MQ Workflow ActiveX activity implementation or support tool implementation typically contains the following parts.



To set up your program, the Container Control must be on the VisualBasic form. You then typically declare the program variables or objects you are going to use.

You should then initialize the MQ Workflow API by calling the Connect() method so that resources held by the API are allocated correctly. Connect() - and Disconnect() - are to be called at the begin respectively end of each thread.

An activity implementation can then retrieve the activity's input and output containers from the MQ Workflow program execution agent that started this program. A support tool can retrieve the activity's input container only.

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 MQ Workflow program execution agent. The _RC value of your output container tells the execution server about the overall outcome of your program.

The output container is passed back to the MQ 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 MQ Workflow navigation.

As a last step, you disconnect from the MQ Workflow API so that resources held by the API are deallocated correctly.

Your activity implementation can as well behave like a client application (see "Chapter 14. An MQ Workflow client application" on page 159) and request services from an MQ 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.

Part 4. The JAVA API

The MQ Workflow Java API consists of:

- A set of API classes that provide MQSeries Workflow API functionality to Java based applications.
- An agent that connects an MQSeries Workflow domain to the Java world.

For a detailed overview of the Java API Classes and the Java CORBA Agent, refer to the *IBM MQSeries Workflow: Installation Guide*.

Chapter 22. Threading considerations for the Java CORBA Agent

The MQ Workflow C and C++ programming languages offer API calls FmcjGlobalConnect()/FmcjGlobal::Connect() and FmcjGlobalDisconnect()/FmcjGlobal::Disconnect() which are to be called whenever the application starts or ends a thread. Framing a thread with these calls guarantees that communication with the MQ Workflow execution server can be handled properly by the API. There are, however, no such calls in the MQ Workflow Java API.

Communication between the MQ Workflow APIs and the MQ Workflow servers requires one MQSeries connection per thread. For performance reasons, these connections are cached in the C-language layer of the MQ Workflow API, which is part of the JNI code. MQSeries connections are managed through handles which are owned by the thread that opened the connection. That is, each thread has to acquire its own connection handle; it cannot use the handles of other threads. When a thread ends, the connection has to be closed for the MQSeries resources to be freed. If an application fails to close the connections, MQSeries has built-in mechanisms to nevertheless reclaim the resources after some time. But this mechanism is not always fast enough to free the resources in time. There are even configurations where this is not possible at all.

When the MQ Workflow Java API is used in thread-pooling environments (for example, when it is running as an RMI-IIOP Agent or in a Servlet container), not freeing resources can cause the MQSeries Listener to run out of resources and you will receive error messages like "Maximum number of channels reached" or "Channel program terminated abnormally". The problem encountered here is that the thread pool is managed outside of MQ Workflow (for example, by a Servlet or EJB container) and usually provides no hooks for thread creation or thread destruction. The MQ Workflow APIs can detect if they are invoked from a new worker thread. When the API is invoked from a new worker thread, there is no connection handle and a new handle is created. But there is no mechanism to get informed when a worker thread ends.

Although it is possible to periodically check if the thread associated with a connection handle ended, it is too late to issue a Disconnect() since MQSeries connection handles are thread specific and the thread already ended. Also, since none of the thread-pool owners maintains an affinity between a client

and a worker thread, offering the Connect() and Disconnect() methods in the MQ Workflow Java API does not help to control communication from within threads.

Running out of channels can be avoided by using the MQSeries server interface (MQI) instead of the client interface (MQIC) in such configurations. This setup allows MQSeries to reclaim its resources without the application having to specifically call disconnect. Using MQI requires a local Queue Manager to be configured on the same machine which is a so-called *client concentrator* setup. The MQ Workflow Configuration automatically uses a client concentrator setup for these types of configurations. Details about the client concentrator setup can be found in the *IBM MQSeries Workflow: Installation Guide*.

JNDI locator policy

The JNDI locator policy normally uses the built in ORB of the Java 2 runtime environment. This ORB, as most other ORBs, manages a pool of worker threads and offers no mechanisms to hook the thread control methods. Thus, using the JNDI locator policy requires an MQSeries server installation and a client concentrator setup on the machine where the Java CORBA Agent is running. Details about the client concentrator setup can be found in the *IBM MQSeries Workflow: Installation Guide*.

OSA, IOR, and COS locator policies

Note: Using the OSA, IOR, and COS locator policies is deprecated.

If you use the OSA, IOR, or COS locator policies, you can specify the number of threads in the thread pool.

The Visibroker ORB uses a thread pool as the default threading policy when accessing the Java CORBA Agent via one of the CORBA locator policies, that is, via a Visibroker Smart Agent (OSA), an Interoperable Object Reference (IOR), or a COS Naming Service (COS). To overcome running out of channels you should set the maximum number of threads, **OAthreadMax**, and the minimum number of threads, **OAthreadMin**, to the same value so that no worker threads are dynamically created or ended depending on the workload. This applies to all operating system platforms.

Note that this kind of setup may constantly require a lot of system resources. It can also be challenging to determine the optimum number of threads to handle the workload. Thus, the recommendation is to have a client concentrator setup.

To specify the number of threads opened, edit the Java CORBA Agent startup script (batch file) for a specific configuration and add the following statements to the end of the line containing "com.ibm.workflow.agent.Main":

```
-OAthreadMin xxx -OAthreadMax xxx    where xxx denotes the number of threads
```

For example, on Windows NT for configuration TTT:

```
java -Xbootclasspath: ... -classpath ... com.ibm.workflow.agent.Main -yTTT
               should be changed to
java -Xbootclasspath: ... -classpath ... com.ibm.workflow.agent.Main -yTTT
                          -OAthreadMin xxx -OAthreadMax xxx [7]
```

For further details on the Visibroker ORB command line parameters see the documentation that comes with the Visibroker for Java.

RMI locator policy

Using the RMI locator policy is adequate for applications that do not handle a large number of concurrent users, like test and prototyping environments. The RMI specification does not define a specific thread-dispatching policy so that each RMI implementation shows different runtime characteristics.

For example, the standard implementation suffers from several shortcomings:

- There is no configurable upper limit on the number of server (agent) threads to handle client requests. Thus the server (agent) could be overwhelmed with requests and thereby exhaust its resources. Developing a solution to this problem is difficult at best because RMI does not provide a hook for implementing custom threading policies as it does for custom sockets.
- Threads are not reused so that resources are wasted.

Note: Using the RMI locator policy is deprecated.

Microsoft JVM/Internet Explorer V4/V5 and RMI

Although Microsoft does not officially support RMI, a file named rmi.zip' can be found on Microsoft's ftp site (ftp://ftp.microsoft.com/developr/MSDN/UnSup-ed/rmi.zip). Note that 'UnSup-ed' means 'unsupported'.

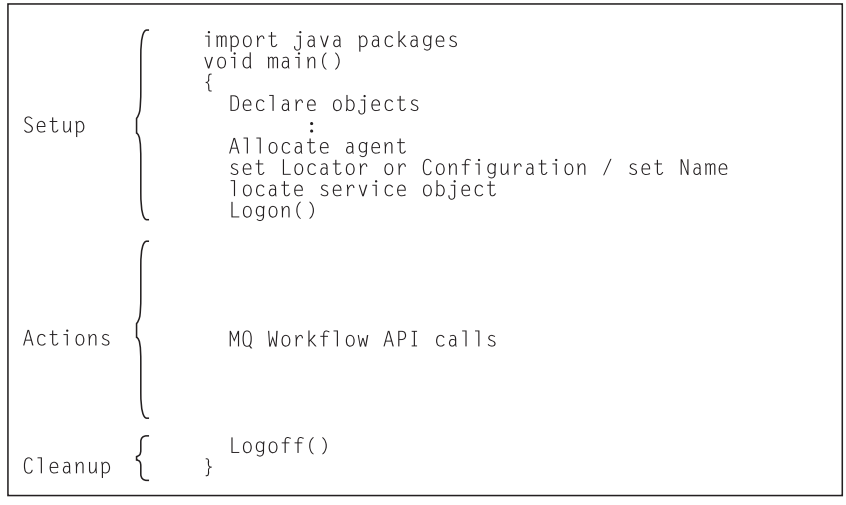
For Windows NT, this file must be extracted into the \Winnt\Java\Trustlib directory, for Windows 95 or Windows 98 into \Windows\Java\Trustlib. Add rmi.zip to the TrustedClasspath in the registry:

7. The line is split for printing purposes only.

```
REGEDIT4 [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Java VM]
         "TrustedClasspath"="c:\winnt\java\trustlib\rmi.zip;..."
```

Chapter 23. An MQ Workflow client application

An MQ Workflow Java client application typically contains the following parts, not necessarily divided that clearly.



To set up your program, you typically declare the program variables or objects you are going to use and you import the MQ Workflow Java API packages you need for your actions.

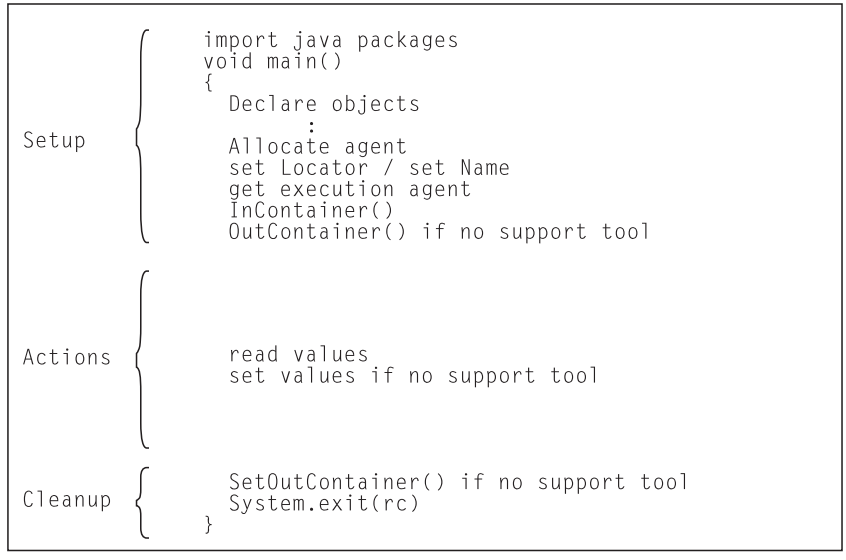
You then need to access a service object which represents the server you are going to ask services from. You do this by locating it via an appropriate agent. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your 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 MQ Workflow objects you are authorized for.

At the end of your program, you log off in order to close the session to the server.

Chapter 24. An MQ Workflow activity implementation

An MQ Workflow Java activity implementation or support tool 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 import the MQ 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.

An activity implementation can then retrieve the activity's input and output containers from the MQ Workflow program execution agent that started this program. A support tool can retrieve the activity's input container only.

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 MQ Workflow program execution agent. The return value of your program or the _RC value in the output container tells the program execution agent about the overall outcome of your program.

The output container is passed back to the MQ 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 MQ Workflow navigation.

Your activity implementation can as well behave like a client application (see "Chapter 14. An MQ Workflow client application" on page 159) and request services from an MQ 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.

Refer also to

```
<MQ Workflow installation directory>\smp\java\actimpl\ActImpl.java
```

for an example of an activity implementation in Java.

The Java High Performance Bridge

The Java High Performance Bridge serves to eliminate the overhead of creating a new Java Virtual Machine (JVM) for each invocation of an activity implementation written in Java. The Java High Performance Bridge instantiates a JVM in the address space of a program execution agent which can then be used for all invocations of Java-based activity implementations.

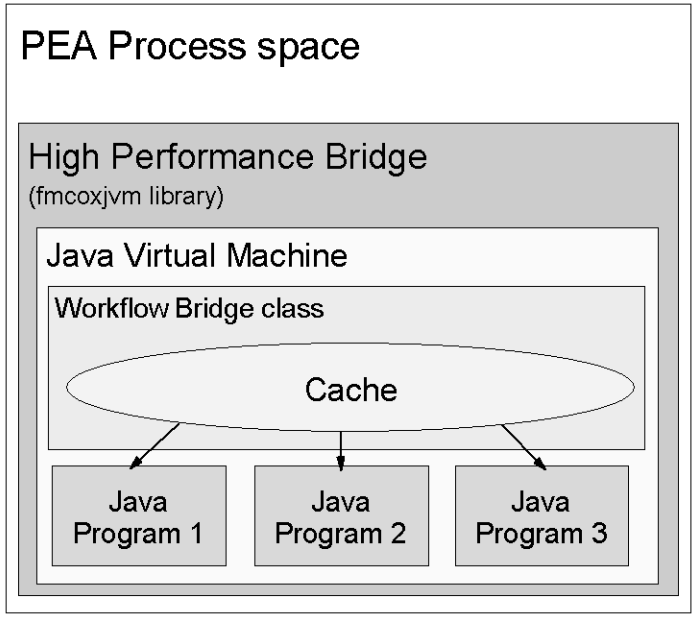


Figure 13. Overview

Furthermore, Java classes providing the activity implementations are loaded upon initial invocation and are cached thereafter. Consequently, subsequent invocations reuse the already loaded class definitions reducing loading time even further.

Similarly, static variables are cached along with the main methods of the classes. This means that invoked services can maintain state between invocations. This feature can, for example, be exploited by activity implementations in order to cache resources which are frequently used but expensive to obtain. For example, JDBC connections or EJB Home references can be created once and then used by subsequent program executions.

The biggest performance gain using the Java High Performance Bridge can be achieved in scenarios where multiple activity instances using the same activity implementations are started over time.

Setup on Windows platforms

To use the Java High Performance Bridge on the Windows platforms, the program settings of a Java-based implementation must be adapted.

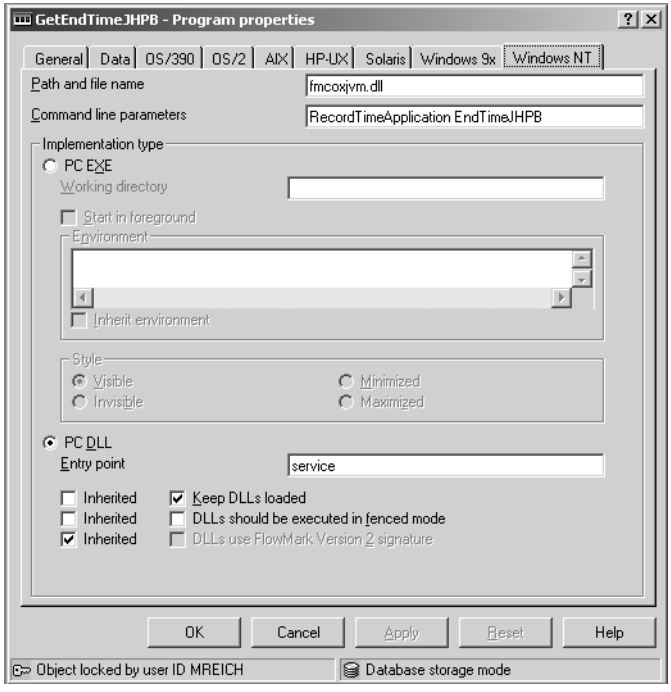


Figure 14. Configuration of the Java High Performance Bridge for Windows

1. The *Path and file name* has to be set to fmcoxjvm.dll.
2. The *Command line parameters* have to specify the class name and the arguments implementing the activity.

   Note: The used class files must be stored in a location which is part of the classpath that the PEA uses for the invocation of activity implementations.
3. *Implementation type* must be *PC DLL*. The *Entry Point* is service.
4. To gain the benefits of the Java High Performance Bridge, the checkbox *Keep DLLs loaded* must be checked. To be able to do so, you must first uncheck the corresponding *Inherited* checkbox.
5. Further it must be ensured that the Dll is not run in fenced mode. This is done by unchecking the *Inherited* checkbox and keeping the *DLLs should be executed in fenced mode* checkbox unchecked.

The following example shows an FDL that describes an activity implementation which uses the Java High Performance Bridge:

```
PROGRAM 'GetStartTimeJHPB' ( 'TimeContainer', 'TimeContainer' )
    WINNT DLL PATH_AND_FILENAME "fmcoxjvm.dll"
    ENTRY_POINT "service"
```

```
    PARAMETER "RecordTimeApplication StartTimeJHPB"
    KEEP_LOADED
    NO FENCED_MODE
END 'GetStartTimeJHPB'
```

Setup on UNIX

To use the Java High Performance Bridge on the AIX and Sun Solaris platforms, the program settings of a Java-based implementation must be adapted.

Note: HP-UX is not supported.

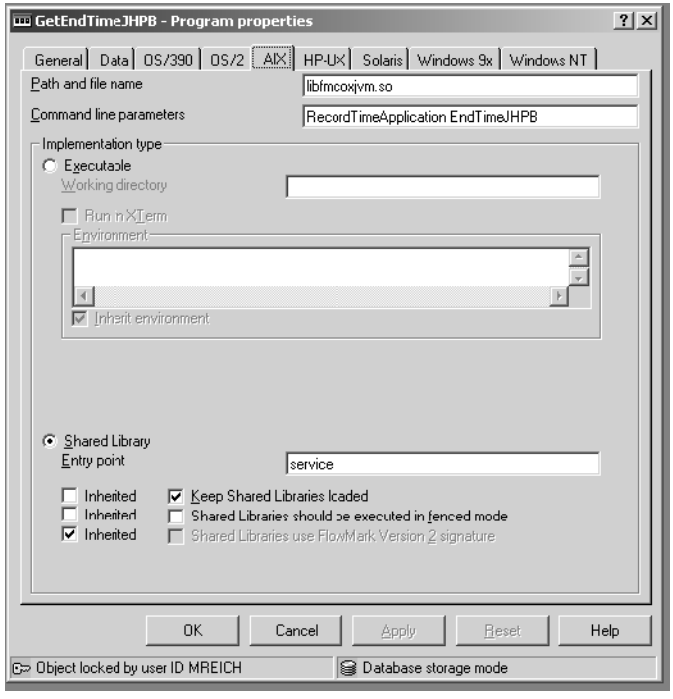


Figure 15. Configuration of the Java High Performance Bridge for Windows

1. The *Path and file name* has to be set to libfmcoxjvm.so.
2. The *Command line parameters* have to specify the class name and the arguments implementing the activity.

   Note: The used class files must be stored in a location which is part of the classpath that the PEA uses for the invocation of activity implementations.
3. *Implementation type* must be *Shared library*. The *Entry Point* is service.

4. To gain the benefits of the Java High Performance Bridge, the checkbox *Keep Shared libraries loaded* must be checked. To be able to do so, you must first uncheck the corresponding *Inherited* checkbox.
5. Further it must be ensured that the Dll is not run in fenced mode. This is done by unchecking the *Inherited* checkbox and keeping the *Shared libraries should be executed in fenced mode* checkbox unchecked.

The following example shows an FDL that describes an activity implementation which uses the Java High Performance Bridge:

```
 PROGRAM 'GetStartTimeJHPB' ( 'TimeContainer', 'TimeContainer' )
    AIX DLL PATH_AND_FILENAME "libfmcoxjvm.so"
    ENTRY_POINT "service"
    PARAMETER "RecordTimeApplication StartTimeJHPB"
    KEEP_LOADED
    NO FENCED_MODE
END 'GetStartTimeJHPB'
```

Programming considerations

Maintaining state between invocations

The state of a service implemented as an activity callable by the Java High Performance Bridge can be maintained by using static variables in the implementation classes. Because static variables are cached with the main methods, the values of these variables can be read by succeeding invocations.

Static variables can also be used for caching resources expensive to retrieve. They must be created only once and remain cached for subsequent executions of the same program.

Since the Java High Performance Bridge and thus the activity implementations are running in *unfenced* mode, it is important to consider the following aspects:

Program termination

Because running *unfenced*, a system exit call in the activity implementation causes the program execution agent to exit. Instead of calling System.exit(rc), you need to pass back the return code of your program as part of the output container (_RC).

Threading considerations

Because the activity implementation is kept loaded, only one instance of the appropriate class is instantiated for multiple invocations. All static variables are kept loaded until the class is reloaded, for example, when the program execution agent is restarted. To avoid inconsistencies due to concurrent execution, the activity implementation needs to care for its thread-safeness. This means that the activity implementations are to be coded reentrantly and need to care for necessary synchronizations.

Output to standard out

Programs generating output to standard out are not suitable for execution via the Java High Performance Bridge. Each output generated is sent to the program execution agent, which displays it in its own window. This procedure is significantly slower than passing data to the window of a JVM.

Chapter 25. Compiling

All programs developed for use with the MQ Workflow Java API Classes must import the packages provided by MQ Workflow. These files have been installed on your system if you selected to install the MQ Workflow Development Kit. They are installed by default in the \bin\java3300 subdirectory of the installation directory.

JDK 1.2.y (y=2 or higher) or JDK 1.3 can be used to compile and run your applications accessing the MQ Workflow Java API. A sample compile statement is:

```
javac <java file>.java
```

The CLASSPATH must include either fmcojagt.jar or fmcojapi.jar. Make sure that only one of the jar files is included in the CLASSPATH since fmcojagt.jar is a strict superset of fmcojapi.jar. fmcojapi.jar contains all client side stubs for the JNDI, RMI, OSA, COS, and IOR locator policies. fmcojagt.jar additionally contains the corresponding skeletons and the LOC locator policy.

JNDI locator policy

When using the JNDI locator policy, the naming context factory class and the URL of the JNDI Naming Service have to be specified. The *IBM MQSeries Workflow: Installation Guide* describes how these parameters are set on the command line. Alternatively, the parameters can also be set in the application. For example:

```
public static void main ( String[] args )
{
...
Agent       agent= new Agent();
Properties  prop = System.getProperties();

prop.put( javax.naming.Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.cosnaming.CNCtx.Factory") ;
prop.put( javax.naming.Context.PROVIDER_URL,
          "iiop://<Hostname of JNDI Name Server>:<JNDI Name Server Ports"" );

agent.setContext( args, prop );
agent.setLocator( Agent.JNDI_LOCATOR );
agent.setName( "<AgentName>");
...
}
```

The previous example shows a COS Naming implementation of the JNDI specification. Other JNDI service providers, for example, an LDAP server, can require the specification of additional or different properties to perform a lookup for objects.

Chapter 26. How to use the MQ Workflow Java API from within IBM VisualAge for Java

1. If you are using the Inprise VisiBroker for Java ORB, you need to create it first:
   - In the Workbench, add a project for the ORB, for example, “VisiBroker ORB”.
   - Import the files vbjorb.jar and vbjcosnm.jar into the newly created project. You can ignore any problems concerning the package ’com.visigenic.vbroker.CORBA’.
2. Create a project for the MQ Workflow JAVA API, for example, “MQWF Java API”, and import the file fmcojagt.jar into that project. If you did not import the VisiBroker ORB first because you are using local, JNDI, or RMI bindings, you will encounter several problems in the com.ibm.workflow.corba package which you can ignore.

   Note that importing the MQ Workflow Java API into VisualAge for Java is necessary. It is not sufficient to enter fmcojagt.jar in the “Window” - “Options...” - “Resources” - “Workspace class path:” field.
3. Create your actual project which will use the MQ Workflow Java APIs, for example, ’MyProject’. Write your program as usual, for example, a class HelloWorld which contains the public ’main’ method.
4. Before running your program, select ’Properties’ on your HelloWorld class. Open the “Classpath” page and select “Project path”. Select “Edit”, then select the “VisiBroker ORB”, if necessary, and “MQWF Java API” packages to be included in your project classpath. Now you are ready to run your application.

Running the MQ Workflow Java CORBA Agent inside the WebSphere Test Environment

If you want to start the MQ Workflow Java CORBA Agent inside the WebSphere Test Environment of Visual Age for Java, you have to perform the following steps:

1. Install and configure the Java CORBA Agent as usual.
2. Create a project for the Java CORBA Agent as described above.
3. In the ″Projects″ tab, select your Java CORBA Agent project. Select the package ″com.ibm.workflow.agent″ and then the class ″Main″. Right-click on the ″Main″ class and select ″Properties″ from the context menu.
4. In the ″Program″ tab go to the ″Command line arguments″ field and enter ″-y <name of your configuration>. For example, ″-y FMC1″.

5. In the ″Class Path″ tab, click on the ″Edit″ button of the Project path and select ″IBM WebSphere Test Environment″. If required by your application, select additional projects. Click ″OK″ to continue.
6. Click on the ″Edit″ button of the Extra directories path. Click on the ″Add Jar/Zip″ button and select the file ″fmcojagt.jar″ located in the directory ″<MQ Workflow installation directory>\bin\java330\″. Click on ″Open″ and then ″OK″ to continue.
7. Make sure that ″Save in repository (as default)″ is checked and click on ″OK″ to finish.
8. To start the Java CORBA Agent you have to start the WebSphere Test Environment first. If you configured the Java CORBA Agent for the JNDI locator policy, you also have to start the Persistent Name Server. Start the Java CORBA Agent by right-clicking the class ″Main″ in the package ″com.ibm.workflow.agent″ and then clicking ″Run″ followed by ″Run main″ in the context menu.
9. You can stop the Java CORBA Agent by selecting it and clicking ″Terminate″ in the WebSphere Test Environment Console.

Chapter 27. Troubleshooting

As a general rule of thumb: Whenever you experience problems with the MQ Workflow Java API:

- Turn off the JIT (Just in Time) compiler.
- Make sure you are using the most recent service level of the Java Development Kit.
- Consult your JDK/JRE Application Server documentation for detailed guidelines.

Following these recommendations is especially important when you experience exceptions providing a stack trace. Stack traces can be inaccurate when used with the JIT Compiler; the JIT Compiler may have optimized your code in a way that method calls no longer appear in the stack trace. Therefore, when you contact the MQ Workflow support team, provide only stack traces with the JIT Compiler turned off.

Chapter 28. Object management

Workflow process models, their instances, and resulting work items are all objects persistently stored in an MQ 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 are supporting objects, which do not reflect persistent ones. An example is the Agent object, which allows to obtain a reference to an ExecutionService object.

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 life time of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQ Workflow API.

As all resource memory is finally owned by the application process itself, you can access all objects from different threads within that process. MQ Workflow does not hinder you from using threads; it is coded reentrantly. On the other hand, the MQ Workflow Java API explicitly supports threads for action methods only. That is, all action methods are synchronized. Accessor methods are not synchronized because they normally read values only. When, however, the accessed value is not yet available in the API cache, the object is automatically refreshed from the server so that you need to synchronize parallel accesses on that object. Refer also to the discussions on "Accessor/mutator API calls" on page 106 and "Chapter 12. Stateless server support" on page 87.

Garbage Collection when using Java API classes

Garbage collection is normally running 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 nonreferenced 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 pendants of not referenced 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, in fact, can do that, however, they are using proprietary CORBA extensions to achieve this). Agent-side pendants of client objects registered with an ORB by using a connect method have to be disconnected explicitly. When using MQ Workflow Java API classes, the user is provided with a build-in garbage collection mechanism, the MQ Workflow Java API Classes Reaper, that does housekeeping when the transmission of data is done by a CORBA Object Request Broker (ORB). Before starting the MQ Workflow Java API CORBA 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. Default value is 300000 msec.
- 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.

Part 5. The XML message interface

The following chapters provide a description of the MQ Workflow XML message-based interface. It explains the format of a message and how XML can be used:

- Sending requests to MQ Workflow

  An action can be started on an MQ Workflow server by sending a message to the MQ Workflow XML input queue.

  This allows any application that supports the MQ Workflow XML message format to request an action from MQ Workflow.

- Invoking an activity implementation

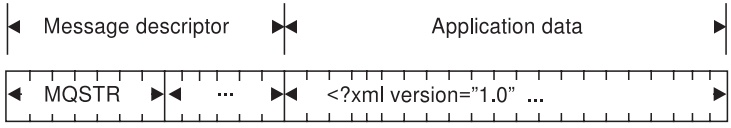
  An activity implementation is invoked by MQ Workflow by sending an appropriate message to a user-defined MQSeries queue.

  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 in-house application, or a standard program, such as MQSeries Integrator V2.

Chapter 29. The MQ Workflow XML message

MQ 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 MQ Workflow XML message itself



For more information about the MQMD, the application data, and how to send and receive messages in an MQSeries network, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries messages".

Relevant MQSeries Message Descriptor (MQMD) fields

The following fields of the MQSeries message descriptor are used by MQ Workflow:

- UserIdentifier

  The user who sent the message. For request messages sent to MQ Workflow this information is used as the MQ Workflow user on whose behalf the request is performed. Also, authorization checks are performed using this user ID. For invoke messages sent by MQ Workflow this field contains the user on whose behalf the activity implementation is to be started.

- Format

  A fixed string indicating that this message contains an MQ Workflow XML message. Its value is defined by the MQSeries constant MQFMT_STRING (MQSTR). For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

- 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 MQ Workflow, the MQ Workflow response has the same persistence as the

request. XML requests sent by MQ Workflow are persistent and responses sent by invoked activity implementations should also be persistent.

- Expiry

  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 MQ Workflow, the expiry of the MQ Workflow response is set to the expiration value in the request minus the time spent on execution. XML requests sent by MQ 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 MQ Workflow, the MQ Workflow response contains the same correlation ID as the request. XML requests sent by MQ Workflow contain a correlation ID and responses sent by an application should return that correlation ID.

- BackoutCount

  A count of the number of times the message has been returned to the input queue because the transaction failed and was rolled back. In other words, the backout count denotes the number of times processing of the message was not successful.

For detailed information, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries Messages".

The application data

MQ 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 MQ Workflow XML message contains the following information:

- An MQ Workflow XML message header, that is, information that is common for all messages, for example, the user context
- An MQ Workflow message name which specifies the request or response contained in the message, for example, a "ProcessTemplateExecute" request
- The parameters that are needed to execute the request or to analyze the response, for example, an input container

When processing an MQ Workflow XML message, MQ Workflow checks if the message has the correct format - see "Chapter 32. Error Handling" on page 227.

The MQ Workflow XML message header

The MQ 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 reply queue specified in the MQMD of the request message. If *ResponseRequired* is not set, then the default value assumed by MQ Workflow for requests is ″Yes″, while the default value for responses is ″No″.

- 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 MQ 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 “Chapter 9. Handling containers” on page 45. The following example shows a container of type *CreditData* containing three container elements of a basic type, *Amount* of basic type LONG and *Currency* and *Risk* of basic type STRING, and a nested container element *Customer* of type CustomerData. *CustomerData* contains a container element *Name* of basic type STRING and an array *Account* with two elements of basic type LONG.

```
<CreditData>
   <Customer>
      <Name>User1</Name>
      <Account>4711</Account>
      <Account>1100</Account>
   </Customer>
```

```
   <Amount>100000</Amount>
   <Currency>CurrencyX</Currency>
   <Risk>high</Risk>
</CreditData>
```

The following rules apply to containers in the message-based interface:

- A container, actually the user-defined part of the container without the pre-defined data members, is identified by its type, that is, the name of the associated data structure. In other words, the name of the XML element describing the container is the name of the data structure specifying the type of the container; CreditData in the example above.
- Container elements are specified by their name; their type is not part of the XML message.
- Container elements can be of a basic type or denote another data structure as their type. Basic container elements are mapped to PCDATA elements, while container elements of a non-basic type are decomposed into XML subelements according to their structure.
- The structure of XML elements representing a container or container element of a non-basic type reflects the structure of the associated data structure. Therefore, data member names are not prefixed; there is no 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 MQ Workflow Buildtime Verification encounters one of these situations, it issues a warning.
- XML tags must not contain blanks. It follows that data structure, and data member names must not contain blanks. If such a name containing blanks is to be referred to in an XML message, then the name without blanks is to be used. For example, the "Default Data Structure" is to be referred to as <DefaultDataStructure>.

  Note: Ambiguities arise when a model contains data structures with names differing only by blanks. The data modeler has to prevent that situation.
- Data structure and data member names must not contain reserved XML characters as there are '<', '>', and '/'. If such a name is used in an XML message, the generated XML is not well formed and cannot be parsed.
- The specification of container elements (data structure members) in the XML message is optional. If a data member is not specified in the XML message, MQ Workflow sets its value to *Null* (not set).

Data type encoding rules:

- The values of container elements of basic types STRING, LONG, and FLOAT can be coded directly.

- For boolean parameters, values ″false″ and ″true″ (case insensitive) should be used. Values 0 and 1 are also supported.
- Binary data has to be encoded into its printable version. MQ Workflow uses base64 encoding to represent binary data in XML messages. Note that this encoding preserves length information. Refer to http://www.cis.ohio-state.edu/htbin/rfc/rfc2045.html for base64 encoding rules.
- Leading blanks, trailing blanks, new line characters, and tab characters are removed from values of LONG, FLOAT, and boolean types. Such characters are allowed to support you in formatting your XML message. They are, however, not removed from values of types STRING or BINARY so that the value specified remains unchanged.

Representation of arrays:

- Arrays are specified as a sequence of elements.
- Since arrays are to be specified as a sequence of elements, arrays with one element only are not supported because they cannot be distinguished from non-array elements.
- The <null/> element can be used to specify that an array element is *Null* (not set). For example, anticipate a container of type *Error*. Error contains an ID and three reason codes (an array of reason codes). Assumed that the second reason code is not set, this can be specified as:

```
<Error>
    <ID>111</ID>
    <ReasonCode>12</ReasonCode>
    <ReasonCode><null/></ReasonCode>
    <ReasonCode>5050</ReasonCode>
</Error>
```

Pre-defined data members:

- Pre-defined data members are specified in the same way as user-defined data members. They are to be defined right after the XML container element, for example, after <ProcInstInputData>. Again, they have to be decomposed according to their data structure. For example:

```
<_ACTIVITY_INFO>
   <Priority>1</Priority>
</_ACTIVITY_INFO>
```

Execute process instance example

The following example shows an XML message that requests the execution of a process instance:

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateExecute>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
```

```
      <ProcInstName>Credit_Request#658321</ProcInstName>
      <KeepName>true</KeepName>
      <ProcInstInputData>
          <_ACTIVITY_INFO>
            <Priority>1</Priority>
          </_ACTIVITY_INFO>
          <CreditData>
            <Customer>
              <Name>User1</Name>
              <Account>4711</Account>
              <Account>1100</Account>
            </Customer>
            <Amount>100000</Amount>
            <Currency>CurrencyX</Currency>
            <Risk>high<Risk>
          </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 MQ Workflow are converted from their format as specified in the encoding keyword to the MQ Workflow code page as necessary. XML messages sent by MQ Workflow (responses and activity implementation invocation requests) are always encoded in UTF-8.

Refer to the readme.1st, chapter “XML code page support”, for a list of code pages supported by the MQ Workflow XML parser.

Chapter 30. Sending requests to MQ Workflow

The MQ Workflow message-based interface can be used to request services from MQ Workflow. This is depicted in the following figure:

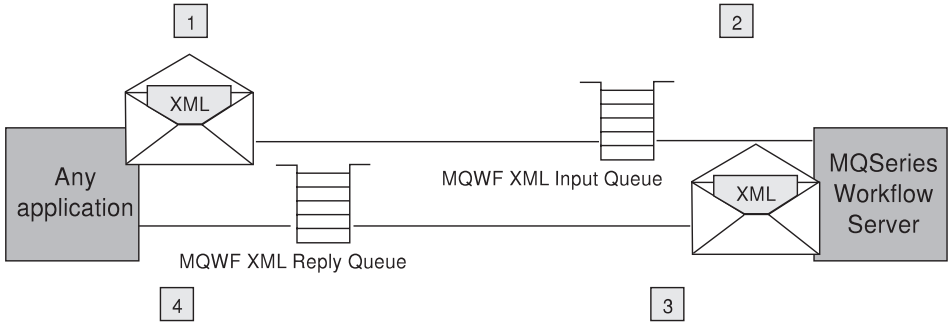


Figure 16. Sending requests to MQ Workflow

1. An application creates an MQ Workflow XML message and puts it into the MQ Workflow XML input queue.
2. MQ Workflow reads the XML message out of the XML input queue and processes the request.
3. MQ Workflow creates a response MQ Workflow XML message and puts it into the reply queue. Note that the reply queue information is part of the MQSeries Message Descriptor (MQMD) of the incoming XML message.

   For more information about the MQMD, the application data, and how to send and receive messages in an MQSeries network, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries messages".
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 571.
- "ExecuteProcessInstance()" on page 585.

XML input queue

The XML input queue <prefix>.<SystemGroupName>.<SystemName>.EXE.XML respectively <prefix>.<SystemGroupName>.EXE.XML is an MQSeries queue to which MQ Workflow is listening.

Only XML messages are accepted as input to this queue. The XML message has to conform to the MQ Workflow XML message format. If it does not conform, then a GeneralError XML message is put into the reply queue.

For more information refer to chapters "Relevant MQSeries Message Descriptor (MQMD) fields" on page 209 and "The application data" on page 210. For more information on error handling refer to "Chapter 32. Error Handling" on page 227.

Authentication and authorization

For authentication MQ Workflow's message-based interface relies on MQSeries. MQ Workflow does not perform any additional authentication. For setting up MQSeries security, refer to *IBM MQSeries System Administration*, chapter "Protecting MQSeries Objects".

The value of the UserIdentifier field in the MQMD of the incoming message is used as the MQ Workflow user on whose behalf the request is to be performed. Authorization checks for that user are performed as usual.

Note: MQSeries UserIdentifier constraints differ from the ones defined for the MQ Workflow system. Since authorization is checked by MQ Workflow, the UserIdentifier in the MQMD of an XML message must be a valid MQ Workflow user. This has to be ensured by the application programmer and MQ Workflow administrator.

Create and start a process instance example

The following example shows an XML message that requests the creation and start of a process instance. Assumed that the process input data is described by data structure *Application*:

```
STRUCTURE 'Application'
  'InsuredID'      : LONG;
  'Type'           : LONG;
  'SpecialRisk'    : Long;
  'Accident'       : Long;
  'Ammount'        : FLOAT;
  'StartDate'      : STRING;
  'Term'           : Long;
  'Payment'        : Long;
  'Doctor'         : STRING;
```

```
  'Weight'         : Long;
  'Height'         : STRING;
  'Smoker'         : Long;
  'Illness'        : STRING;
  'Hospitalization': STRING;
  'Risks'          : STRING;
END 'Application'
```

then the XML message can look like following:

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateCreateAndStartInstance>
      <ProcTemplName>Medical_Opinion</ProcTemplName>
      <ProcInstName>Medical_Opinion#448</ProcInstName>
      <KeepName>false</KeepName>
      <ProcInstInputData>
          <Application>
            <InsuredID>A</InsuredID>
            <Type>4711</Type>
            <StartDate>12.01.2000</StartDate>
            <Doctor>DoctorX</Doctor>
            <Weight>200</Weight>
            <Smoker>1<Smoker>
          </Application>
    </ProcInstInputData>
  </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

Chapter 31. Invoking an activity implementation

Activity implementations are usually started by MQ 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 that MQ Workflow sends 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 invoke the program that is modeled as the implementation of the activity. For doing so, all the necessary information is passed to the MQSeries application by means of an XML message. The MQSeries application must return with an appropriate XML response, if requested by MQ 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 or a program such as MQSeries Integrator, provided it can deal with the MQ Workflow XML message format.

A UPES and a program activity to be performed by that UPES are modeled in the MQ Workflow Buildtime. The program activity is modeled using the activity property sheet.

Two invocation modes for the activity implementation can also be modeled:

- Synchronous invocation (the standard case), where MQ 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 right after the invocation message has been sent. No result data is expected by MQ Workflow and process navigation continues.

The following figure depicts the synchronous invocation of an activity implementation:

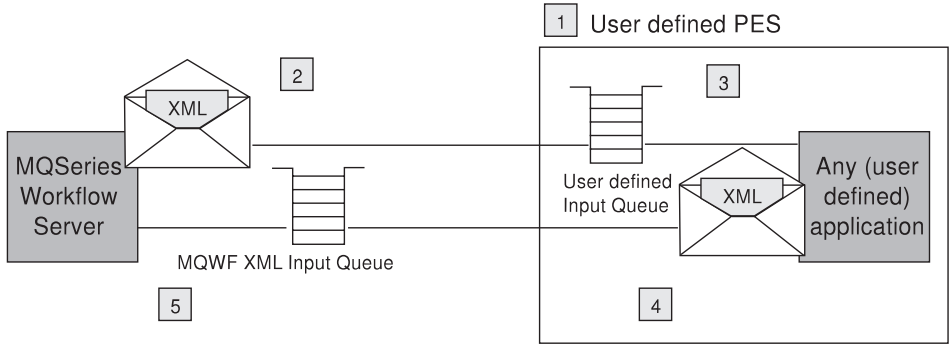


Figure 17. Starting an activity implementation

1. A UPES must have been defined using MQ Workflow Buildtime.
2. When an activity implementation is to be started, MQ Workflow 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 MQ Workflow.
4. When the activity implementation has finished, the application creates a response MQ Workflow XML message, if required, and puts it into the reply queue. Note that the reply queue information is part of the MQMD of the incoming XML invocation message.
5. MQ Workflow 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 MQ Workflow system by modeling it in MQ Workflow Buildtime. Essential attributes are the name, the version, and the queue it represents.

The UPES version denotes the MQ Workflow API version that is supported by the UPES. MQ Workflow only sends messages to the UPES that are supported. When modeled in the MQ Workflow Buildtime, it becomes a tag in

the FDL when the UPES is exported. If no UPES version is specified when imported into the MQ Workflow Runtime, the FDL version becomes also the UPES version.

Note: You need to upgrade the UPES version in order to receive new messages that become available with new MQ Workflow versions or releases.

For more information on UPES attributes refer to the online help of MQ Workflow Buildtime.

The application that is listening to the UPES queue is not managed by MQ Workflow. A system administrator is responsible for administering the application. From an MQ Workflow point of view, the invocation of an activity implementation was successful when the invocation message is successfully put into the UPES queue.

For more information on how to create and administer MQSeries queues, refer to the *IBM MQSeries System Administration*.

Messages sent to a UPES

Following messages are sent to a UPES by MQ Workflow, depending on the version of the UPES:

Message	UPES Version	Remarks
ActivityImplInvoke	Version 3.2.2 and higher	Message to invoke an activity implementation. A response is required in case of a synchronous invocation.
ActivityExpired	Version 3.3.0 and higher	Message to inform about the expiration of an activity. No response is expected.
TerminateProgram	Version 3.3.0 and higher	Message to inform about the termination of an activity. No response is expected.

In general, when a message has been placed into a UPES input queue, it becomes the responsibility of the UPES to get that message from the queue, process the message, and to put a response message into the reply queue if requested. The UPES can use MQSeries transactional capabilities to bring get and put messages into a transactional context.

ActivityImplInvoke message

When the UPES receives an *ActivityImplInvoke* message, the UPES is asked to invoke the program specified in the *ImplementationData*[8] XML element.

It is up to the UPES how, when, and where to invoke that program. Note that the UPES has to remember the ActImplCorrelID of an incoming ActivityImplInvoke message so that it can be returned to MQ Workflow in the ActivityImplInvokeResponse and to correlate ActivityExpired or TerminateProgram messages to an invocation.

Depending on the nature of the activity instance, the activity implementation, that is, the specified program, may only need to be triggered and then runs asynchronously to the MQ 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 MQ 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 information 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 MQ Workflow 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 (XML element *ActImplCorrelID*), which is passed in the invocation request by MQ Workflow, and has to be passed back in the response.

  The *ResponseRequired* element in the MQ Workflow ActivityImplInvoke message is set to 'Yes' to specify that the invocation is synchronous and that MQ Workflow expects a 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.

  The *ResponseRequired* element in the MQ Workflow ActivityImplInvoke message is set to 'No' to specify that the invocation is asynchronous and that MQ Workflow does not expect a response.

8. It is recommended to use the information from the ImplementationData XML element and to not misuse information from other elements.

The ActivityImplInvoke message provides the input container data as well as initial values set in the output container of the program to be invoked. The input container data is part of the *ProgramInputData* XML element; the initial values of the output container are part of the *ProgramOutputDataDefaults* XML element.

The UPES is responsible for processing of the container data, that is, to:

1. Pass the input container values to the activity implementation.
2. Copy the initial values of the output container into the output container (ProgramOutputData XML element) that is passed back to MQ Workflow. MQ Workflow only takes the values passed back in the ActivityImplInvokeResponse as output data; this allows values to be unset (to be set to null).
3. Copy the output values of the activity implementation into the output container (ProgramOutputData XML element) that is passed back to MQ Workflow.

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 ActivityImplInvokeResponse is expected by the MQ Workflow execution server. This message has to provide:

- The ActImplCorrelID of the associated ActivityImplInvoke message so that the MQ Workflow execution server can correlate the response with the Invoke request.
- For a successful execution, the return code and output container.
- For an unsuccessful execution, an exception passing the error code. The error code must be understood by MQ Workflow. See the file *fmcmretc.h* for a list of valid codes.

ActivityExpired message

This message informs the UPES about the expiration of an activity. It can only be sent after an ActivityImplInvoke message has been placed into the UPES input queue. If an ActivityImplInvokeResponse is expected, it informs the UPES that the response is not expected anymore. If the UPES sends a response, that response is ignored; the activity already expired. As a consequence, the UPES may interrupt program execution.

Correlation between the ActivityExpired message and the ActivityImplInvoke message can be done by means of the ActImplCorrelID.

TerminateProgram message

This message informs the UPES about the termination of an activity or work item. It can only be sent after an ActivityImplInvoke message has been placed into the UPES input queue. If an ActivityImplInvokeResponse is expected, it informs the UPES that the response is not expected anymore. If the UPES sends a response, that response is ignored; the activity is already terminated. As a consequence, the UPES may terminate program execution and perform compensation actions.

Correlation between the TerminateProgram message and the ActivityImplInvoke message can be done by means of the ActImplCorrelID.

Authorization

For invocation messages sent by MQ Workflow, the UserIdentifier field in the MQMD is set to the user ID of the user on whose behalf the activity instance has been started. Additionally, the <Starter> element in the invocation message itself is set to that user ID. The UPES applications can use this information to implement their own authorization schemes.

Note: The CorrelID in the MQMD is also set to the user ID. This information can be used by a UPES to listen to XML messages for specific users only.

Synchronous invocation example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>Yes</ResponseRequired>
  </WfMessageHeader>
  <ActivityImplInvoke>
    <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
    <Starter>user1</Starter>
    <ProgramID>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProgramName>PerformOrder</ProgramName>
    </ProgramID>
    <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>
          <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>
          <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
          <_PROCESS>CreditRequest#123</_PROCESS>
          <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
          <CreditData>
            <Customer>
              <Name>User1</Name>
            </Customer>
            <Amount>1000</Amount>
            <Currency>CurrencyX</Currency>
          </CreditData>
      </ProgramInputData>
      <ProgramOutputDataDefaults>
          <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
          <_PROCESS>CreditRequest#123</_PROCESS>
          <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
          <CreditData>
            <Risk>high</Risk>
          </CreditData>
      </ProgramOutputDataDefaults>
    </ActivityImplInvoke>
  <WfMessage>
```

UPES response example

```
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <ActivityImplInvokeResponse>
    <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
    <ProgramRC>0</ProgramRC>
    <ProgramOutputData>
       <CreditData>
          <Customer>
            <Name>User1</Name>
          </Customer>
```

```
            <Amount>1000</Amount>
            <Currency>CurrencyX</Currency>
            <Risk>low</Risk>
          </CreditData>
        </ProgramOutputData>
      </ActivityImplInvokeResponse>
    </WfMessage>
```

Chapter 32. Error Handling

This chapter describes how MQ Workflow handles errors which can occur during the processing of an incoming XML message.

MQ Workflow XML message life cycle

Each message that is received by MQ Workflow goes through a certain chain of processing steps. During each of these processing steps, errors can occur.

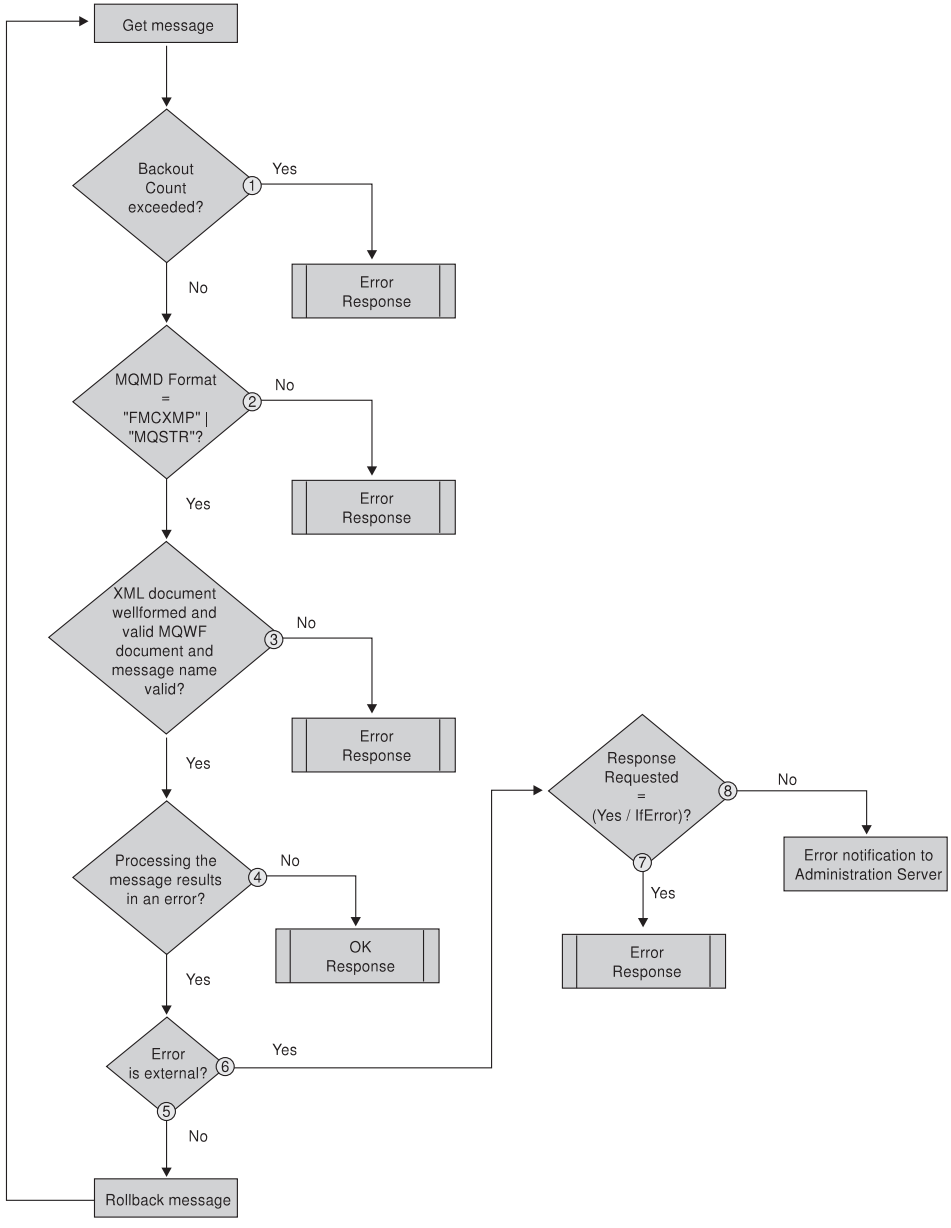
In general, the life cycle of an XML message within MQ Workflow can be described as follows:

1. Get the XML message from the MQ Workflow XML input queue
2. Parse the message
3. Determine the message name, for example, ProcessTemplateExecute
4. Check the validity of the application data parameters
   - Whether mandatory parameters are provided
   - Whether parameter values are syntactically correct
   - Whether semantical interdependencies are fulfilled
   - Whether container values fit to the container schema
   - Whether container values are syntactically correct
5. Process the message
6. Put the XML response message into the reply queue
7. Commit the transaction

When an error occurs, a response describing the error is created and put into the reply queue identified by the Reply2q and ReplyToQMgr fields in the MQMD fields of the input message. The reply queue address is abbreviated to REPLY2Q in the rest of this chapter.

General error processing

The following flowchart describes in more detail the life cycle of an MQ Workflow XML message and the errors which can occur and how they are handled.



1. When a message has been received from the XML input queue, then its backout count, that is, the number of times it has already been tried to process, is checked. If the backout count is exceeded, a GeneralError response message is returned. See “Sending a response” on page 230 for more information.

Note: The backout count is increased when the transaction is rolled back - see 5.

2. The MQMD Format field is checked for a correct value, namely, whether it is set to the MQSeries constant FMTMQ_STRING (MQSTR). If not, a GeneralError response is returned.

   Note: For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

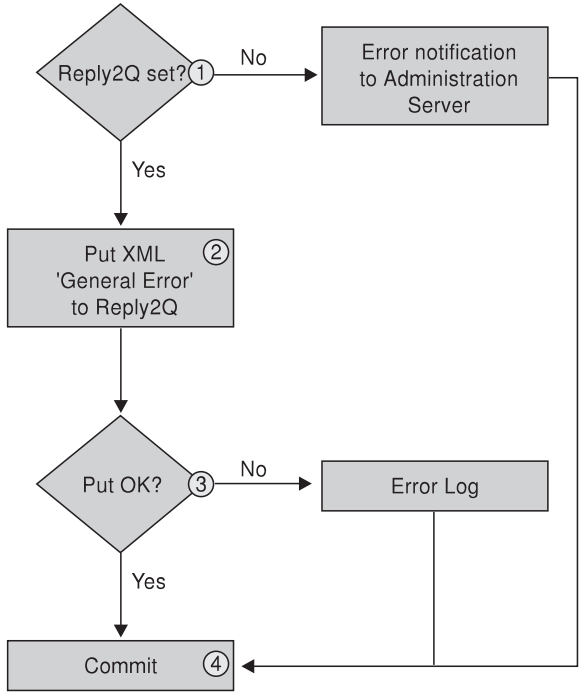
3. The XML message is checked for being well formed. Additionally, the message name must specify a function supported in the XML interface. If the XML message is not well formed or if the message name is not supported, a GeneralError response is returned.
4. The XML message is processed. If processing has been successful, the transaction is committed and a response denoting the successful processing is returned. If an error occurs, continuation depends on the kind of the error.
5. If the error is caused by internal reasons, for example, by a database lock, then MQ Workflow tries to newly process the message because such an error is usually only temporarily encountered.

   The message is rolled back so that it can be processed again. Backing out the message increases the backout count.

   Note: Depending on the severity of the error, rolling back a message can cause the MQ Workflow server to shut down.

6. If the error is caused by external reasons, for example, because a parameter is syntactically incorrect, then the response, if any, for the function requested describes the error.
7. A response describing an error is returned if the *ResponseRequired* element of the incoming message is set to 'Yes' or to 'IfError'.
8. Otherwise, an error notification which describes the error encountered is sent to the MQ Workflow administration server.

Sending a response



When a response is to be returned to the sender of an XML message

1. It is checked whether the REPLY2Q is set. If not, an error notification is sent to the MQ Workflow administration server.

   The administration server logs the error into a database. It can then be queried using the administration client. For more information refer to the *IBM MQSeries Workflow Administration Guide*, chapter "The error log".
2. If the REPLY2Q is set, the response is put into the specified reply queue.
3. If the 'Put' failed, an error log entry is written.
4. In either case, whether the 'Put' was successful or not, the transaction and thus the message read is committed so that the next message can be processed.

Detailed error processing

This chapter discusses some errors in more detail.

Wrong message format in the MQMD

The first time an error can occur is when the XML incoming message specifies an invalid Format field in the MQMD. Valid formats are MQFMT_STRING (MQSTR). For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

If the *Format* field is wrong, a GeneralError XML response message is sent to the REPLY2Q. Refer to "The GeneralError message" on page 234 for the specification of a GeneralError message.

When putting the response into the reply queue fails, an error notification is sent to the MQ Workflow administration server. The original XML message is committed and thus removed from the input queue.

The error returned is:

```
:msgID.       FMC_ERROR_XML_DOCUMENT_FORMAT
:msgNum.      1107
:severity.    Error
:msgText.     "The MQMD format field value '%1$s' of the XML document is
              incorrect.\\n"
$1: The value of the MQMD format field
```

The following XML message is returned:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
            FMC01107E The MQMD format field value 'ABC ' of the XML document
            is incorrect.\n
        </MessageText>
        <Origin>p:\v322\src\fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

Wrong message name or XML document not well formed

The next time to detect an error is during parsing of the XML message. The XML message is checked for being well formed[9] and the message name is investigated.

When the XML message is not well formed or when the message name is unknown, a GeneralError response is sent to the REPLY2Q. When putting the response into the reply queue fails, an error notification is sent to the MQ Workflow administration server, that is, put into the ADMINPUTQ. The original XML message is committed and thus removed from the input queue.

Note that, at this stage, any setting of the *ResponseRequired* tag is ignored because its determination is not always possible.

Possible errors returned are:

```
:msgID.       FMC_ERROR_XML_DOCUMENT_INVALID
:msgNum.      1100
:severity.    Error
"Incorrect XML document. The message that is returned by the XML parser is %1$s\\n"
$1: The error message that occurred during parsing of the XML message

:msgID.       FMC_ERROR_NO_MQSWF_DOCUMENT
:msgNum.      1101
:severity.    Error
"The XML document is not a valid MQSeries Workflow XML document.\\n"

:msgID.       FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED
:msgNum.      1102
:severity.    Error
"MQSeries Workflow message '%1$s' is not XML enabled.\\n"
$1: The XML MQSeries Workflow message name, for example, ProcessTemplateDelete
```

Message processing errors

When processing the incoming XML message, the validity of the application data is checked. It is checked whether:

- The *ResponseRequired* tag in the XML message header is set correctly
- The 'UserContext' obeys its rules, that is, obeys its length constraints (<= 254 bytes)
- All parameters are correct:
  - Whether mandatory parameters are provided
  - Whether parameter values are syntactically correct
  - Whether semantical interdependencies are fulfilled

9. A well formed XML document is defined by the XML standard and guarantees a certain level of syntactical correctness. A valid MQ Workflow document additionally requires that the root element name is 'WfMessage' with an optional nested element 'WfMessageHeader' followed by the message name.

- Whether container values fit to the container schema
- Whether container values are syntactically correct

If an error occurs, the response message for the requested function is used to describe the error, for example, a ProcessTemplateCreateAndStartInstanceResponse. A GeneralError response is not sent.

If an error occurs and *ResponseRequired* is set to:

Yes A response is sent in any case.

IfError Only error responses are sent.

No No response is sent.

Note: Errors of incoming XML responses, for example, an ActivityImplInvokeResponse, are treated the same way as incoming requests. The only difference is that, in case of an error, a GeneralError is sent instead of a specific error response message.

XML processing error response example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
     <ResponseRequired>No</ResponseRequired>
   </WfMessageheader>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
      <RC>1105</RC>
      <Parameters>
         <Parameter>InsuredID</Parameter>
         <Parameter>Application</Parameter>
      </Parameters>
      <MessageText>FMC01105E Data member 'InsuredID' value of data structure
      'Application' has the wrong type.\n</MessageText>
      <Origin>d:\v32_67\src\fmcmctnm.cxx(340)</Origin>
    </Exception>
   </ProcessTemplateCreateAndStartInstanceResponse>
<WfMessage>
```

Errors when returning a response

When an XML response message is put into an MQSeries queue, an error can occur, for example, when the specified queue is undefined or when that queue is full.

Note: A similar error can occur when the UPES queue into which an ActivityImplInvoke message is to be put is not part of the MQ Workflow topology data.

When putting the response into the reply queue fails, the error and the first 500 bytes of the response are logged into an error log. The original XML message is committed and thus removed from the input queue. All *put* errors are assumed to be permanent, that is, it is assumed that all subsequent puts will also fail. Therefore, processing of the incoming message is not retried.

Example of an error log entry:

```
              MQSeries Workflow 3.2 Error Report

Report creation = 09.05.00 17:29:11

Error location  = File=e:\v322\src\fmccdxmm.cxx, Line=565,
Function=
FmcXMLMQDevice::Put(FmcDeviceCtxRef&,FmcDevDataRef&,FmcDeviceControler&)
Error data= FmcMQPUTException, MQ queue manager name=FMCQM, MQ queue name=DDDD,
            MQ completion code=2, MQ reason code=2085,
            Application data (frist 500 bytes)=
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>No</ResponseRequired>
   </WfMessageHeader>
   <UserContext>This data is sent back in response</UserContext>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
      <RC>1108</RC>
      <Parameters>
         <Parameter>ResponseRequired</Parameter>
```

Backout count exceeded

In general, the incoming XML message is committed after error handling is completed.

For internal errors like a database lock, the open transaction is, however, rolled back and the backout count is increased. When the backout count reaches a limit set in the runtime database, a GeneralError response message is put into the REPLY2Q and the incoming message is committed.

The error returned is:

```
:msgID.       FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED
:msgNum.      1106
:severity.    Error
:msgText.     "The backout count of the XML document is exceeded.
               The XML document cannot be processed.\\n"
```

The GeneralError message

A GeneralError response message is sent to inform the receiver that an error occurred and that a manual intervention might be required. No response is expected.

Following is an excerpt of the DTD describing the GeneralError XML message:

```
<!ELEMENT WfMessage
          WfMessageHeader?,
          GeneralError >
<!ELEMENT WfMessageHeader    (ResponseRequired?, UserContext?)   >
<!ELEMENT UserContext        (#PCDATA)                           >
<!ELEMENT ResponseRequired   (#PCDATA)                           >
          <!-- Expected values: {No,IfError,Yes}               -->
<!ELEMENT GeneralError       (Exception)                         >
<!ELEMENT Exception          (Rc?,Parameters?,MessageText,Origin)>
<!ELEMENT Parameters         (Parameter*)                        >
<!ELEMENT Parameter          (#PCDATA)                           >
<!ELEMENT Origin             (#PCDATA)                           >
```

For example,

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
           FMC01107E The MQMD format field value 'ABC ' of the XML document is
           incorrect.\n
        </MessageText>
        <Origin>p:\v322\src\fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

Chapter 33. The MQ Workflow XML message format

The following XML syntax is used to describe the format of MQ 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.

Note that 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.

```
<!-- FmcXMLIF.dtd == DTD for MQSeries Workflow messages -->
<!-- Message ======================================== -->
<!ELEMENT WfMessage
     ( WfMessageHeader?,
       ( ProcessTemplateCreateAndStartInstance
       | ProcessTemplateCreateAndStartInstanceResponse
       | ProcessTemplateExecute
       | ProcessTemplateExecuteResponse
       | ActivityExpired
       | ActivityImplInvoke
       | ActivityImplInvokeResponse
       | AuditTrailRecord
       | TerminateProgram
       | GeneralError ) ) >
<!-- ================================================
       Workflow Message Header
     ================================================ -->
<!ELEMENT WfMessageHeader           (ResponseRequired?,UserContext?)>
<!-- Opaque -->
<!ELEMENT UserContext               (#PCDATA)> <!-- Length<=254 bytes -->
<!-- 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 ) >
<!-- ActivityExpired ================================= -->
<!ELEMENT ActivityExpired
      ( ActImplCorrelID ) >
<!-- ActivityImplInvoke ============================== -->
<!ELEMENT ActivityImplInvoke
      ( ActImplCorrelID,
        Starter,
        ProgramID,
        (ImplementationData)*,
        ProgramInputData,
        ProgramOutputDataDefaults ) >
<!ELEMENT ActivityImplInvokeResponse
      ( ActImplCorrelID,
        (( ProgramRC,
           ProgramOutputData )
      | Exception )) >
<!-- AuditTrailRecord ================================ -->
<!ELEMENT AuditTrailRecord
      ( Timestamp,
        AuditEvent,
        ProcInstName,
        ProcInstID,
        ProcInstTopLevelName,
        ProcInstTopLevelID,
        ProcInstParentName?,
        ProcInstParentID?,
        ProcTemplName,
        ProcTemplValidFromDate?,
        BlockNames?,
        UserID?,
        SecondUserID?,
        ActivityName?,
        ActivityType?,
        ActivityState?,
        SecondActivityName?,
        CommandParameters?,
        AssociatedObject?,
        ObjectDescription?,
        ProgramName?,
        ProgramRC?  ) >
<!-- TerminateProgram ================================ -->
<!ELEMENT TerminateProgram
      ( ActImplCorrelID ) >
```

```
<!-- GeneralError ====================================== -->
<!ELEMENT GeneralError  (Exception) >
<!-- ================================================
          Data Structures
     ================================================ -->
<!ENTITY %STRING  "(#PCDATA)">
<!ENTITY %LONG    "(#PCDATA)">
<!ELEMENT null EMPTY>

<!ELEMENT _PROCESS_INFO
          ( Role?, Organization?, ProcessAdministrator?, Duration? )>
<!ELEMENT _ACTIVITY_INFO
          ( PRIORITY?, MembersOfRoles?, CoordinatorOfRole?,
            Organization?, OrganizationType?,
            LowerLevel?, UpperLevel?,
            People?, PersonToNotify?,
            Duration?, Duration2? )>
<!ELEMENT _ACTIVITY            %STRING;>
<!ELEMENT _PROCESS             %STRING;>
<!ELEMENT _PROCESS_MODEL       %STRING;>
<!ELEMENT _RC                  %LONG;>
<!ELEMENT ROLE                 %STRING;>
<!ELEMENT Organization         %STRING;>
<!ELEMENT ProcessAdministrator %STRING;>
<!ELEMENT Priority             %STRING;>
<!ELEMENT MembersOfRoles       %STRING;>
<!ELEMENT CoordinatorOfRole    %STRING;>
<!ELEMENT OrganizationType     %LONG;>
<!ELEMENT LowerLevel           %LONG;>
<!ELEMENT UpperLevel           %LONG;>
<!ELEMENT People               %STRING;>
<!ELEMENT PersonToNotify       %STRING;>
<!ELEMENT Duration             %LONG;>
<!ELEMENT Duration2            %LONG;>
<!ENTITY %_CONTAINER_INFO
         "_RC?, _PROCESS?, _PROCESS_MODEL?, _ACTIVITY?,
          _PROCESS_INFO?, _ACTIVITY_INFO?" >
<!-- ====================================================
      Sample Data Structure CreditData
==================================================== -->
<!ELEMENT CreditData
          ( Customer, Amount?, Currency?, Risk? ) >
<!ELEMENT Risk     %LONG;>
<!ELEMENT Currency %STRING;>
<!ELEMENT Amount   %LONG;>
<!ELEMENT Customer %STRING;>
<!-- ====================================================
      Sample Entity Container
      any used data structure must be included, for example,
      <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
==================================================== -->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">

<!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,
        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?) >
<!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 ActivityName              (#PCDATA) >
<!ELEMENT AssociatedObject          (#PCDATA) >
<!ELEMENT BackwardMappingFormat     (#PCDATA) >
<!ELEMENT BackwardMappingParameters (#PCDATA) >
<!ELEMENT BlockNames                (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT CommandParameters         (#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) >
<!ELEMENT MappingType               (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT ObjectDescription         (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT PathAndFileName           (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgramName               (#PCDATA) >
```

```
<!ELEMENT ProgramParameters         (#PCDATA) >
<!ELEMENT SecondActivityName        (#PCDATA) >
<!ELEMENT SecondUserID              (#PCDATA) >
<!ELEMENT ServiceName               (#PCDATA) >
<!ELEMENT ServiceType               (#PCDATA) >
<!ELEMENT Starter                   (#PCDATA) >
<!ELEMENT WorkingDirectoryName      (#PCDATA) >
<!-- Opaque -->
<!ELEMENT ActImplCorrelID           (#PCDATA) > <!-- Length =  80 bytes -->
<!ELEMENT ProcInstID                (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcInstParentID          (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcInstTopLevelID        (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcTemplID               (#PCDATA) > <!-- Length <= 64 bytes -->
<!-- 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) >
<!ELEMENT ProcTemplValidFromDate    (#PCDATA) >
<!ELEMENT Timestamp                 (#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 } -->
<!ELEMENT AuditEvent             (#PCDATA)  > <!-- Expected values:
                                                   { 21000, 21001, 21002 ....
                                                     see Administration Guide } -->

<!ELEMENT ActivityType           (#PCDATA)  > <!-- Expected values:
                                                   { Program Activity,
                                                     Process Activity,
                                                     Block Activity } -->
<!ELEMENT ActivityState          (#PCDATA)  > <!-- Expected values:
                                                   { 21200 (Ready),
                                                     21201 (Running),
```

```
21202 (Finished),
21203 (CheckedOut),
21204 (Force-Finished),
21205 (Terminated),
21206 (Suspended),
21207 (InError),
21208 (Executed),
21209 (Skipped),
21210 (Deleted),
21211 (Suspending),
21212 (Terminating),
21213 (Expired) } -->
```

Part 6. Using the MQ Workflow APIs

Chapter 34. Using the MQ Workflow Runtime API

Overview of the Runtime API

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

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

These programs typically use only a subset of the MQ Workflow API. For example, an activity implementation typically only accesses its containers, that is, only uses the so-called "Container API". The MQ Workflow API, that is, its header files and library structures or its ActiveX Controls or its import packages take this fact into account.

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

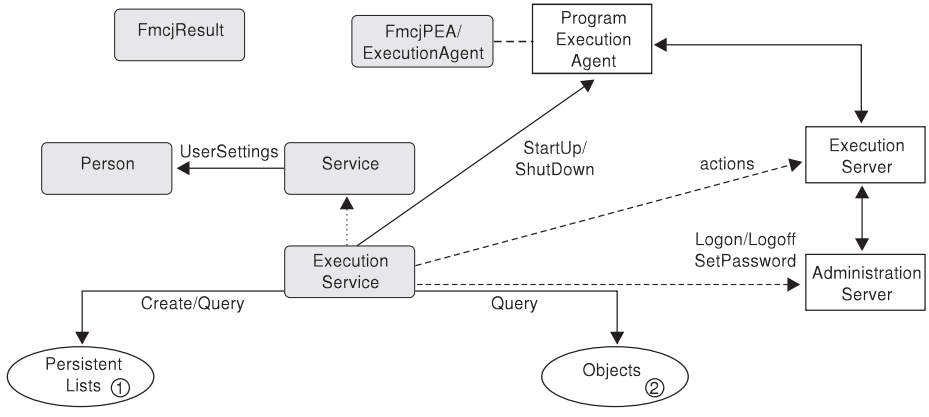


Figure 18. Setting up client/server communication. Legend: ⇢ Inheritance (C++); ⟶ provides for access; ⟶ sends messages to

As a first step, an ExecutionService object must be obtained (constructed/allocated/located). An ExecutionService object represents a session between a user and an MQ Workflow execution server. It essentially

provides the basic API calls to set up a communication path to the specified MQ Workflow execution server and to establish the user session (Logon() respectively 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 MQ Workflow database.
- Query information about the logged-on user or change that user's password.
- Start up respectively shut down a program execution agent associated to the logged-on user. This becomes necessary when work items are to be executed by MQ Workflow specific means.

In C and C++, all API call calls update a so-called result object. Detailed information about an erroneous request can be obtained from there. See "Handling errors" on page 10 for more information.

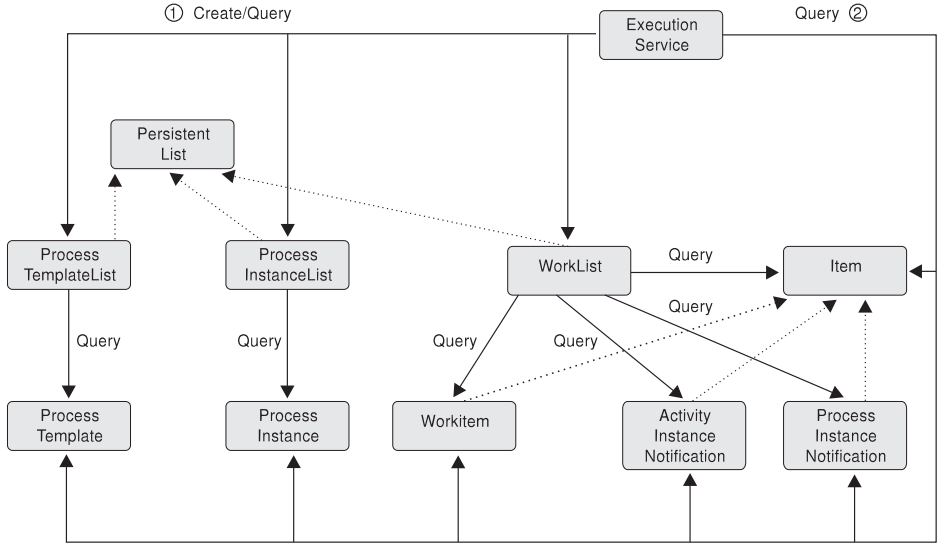


Figure 19. Querying objects. Legend: --► Inheritance (C++); ⟶ provides for access

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. Note that in

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 on 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.

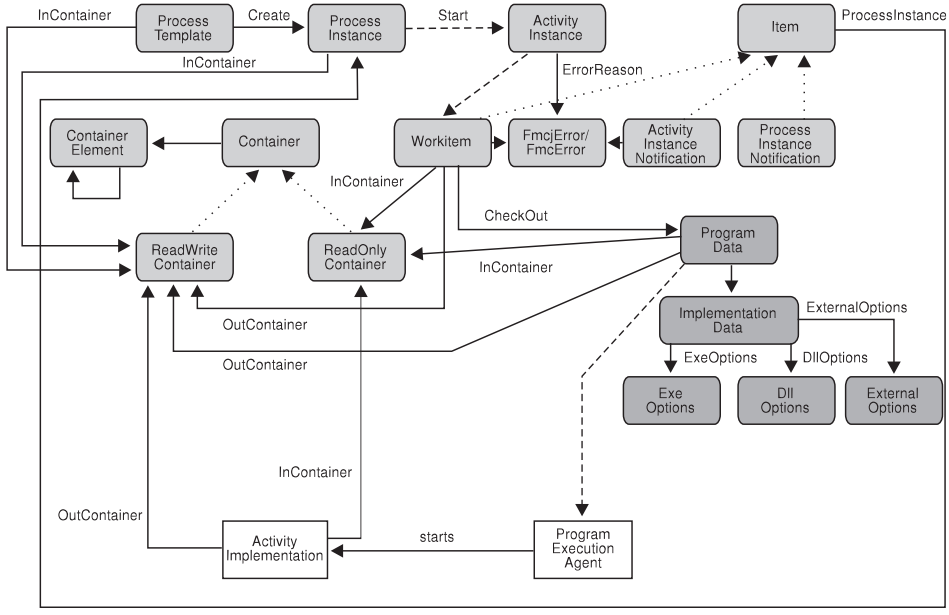


Figure 20. Dealing with process instances and (work) items. Legend: --➔ Inheritance (C++); ⟶ provides for access; — ⟶ data is passed to or results in

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 wrting values. See “Chapter 9. Handling containers” on page 45 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 MQ 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 MQ Workflow-specific means, that program data is automatically sent to the program execution agent 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 agent. 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.

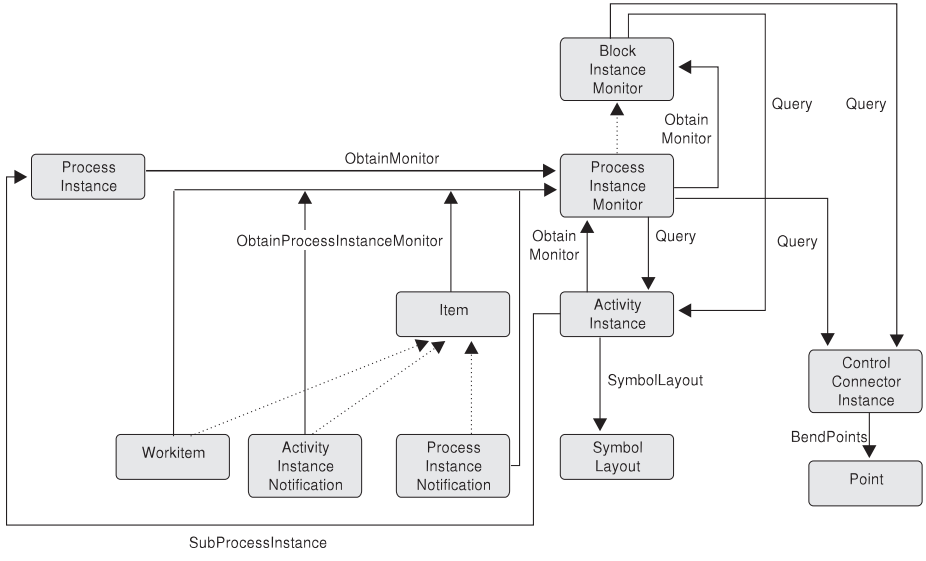


Figure 21. Monitoring a process instance. Legend: --➤ Inheritance (C++); ⟶ provides for access

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 monitors for activities of type Block or process instance monitors for activities of type Process, that is, for subprocess instances. See “Chapter 10. Monitoring a process instance” on page 79 for more information.

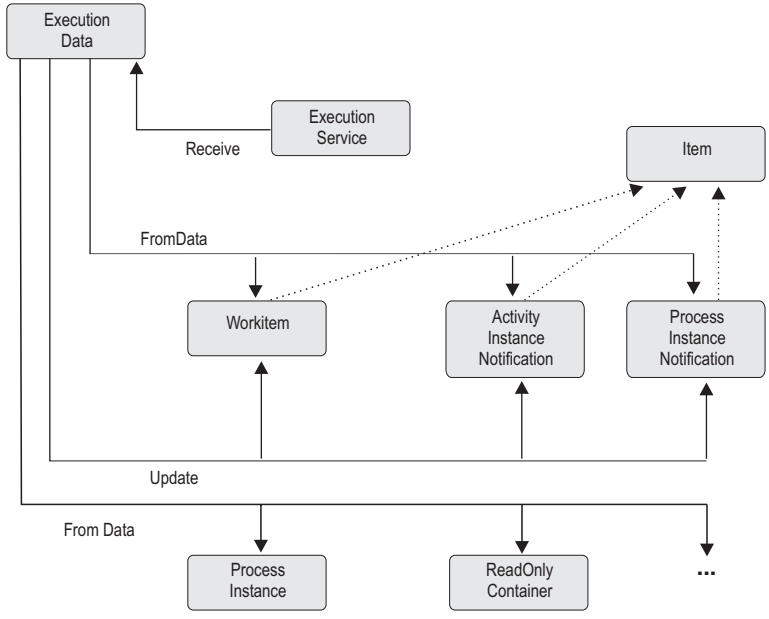


Figure 22. Handling data sent by an MQ Workflow server. Legend: --➤ Inheritance (C++); ⟶ provides for access

When the process setting specifies a *push refresh policy*, then the MQ Workflow execution server pushes changes on work items or notifications to a present client. In this case, or when the application issued 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 “Chapter 5. Client/server communication and data access models” on page 19 for more information.

API classes

An alphabetical list of API classes or function prefixes follows. Unless otherwise stated, all indicated names are valid ActiveX orJava classes. To become valid C++ classes, a prefix of Fmcj has to be added. To become valid C-language calls, the class name must be prefixed with Fmcj and extended by the actual function name. For example, if your Java class is Workitem, then your C++ class is named FmcjWorkitem and all your functions in the C-language start with FmcjWorkitem; FmcjWorkitemStart, for example, is a

valid C-language function.

Class/Object	Description
ActivityInstance	An instance of a workflow process template activity.
ActivityInstanceArray	The ActiveX result of a query for activity instances.
ActivityInstanceNotifArray	The ActiveX result of a query for activity instance notifications.
ActivityInstanceNotification	A notification associated with an activity instance.
ActivityInstanceNotificationVector	The C-language result of a query for activity instance notifications.
ActivityInstanceVector	The C-language result of a query for activity instances.
Agent	An agent in the Java API to access an MQ Workflow domain.
Container	The data container of a work item or a process instance.
ContainerArray	The ActiveX means of holding a container.
ContainerElement	An element of a data container.
ContainerElementArray	The ActiveX result of a query for container elements.
ContainerElementVector	The C-language result of a query for container elements.
ControlConnectorArray	The ActiveX result of a query for control connector instances.
ControlConnectorInstance	The instance of a control connector between two activity instances.
ControlConnectorInstanceVector	The C-language result of a query for control connector instances.
DateAndTime	Representation of date and time values. FmcjCDateTime is the C-language equivalent structure. The C++ class is called FmcjDateTime. Java uses the Calendar object.
DllOptions	The program implementation definitions for a dynamic link library.
ExecutionData	Information pushed by an MQ Workflow execution server or the response to an asynchronous request.

Class/Object	Description
ExecutionAgent	The Java representation of an MQ Workflow program execution agent. The C++ class is called FmcjPea.
ExecutionService	The representation of a session between a user and an MQ Workflow execution server so that services can be requested.
ExecutionServiceArray	The ActiveX means of holding an execution service.
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; the ActiveX class fmcError.
FmcException	The Java representation of an exception.
Global	A means to group API calls which are global API API calls in C and C++.
ImplementationData	The program implementation definitions.
InstanceMonitor	The monitor for a process instance or an activity instance of type *Block* or *Process*.
Item	An item associated to a user; can be a work item or notification; not available in ActiveX.
ItemVector	The C-language result of a query for items.
Message	A means to request an NLS regarding formatted message for a known message ID; only C-language and C++.
PersistentList	A list definition stored persistently; not available in ActiveX.
Person	User-specific settings for the user logged on to an MQ Workflow execution server.
Point	Describes the bend points of a control connector instance.
PointArray	The ActiveX result of a query for control connector instance bend points.
PointVector	The C-language result of a query for bend points.
ProcessInstance	An instance of a workflow process template.
ProcessInstanceList	A list to group process instances.

Class/Object	Description
ProcessInstanceListArray	The ActiveX result of a query for process instance lists.
ProcessInstanceListVector	The C-language result of a query for process instance lists.
ProcessInstanceNotifArray	The ActiveX result of a query for process instance notifications.
ProcessInstanceNotification	A notification associated with a process instance.
ProcessInstanceNotificationVector	The C-language result of a query for process instance notifications.
ProcessInstanceVector	The C-language result of a query for process instances.
ProcessTemplate	A workflow process template consisting of activities and containers and their control and data flow.
ProcessTemplateList	A list to group process templates.
ProcessTemplateListArray	The ActiveX result of a query for process template lists.
ProcessTemplateListVector	The C-language result of a query for process template lists.
ProcessTemplateVector	The C-language 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; not available in ActiveX.
ReadOnlyContainerHolder	In Java, a data container that contains the output container of a process instance; returned by executeProcessInstance().
ReadWriteContainer	A data container that can be read and written to; not available in ActiveX.
Result	The detailed result of a request; only C-language and C++.
Service	Provides for common aspects of MQ Workflow services; not available in ActiveX.
StringArray	The ActiveX result of a query resulting in a list of strings or the ActiveX means of providing a list of strings.

Class/Object	Description
StringVector	The C-language result of a query resulting in a list of strings or the C-language 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.
WorkitemArray	The ActiveX result of a query for work items.
WorkitemVector	The C-language result of a query for work items.
Worklist	A list to group work items or notifications.
WorklistArray	The ActiveX result of a query for worklists.
WorklistVector	The C-language result of a query for worklists.

API calls per class

Note: In the following descriptions, the basic methods listed are not differentiated by programming 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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an activity instance object.	94
Copy()	Allocates and initializes the storage for an activity instance object by copying.	99
Deallocate()	Deallocates the storage for an activity instance object.	99
destructor()	Destructs an activity instance object.	99
Equal()	Compares two activity instances.	97
IsComplete()	Indicates whether the complete activity instance information is available.	100
IsEmpty()	Indicates whether no activity instance information is available.	101
Kind()	States the kind of the activity instance, whether it is a program, a process, or a block.	101, 118

Basic methods	Description	Page
operator=()	Assigns an activity instance to this one.	97
operator==()	Compares two activity instances.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache. Note that the activity instances returned by an instance monitor fetched from the server 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.	110
ActivationTimeIsNull()	P/B	Indicates whether an activation time is set.	142
Category()	P/C	Returns the process category of the activity instance.	137
CategoryIsNull()	P/B	Indicates whether a category is set.	142
Description()	P/C	Returns the description of the activity instance.	137
DescriptionIsNull()	P/B	Indicates whether a description is set.	142
Documentation()	S/C	Returns the documentation of the activity instance.	137
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	142
EndTime()	S/D	Returns the ending time of the activity instance.	110

Accessor methods	Set/ Type	Description	Page
EndTimeIsNull()	S/B	Indicates whether an end time is set.	142
ErrorReason()	S/O	Returns an error object describing the reason why the activity instance is in state InError.	139
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	142
ExitCondition()	S/C	Returns the exit condition of the activity instance.	137
ExpirationTime()	S/D	Returns the expiration time of the activity instance.	110
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	142
FirstNotificationTime()	S/D	Returns the time the first notification for the activity instance is to occur or has occurred.	110
FirstNotificationTimeIsNull()	S/B	Indicates whether a first notification time is set.	142
FirstNotifiedPersons()	S/M	Returns the persons who received a first notification for the activity instance.	138
FullName()	P/C	Returns the fully qualified name of the activity instance (dot notation).	137
Icon()	P/C	Returns the icon associated with the activity instance.	137
Implementation()	P/C	Returns the name of the implementing program of the activity instance.	137
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	142
InContainerName()	S/C	Returns the name of the input container of the activity instance.	137
LastModificationTime()	P/D	Returns the last time a primary attribute of the activity instance was changed.	110
LastStateChangeTime()	P/D	Returns the last time the state of the activity instance changed.	110
ManualExitMode()	S/B	Returns whether the exit mode of the activity instance is manual.	109

Accessor methods	Set/ Type	Description	Page
ManualStartMode()	S/B	Returns whether the start mode of the activity instance is manual.	109
Name()	P/C	Returns the name of the activity instance.	137
OutContainerName()	S/C	Returns the name of the output container of the activity instance.	137
PersistentOid()	P/C	Returns a representation of the object identification of the activity instance.	137
Priority()	P/I	Returns the priority of the activity instance.	136
PriorityIsNull()	P/B	Indicates whether a priority is set.	142
ProcessAdmin()	S/C	Returns the process administrator of the activity instance.	137
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	142
ProcessInstanceName()	P/C	Returns the name of the process instance the activity instance is part of.	137
ProcessInstanceState()	P/E	Returns the state of the process instance the activity instance is part of.	111, 133
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	137
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the activity instance is part of.	137
SecondNotificationTime()	S/D	Returns the time the second notification for the activity instance is to occur or has occurred.	110
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	142
SecondNotifiedPersons()	S/M	Returns the persons who received a second notification for the activity instance.	138
Staff()	S/M	Returns all persons a work item for the activity instance is assigned to.	138

Accessor methods	Set/ Type	Description	Page
StartCondition()	S/C	Returns the start condition of the activity instance.	137
Starter()	P/C	Returns the starter of the activity instance or the requestor of a ForceFinish action.	137
StarterIsNull()	P/B	Indicates whether a starter is set.	142
StartTime()	P/D	Returns the start time of the activity instance.	110
StartTimeIsNull()	P/B	Indicates whether a start time is set.	142
State	P/E	Returns the state of the activity instance.	111, 115
StateOfNotification()	S/E	Returns the notification state of the activity instance.	111, 114
SupportTools()	P/M	Returns the support tools associated with the activity instance.	138
SymbolLayout()	S/O	Returns the symbol layout of the activity instance.	139

Refer to “Action API calls” on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
ForceFinish()	Force finishes the activity instance.	341
ForceFinishWithContainer()	In ActiveX, force finishes the activity instance and passes a container.	341
ForceFinish2()	In Java, force finishes the activity instance and passes a container.	341
ForceRestart()	Forces the restart of the activity instance.	344
ForceRestartWithContainer()	In ActiveX, force restarts the activity instance and passes a container.	344
ForceRestart2()	In Java, force restarts the activity instance and passes a container.	344
InContainer()	Retrieves the input container of the activity instance.	347
ObtainInstanceMonitor()	Retrieves the instance monitor for the process instance the activity instance is part of in ActiveX.	492

Action methods	Description	Page
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance the activity instance is part of; all languages but ActiveX.	349
OutContainer()	Retrieves the output container of the activity instance.	352
Refresh()	Refreshes the specified activity instance from the server.	354
SubProcessInstance()	Retrieves the process instance implementing the activity instance of type Process.	356
Terminate()	Terminates the specified activity instance.	359

ActivityInstanceArray

An activity instance array represents the result of a query for activity instances in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the activity instance array.	41

ActivityInstanceNotification

An activity instance notification represents a notification for an activity instance. **All API calls of FmcjItem are also applicable to activity instance notifications.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an activity instance notification object.	94
Copy()	Allocates and initializes the storage for an activity instance notification ocopybject by copying.	99
Deallocate()	Deallocates the storage for an activity instance notification object.	99
destructor()	Destructs an activity instance notification object.	99
Kind()	In the C++ language, states that the object is an activity instance notification.	101, 131

Basic methods	Description	Page
operator=()	Assigns an activity instance notification to this one.	97
operator==()	Compares two activity instance notifications.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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
ActivityKind()	P/E	Returns the kind of the associated activity instance, whether it is a program or process and so on.	101, 118
ErrorReason()	S/O	Returns an error object describing the reason why the associated activity instance is in state InError.	139
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	142
ExitCondition()	S/C	Returns the exit condition of the associated activity instance.	137
ExpirationTime()	S/D	Returns the expiration time of the associated activity instance.	110
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	142
FirstNotificationTime()	S/D	Returns the first notification time of the activity instance, that is, the time when this notification has been created.	110

Activity instance notification

Accessor methods	Set/ Type	Description	Page
Implementation()	P/C	Returns the implementing program or process name of the associated activity instance.	137
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	142
ManualExitMode()	S/B	Returns whether the exit mode of the associated activity instance is manual.	109
ManualStartMode()	S/B	Returns whether the start mode of the associated activity instance is manual.	109
PersistentOidOfActivityInstance()	P/C	Returns the object ID of the associated activity instance.	137
Priority()	P/I	Returns the priority of the associated activity instance.	136
SecondNotificationTime()	S/D	Returns the second notification time of the associated activity instance.	110
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	142
Staff()	S/M	Returns all persons owning a work item for the associated activity instance.	138
StartCondition()	S/C	Returns the start condition of the associated activity instance.	137
State	P/E	Returns the state of the associated activity instance.	111, 115
StateOfNotification()	S/E	Returns the notification state of the associated activity instance.	111, 114
SupportTools()	P/M	Returns the support tools associated with the activity instance.	138
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	142

Refer to “Action API calls” on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
ActivityInstance()	Retrieves the associated activity instance.	363

Action methods	Description	Page
StartTool()	Starts the specified support tool.	366
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	492
ObtainProcessMonitor()	Returns the instance monitor for the associated process instance; all languages but ActiveX.	492

ActivityInstanceNotificationArray

An activity instance notification array represents the result of a query for activity instance notifications in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

ActivityInstanceNotificationVector

An activity instance notification vector represents the result of a query for activity instance notifications in the C-language.

Refer to "C-language vectors" on page 35 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 the C-language.

Refer to "C-language vectors" on page 35 for detailed descriptions of vector access functions.

Activity instance vector

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 agent object represents an MQ Workflow instance in Java.

Refer to "Basic API calls" on page 93 for detailed descriptions of basicAPI calls.

Basic methods	Description	Page
constructor()	Constructs an agent object. Initially an agent has no context, locator policy, or name.	94

Refer to "Accessor/mutator API calls" on page 106 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.	145
addVetoableChangeListener()	O	Adds the specified listener to the set of listeners to be notified of vetoable property changes.	145
getConfigurationID()	C	Returns the configuration to be used for profile accesses.	137
getExecutionAgent()	O	Returns a program execution agent to the calling activity implementation provided that the LOC_LOCATOR policy was used. Otherwise, null is returned.	139

Accessor methods	Type	Description	Page
getLocator()	I	Returns the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR.	136
getName()	C	Returns the name of the agent. If the agent is not bound, an empty string is returned.	137
isBound()	B	Indicates whether the agent bean is bound to a Java CORBA agent.	109
locate()	O	Locates the execution service in the provided system group and system.	139
removePropertyChangeListener()	O	Removes the specified listener from the set of listeners.	145
removeVetoableChangeListener()	O	Removes the specified listener from the set of listeners.	145
setConfigurationID()	C	Sets the configuration ID to be used for profile access by the agent. A locator policy of LOC_LOCATOR is automatically set. Such, a java.beans.PropertyVetoException is thrown when the policy has already been set using the setLocator() method. Only one configuration can be used per application process. Configuration IDs for other than the LOC_LOCATOR policies are to be passed as command line parameters to the agent program.	137
setContext()	O	Sets the context of the agent. This call must precede a setName() call. An applet must set the context by issuing an *agent.setContext(this,null);*	139

Agent

Accessor methods	Type	Description	Page
setLocator()	I	Sets the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR. If LOC_LOCATOR is set, the default configuration ID for profile access is automatically used. Such, a java.beans.PropertyVetoException is thrown when the policy has already been set using the setConfigurationID() method. Configuration IDs for other than the LOC_LOCATOR policies are to be passed as command line parameters to the agent program. This call must precede a setName() call. **Note:** Java RMI agents should only be used for prototyping. They are currently not suited for production purposes.	142
setName()	C	Sets the name of the agent to be contacted as the result of this call.	137
toString()	C	Returns the name of the agent.	137
versionInfo()	C	Returns the version of the agent, that is, only useful formation is returned when the agent is bound.	137

Container

A container represents an input or output data container of a process instance or work item. **All accessor API calls of a container are applicable to read-only and read/write containers.**

Refer to “Basic API calls” on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
IsEmpty()	Indicates whether no container information is available.	101

Refer to “Accessor/mutator API calls” on page 106 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 MQ Workflow predefined members.	54
AllLeaves()	M	Returns all leaf elements of the container including the MQ Workflow predefined members.	54
ArrayBinaryLength()	I	Returns the length of the value of the specified container leaf element in the C-language. The leaf is part of an array and of type BINARY.	65
ArrayBinaryValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is part of an array and of type BINARY.	65
ArrayFloatValue()	F	Returns the value of the specified container leaf element in the C-language. The leaf is part of an array and of type FLOAT.	65
ArrayLongValue()	I	Returns the value of the specified container leaf element in the C-language. The leaf is part of an array and of type LONG.	66
ArrayStringLength()	I	Returns the length of the value of the specified container leaf element in the C-language. The leaf is part of an array and of type STRING.	66
ArrayStringValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is part of an array and of type STRING.	66
AsStream()	C	Returns the container as a binary stream of data.	137
BinaryLength()	I	Returns the length of the value of the specified container leaf element. The leaf is of type BINARY. Binaries are not supported in ActiveX.	65,66
BinaryValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is of type BINARY. Binaries are not supported in ActiveX.	65

Accessor methods	Type	Description	Page
FloatValue()	F	Returns the value of the specified container leaf element in the C-language. The leaf is of type FLOAT.	65
FromStream()	P	In ActiveX, constructs a transient container object from the stream passed.	140
getBuffer()	C	Returns the value of the specified container leaf element in Java. The leaf is of type BINARY.	67
getBuffer2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type BINARY.	67
getDouble()	F	Returns the value of the specified container leaf element in Java. The leaf is of type FLOAT.	67
getDouble2()	F	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type FLOAT.	67
GetElement()	O	Provides access to a container element.	63
getLong()	I	Returns the value of the specified container leaf element in Java. The leaf is of type LONG.	68
getLong2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type LONG.	68
getString()	C	Returns the value of the specified container leaf element in Java. The leaf is of type STRING.	68
getString2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type STRING.	68
GetValueDbl()	F	Returns the value of the specified container leaf element in ActiveX. The leaf is of type FLOAT.	64
GetValueLng()	I	Returns the value of the specified container leaf element in ActiveX. The leaf is of type LONG.	64
GetValueStr()	C	Returns the value of the specified container leaf element in ActiveX. The leaf is of type STRING.	64
LeafCount()	I	Returns the number of user-defined leaf elements of the container.	54

Accessor methods	Type	Description	Page
Leaves()	M	Returns all user-defined leaf elements of the container.	54
LongValue()	I	Returns the value of the specified container leaf element in the C-language. The leaf is of type LONG.	66
MemberCount()	I	Returns the number of structural members in the container.	55
SetStringCcsid()	C	Sets the CCSID to be used for reading or writing strings in the container.	137
SetValue()	F/C/I	Sets the value of the specified container element in ActiveX.	75
SetValueDbl()	F	Sets the value of the specified container leaf element in ActiveX. The leaf element is of type FLOAT.	73
SetValueLng()	F	Sets the value of the specified container leaf element in ActiveX. The leaf element is of type LONG.	73
SetValueStr()	F	Sets the value of the specified container leaf element in ActiveX. The leaf element is of type STRING.	73
StreamLength()	I	In the C and C++ languages, returns the length of the buffer needed to hold the container in stream format.	136
StringLength()	I	Returns the length of the value of the specified container leaf element in the C-language. The leaf is of type STRING.	66
StringValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is of type STRING.	66
StructMembers()	M	Returns the structural members of the container.	55
Type()	C	Returns the type of the container, that is, the data structure name.	56
Value()	C/I/F/N	Returns the value of the specified container leaf element in the C++ language.	66

Refer to “Activity implementation API calls” on page 151 for detailed descriptions of activity implementation API calls.

Container

Activity implementation methods	Description	Page
InContainer()	Accesses the input container from within an activity implementation; for Java see the ExecutionAgent.	369
OutContainer()	Accesses the output container from within an activity implementation; for Java see the ExecutionAgent.	371
RemoteInContainer()	Accesses the input container from within a program started by an activity implementation; for Java see the ExecutionAgent.	373
RemoteOutContainer()	Accesses the output container from within a program started by an activity implementation; for Java see the ExecutionAgent.	376
SetOutContainer()	Sets the output container from within an activity implementation; for Java see the ExecutionAgent.	378
SetRemoteOutContainer()	Sets the output container from within a program started by an activity implementation; for Java see the ExecutionAgent.	380

ContainerArray

A container array represents an array of containers in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	40
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41
RemoveAt()	Removes the element at the indicated position.	42

ContainerElement

A container element represents an arbitrary element of a container.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a container element object.	94
Copy()	Allocates and initializes the storage for a container element object by copying.	99
Deallocate()	Deallocates the storage for a container element object.	99
destructor()	Destructs a container element object.	99
Equal()	Compares two container elements.	97
operator=()	Assigns a container element to another one.	97
operator==()	Compares two container elements.	97
IsEmpty()	Indicates whether no container element information is available.	101

Refer to "Accessor/mutator API calls" on page 106 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 the C-language. The leaf is part of an array and of type BINARY.	69
ArrayBinaryValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type BINARY.	69
ArrayElements()	M	Returns the array elements of the container element.	62
ArrayFloatValue()	F	Returns the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type FLOAT.	70
ArrayLongValue()	I	Returns the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type LONG.	70

Accessor methods	Type	Description	Page
ArrayStringLength()	I	Returns the length of the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type STRING.	70
ArrayStringValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type STRING.	70
BinaryLength()	I	Returns the length of the value of the specified container element leaf element. The leaf is of type BINARY. Binaries are not supported in ActiveX.	69,71
BinaryValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is of type BINARY. Binaries are not supported in ActiveX.	69
Cardinality()	I	Returns the number of array elements of the container element.	62
FloatValue()	F	Returns the value of the specified container element leaf element in the C-language. The leaf is of type FLOAT.	70
FullName()	C	Returns the fully-qualified dotted name of the container element.	57
getBuffer()	C	Returns the value of the specified container element leaf element in Java. The leaf is of type BINARY.	72
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.	72
getDouble()	F	Returns the value of the specified container element leaf element in Java. The leaf is of type FLOAT.	72
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.	72
GetElement()	O	Provides access to an element of the container element.	63
getLong()	I	Returns the value of the specified container element leaf element in Java. The leaf is of type LONG.	72
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.	72

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.	72
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.	72
GetValueDbl()	F	Returns the value of the specified container leaf element in ActiveX. The leaf is of type FLOAT.	69
GetValueLng()	I	Returns the value of the specified container leaf element in ActiveX. The leaf is of type LONG.	69
GetValueStr()	C	Returns the value of the specified container leaf element in ActiveX. The leaf is of type STRING.	69
IsArray()	B	Indicates whether the container element is an array.	58
IsLeaf()	B	Indicates whether the container element is a leaf.	58
IsStruct()	B	Indicates whether the container element is a structure itself.	58
LeafCount()	I	Returns the number of leaf elements of the container element.	59
Leaves()	M	Returns all leaf elements of the container element.	59
LongValue()	I	Returns the value of the specified container element leaf element in the C-language. The leaf is of type LONG.	70
MemberCount()	I	Returns the number of structural members in the container element.	60
Name()	C	Returns the name of the container element.	57
StringLength()	I	Returns the length of the value of the specified container element leaf element in the C-language. The leaf is of type STRING.	70
StringValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is of type STRING..	70
StructMembers()	M	Returns the structural members of the container element.	60
Type()	C	Returns the type of the container element, that is, the data structure name.	57

Accessor methods	Type	Description	Page
Value()	C/I/F/N	Returns the value of the specified container element leaf element in the C++ language.	71

ContainerElementArray

A container element array represents the result of a query for container elements in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	40
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41
RemoveAt()	Removes the element at the indicated position.	42

ContainerElementVector

A container element vector represents the result of a query for container elements in the C-language.

Refer to "C-language vectors" on page 35 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.

ControlConnectorArray

A control connector array represents the result of a query for control connector instances in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

ControlConnectorInstance

A control connector instance object represents a control connector between two activity instances and its state.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a control connector instance object.	94
Copy()	Allocates and initializes the storage for a control connector instance object by copying.	99
Deallocate()	Deallocates the storage for a control connector instance object.	99
destructor()	Destructs a control connector instance object.	99
Equal()	Compares two control connector instance objects on the basis of their source and target activity instances.	97
IsEmpty()	Indicates whether no control connector instance information is available.	101
Kind()	States the kind of the control connector instance, whether it is a transition condition or the "otherwise" connector.	101, 120
operator=()	Assigns a control connector instance object to this one.	97
operator==()	Compares two control connector instance objects on the basis of their source and target activity instances.	97

Refer to "Accessor/mutator API calls" on page 106 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.	138

Control connector instance

Accessor methods	Type	Description	Page
Name()	C	Returns the name associated with the control connector instance.	137
NameIsNull()	B	Indicates whether a name is set.	142
PersistentOidOfSourceActivity()	C	Returns the object ID of the activity instance which is the source of this control connector instance.	137
PersistentOidOfTargetActivity()	C	Returns the object ID of the activity instance which is the target of this control connector instance.	137
State()	E	Returns the state of the control connector instance, whether it is evaluated, and the result of evaluation.	111, 119
TransitionCondition()	C	Returns the transition condition of the control connector instance.	137
TransitionConditionIsNull()	B	Indicates whether a transition condition is set.	142

ControlConnectorInstanceVector

A control connector instance vector represents the result of a query for control connector instances in the C-language.

Refer to "C-language vectors" on page 35 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.

DateAndTime/ FmcjDateTime/ FmcjCDateTime

A DateAndTime object represents date and time values in the ActiveX language. An FmcjDateTime object represents date and time values in the C++ language. An FmcjCDateTime structure represents date and time values in the C-language.

Date/time values are expressed in local time.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls. Following methods are only available in the C++ language.

Basic methods	Description	Page
constructor()	Constructs a date/time object.	94
destructor()	Destructs a date/time object.	99
operator=()	Assigns a date/time object to another one.	97
operator==()	Compares two date/time objects.	97
operator string()	Returns the string representation of the date/time object.	137
IsEmpty()	Indicates whether no date/time information is available.	101

Refer to "Accessor/mutator API calls" on page 106 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.

Following methods are only available in the C++ and ActiveX language.

Accessor methods	Type	Description	Page
Day()	I	Returns the day of the date/time object.	136
Hour()	I	Returns the hours of the date/time object.	136
Minute()	I	Returns the minutes of the date/time object.	136
Month()	I	Returns the month of the date/time object.	136
Second()	I	Returns the sceonds of the date/time object.	136
Year()	I	Returns the year of the date/time object.	136

Following methods are only available in the C-language.

Accessor functions	Type	Description	Page
FmcjDateTimeAsString	C	Returns the string representation of the date/time structure.	137
FmcjDateTimeCurrentTime	D	Returns the current date/time.	110

Accessor functions	Type	Description	Page
FmcjDateTimeIsValid	B	Indicates whether the passed date/time is a valid date/time; must be greater or equal 1970-1-1 12:00 (yyyy-mm-dd hh:mm).	109

DllOptions

A DllOptions object represents the program implementation definitions for a dynamic link library.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a DLL options object.	94
Copy()	Allocates and initializes the storage for a DLL options object by copying.	99
Deallocate()	Deallocates the storage for a DLL options object.	99
destructor()	Destructs a DLL options object.	99
IsEmpty()	Indicates whether no DLL options information is available.	101
operator=()	Assigns a DLL options object to this one.	97

Refer to "Accessor/mutator API calls" on page 106 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.	137
ExecuteFenced()	B	States whether the DLL should run in a separate address space.	109
ExecuteFencedIsNull()	B	Indicates whether execute fended is set.	142
KeepLoaded()	B	States whether the DLL should stay loaded.	109
KeepLoadedIsNull()	B	Indicates whether keep loaded is set.	142
PathAndFileName()	C	Returns the path and file name of the DLL.	137

ExecutionAgent/FmcjPEA

A PEA or ExecutionAgent object represents an MQ Workflow program execution agent.

Refer to "Activity implementation API calls" on page 151 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; for non-Java see the Container.	369
OutContainer()	Accesses the output container from within an activity implementation in Java; for non-Java see the Container.	371
PersistentOidOfActivityInstance()	Returns the object ID of the associated activity instance.	152
ProgramID()	Returns the program identification by which the invoked activity implementation or support tool is known to the program execution agent.	152
RemoteInContainer()	Accesses the input container from within a program started by an activity implementation in Java; for non-Java see the Container.	373
RemoteOutContainer()	Accesses the output container from within a program started by an activity implementation in Java; for non-Java see the Container.	376
RemotePersistentOid OfActivityInstance()	Returns the object ID of the activity instance on whose behalf the activity implementation who started this program was originally started.	152
RemoteUserID()	Returns the user identification on whose behalf the activity implementation who started this program was originally started.	152
UserID()	Returns the user identification on whose behalf the activity implementation was started.	152
SetOutContainer()	Sets the output container from within an activity implementation in Java; for non-Java see the Container.	378

Activity implementation methods	Description	Page
SetRemoteOutContainer()	Sets the output container from within a program started by an activity implementation in Java; for non-Java see the Container.	380

ExecutionData

An execution data object represents data sent from an MQ Workflow execution server.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an execution data object.	94
Copy()	Allocates and initializes the storage for an execution data object by copying.	99
Deallocate()	Deallocates the storage for an execution data object.	99
destructor()	Destructs an execution data object.	99
IsEmpty()	Indicates whether no execution data information is available.	101
Kind()	Returns the kind of the data, whether it is describing a work item creation, deletion, and so on.	101, 121
operator=()	Assigns an execution data object to this one.	97

Refer to "Accessor/mutator API calls" on page 106 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.	140
ErrorFromData()	P	Creates an error description object from the execution data.	140
IsError()	B	States whether the execution data describes an erroneous situation.	109

Accessor methods	Type	Description	Page
PersistentOid()	C	Returns a representation of the object ID of the object described by the execution data.	137
ProcessInstanceFromData()	P	Creates a process instance from the execution data.	140
ProcessInstanceNotificationFromData()	P	Creates a process instance notificationfrom the execution data.	140
ReadOnlyContainerFromData()	P	Creates a container object from the execution data.	140
WorkitemFromData()	P	Creates a work item from the execution data.	140
UserContext()	C	Returns the user context.	137
UserContextIsNull()	B	States whether a user context had been specified.	142

ExecutionService

An execution service object represents a user session to an execution server. **All API calls provided by FmcjService are also applicable.**

Refer to "Basic API calls" on page 93 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 MQ Workflow user's or configuration profile in the currently set configuration.	94
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.	94
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.	94
constructor()	Constructs an execution service object.	94
Copy()	Allocates and initializes the storage for an execution service object by copying.	99

Basic methods	Description	Page
Deallocate()	Deallocates the storage for an execution service object.	99
destructor()	Destructs an execution service object.	99
Equal()	Compares two execution service objects if they represent the same session.	97
operator=()	Assigns an execution service object to this one.	97
operator==()	Compares two execution service objects if they represent the same session.	97

Refer to "Accessor/mutator API calls" on page 106 for detailed descriptions of accessorAPI 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
PersistentActivityInstance()	P	Constructs a transient activity instance object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentActivityInstanceNotification()	P	Constructs a transient activity instance notification object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentInstanceMonitor()	P	Constructs a transient instance monitor object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentPerson()	P	Constructs a transient person object representing the persistent object identified by the passed user identification; does not contact the server.	140

Accessor methods	Type	Description	Page
PersistentProcessInstance()	P	Constructs a transient process instance object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentProcessInstanceList()	P	Constructs a transient process instance list object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentProcessInstanceNotification()	P	Constructs a transient process instance notification object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentProcessTemplate()	P	Constructs a transient process template object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentProcessTemplateList()	P	Constructs a transient process template list object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentWorkitem()	P	Constructs a transient work item object representing the persistent object identified by the passed object identification; does not contact the server.	140
PersistentWorklist()	P	Constructs a transient worklist object representing the persistent object identified by the passed object identification; does not contact the server.	140
ProgramDataFromStream()	P	Constructs a transient program data object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	140

Accessor methods	Type	Description	Page
ProgramTemplateFromStream()	P	Constructs a transient program template object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	140
ReadOnlyContainerFromStream()	P	Constructs a transient read-only container object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	140
ReadWriteContainerFromStream()	P	Constructs a transient read/write container object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	140
SessionID()	C	Returns a string representation of the session.	137
SetSessionContext()	C	Associates the execution service object with the session identified by the passed session identification.	144

Refer to “Action API calls” on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
CreateProcessInstanceList()	Creates a new process instance list on the execution server.	384
CreateProcessTemplateList()	Creates a new process template list on the execution server.	391
CreateWorklist()	Creates a new worklist on the execution server.	398

Action methods	Description	Page
Logoff()	Logs off from the connected execution server.	407
Logon(), logon()	Logs on to the execution server.	409
LogonWithCredentials()	Logs on to the execution server in C and ActiveX by passing user credentials.	409
LogonWithOptions()	Logs on to the execution server in ActiveX and provides additional parameters.	409
logon2()	Logs on to the execution server in Java and provides additional parameters.	409
logon3()	Logs on to the execution server in Java by passing user credentials.	409
logon4()	Logs on to the execution server in Java by passing user credentials and additional parameters.	409
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications the logged-on user has access to.	423
QueryItems()	Retrieves the work items or notifications the logged-on user has access to.	431
QueryProcessInstanceLists()	Retrieves the process instance lists the logged-on user has access to.	437
QueryProcessInstanceNotifications()	Retrieves the process instance notifications the logged-on user has access to.	440
QueryProcessInstances()	Retrieves the process instances the logged-on user has access to.	446
QueryProcessTemplateLists()	Retrieves the process template lists the logged-on user has access to.	451
QueryProcessTemplates()	Retrieves the process templates the logged-on user has access to.	454
QueryWorkitems()	Retrieves the work items the logged-on user has access to.	459
QueryWorklists()	Retrieves the worklists the logged-on user has access to.	466
Receive()	Receives execution data sent by an MQ Workflow execution server.	469
SetPersonAbsent()	Sets the specified person absent or not absent.	475

Execution service

Action methods	Description	Page
TerminateReceive()	Places information in the client input queue to indicate that receiving execution data sent by an MQ Workflow execution server can end.	477

Refer to "Activity implementation API calls" on page 151 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.	416
RemotePassthrough()	Establishes a session between a program started by an activity implementation and an execution server.	472

Refer to "Program execution management API calls" on page 155 for detailed descriptions of program execution management API calls.

Management methods	Description	Page
PEAShutDown()	Requests to shut down the user-associated program execution agent.	418
PEAStartUp()	Starts the user-associated program execution agent.	420

ExecutionServiceArray

An execution service array is an ActiveX means of holding an execution service.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array; the MQ Workflow user or configuration profile in the currently set configuration is searched for the specified system and system group.	40
AddDefault()	Adds the execution service to the array; system and system group are taken from the MQ Workflow user or configuration profile in the currently set configuration.	40

Accessor methods	Description	Page
AddSystemGroup()	Adds the execution service to the array; any system within the specified system group can be taken. It is taken from the MQ Workflow user or configuration profile in the currently set configuration.	40
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41
RemoveAt()	Removes the element at the indicated position.	42

Events	Description	Page
ExecutionServiceRemove()	Removes the execution service from the array.	43
NewExecutionService()	Adds a new execution service to the array.	43

ExeOptions

An ExeOptions object represents the program implementation definitions for an executable.

Refer to “Basic API calls” on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an EXE options object.	94
Copy()	Allocates and initializes the storage for an EXE options object by copying.	99
Deallocate()	Deallocates the storage for an EXE options object.	99
destructor()	Destructs an EXE options object.	99
operator=()	Assigns an EXE options object to this one.	97
IsEmpty()	Indicates whether no EXE options information is available.	101

Refer to “Accessor/mutator API calls” on page 106 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.

Exe options

Accessor methods	Type	Description	Page
AutomaticClose()	B	States whether the window in which the EXE starts should close when the EXE ends.	109
AutomaticCloseIsNull()	B	Indicates whether automatic close is set.	142
Environment()	C	States the environment settings for the EXE.	137
EnvironmentIsNull()	B	Indicates whether an environment is set.	142
InheritEnvironment()	B	States whether the environment settings should be merged with the operating system environment settings.	109
PathAndFileName()	C	Returns the path and file name of the EXE.	137
RunInXTerm()	B	States whether the EXE should start in a separate xterm.	109
RunInXTermIsNull()	B	Indicates whether run in xterm is set.	142
StartInForeGround()	B	States whether the EXE should start in the foreground.	109
StartInForeGroundIsNull()	B	Indicates whether start in foreground is set.	142
WindowStyle()	E	States the initial window style.	111, 123
WindowStyleIsNull()	B	Indicates whether a window style is set.	142
WorkingDirectoryName()	C	States the working directory for the EXE.	137
WorkingDirectoryNameIsNull()	B	Indicates whether a working directory is set.	142

ExternalOptions

An ExternalOptions object represents the program implementation definitions for an external service.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an External options object.	94

Basic methods	Description	Page
Copy()	Allocates and initializes the storage for an External options object by copying.	99
Deallocate()	Deallocates the storage for an External options object.	99
destructor()	Destructs an External options object.	99
operator=()	Assigns an External options object to this one.	97
IsEmpty()	Indicates whether no External options information is available.	101

Refer to "Accessor/mutator API calls" on page 106 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 MQ Workflow container.	137
BackwardMappingFormatIsNull()	B	Indicates whether a backward mapping format is set.	142
BackwardMappingParameters()	M	Returns backward mapping parameters, if any.	138
BackwardMappingParametersIsNull()	B	Indicates whether backward mapping parameters are set.	142
CodePage()	I	Specifies the code page of the service.	136
CodePageIsNull()	B	Indicates whether a code page is set.	142
ExecutableName()	C	Specifies the executable to be invoked by the invocation type and service.	137
ExecutableType()	C	Identifes the type of the executable.	137

Accessor methods	Type	Description	Page
ForwardMappingFormat()	C	Specifies the format for the mapping from an MQ Workflow container to the structure the executable uses.	137
ForwardMappingFormatIsNull()	B	Indicates whether a forward mapping format is set.	142
ForwardMappingParameters()	M	Returns forward mapping parameters, if any.	138
ForwardMappingParametersIsNull()	B	Indicates whether forward mapping parameters are set.	142
InvocationType()	C	Specifies the invocation mechanism to invoke the executable on the service.	137
IsLocalUser()	B	Returns whether a local user is to be resolved instead of using the MQ Workflow user ID.	109
IsMappingRoutineCall()	B	Specifies whether forward and backward mapping routines are to be called.	109
IsSecurityRoutineCall()	B	Specifies whether a security routine is to be called.	109
MappingType()	C	Identifies the type of mapping that should occur.	137
MappingTypeIsNull()	B	Indicates whether a mapping type is set.	142
ServiceName()	C	Identifies the service that is to be called.	137
ServiceType()	C	Identifies the type of service to be called, for example, CICS(R) or IMS(TM).	137
TimeoutPeriod()	E	Specifies how long the program execution agent should wait for a response from the started service, forever, a time period, or never.	111, 124
TimeoutInterval()	I	Specifies the timeout interval in seconds.	136
TimeoutIntervalIsNull()	B	Indicates whether a timeout interval is set.	142

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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an Error object.	94
Copy()	Allocates and initializes the storage for an Error object by copying.	99
Deallocate()	Deallocates the storage for an Error object.	99
destructor()	Destructs an Error object.	99
Equal()	Compares two Error objects on the basis of their return codes and parameters.	97
IsEmpty()	Indicates whether no Error information is available.	101
operator=()	Assigns an Error object to this one.	97
operator==()	Compares two Error objects on the basis of their return codes and parameters.	97

Refer to "Accessor/mutator API calls" on page 106 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 an NLS regarding formatted message.	137
Parameters()	M	Returns the parameters of the error; these are to be incorporated into the message text.	138
Rc()	I	Returns the return code remembered in the error object.	136

FmcException

An FmcException object represents a description of an exception thrown by Java.

FmcException

Refer to "Accessor/mutator API calls" on page 106 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 an NLS regarding formatted message.	137
nestedException()	-	Returns an exception thrown by the communication layer. **Note:** The nested exception can be inspected by (down-)casting to either org.omg.CORBA.SystemException or to java.rmi.RemoteException depending on the used communication protocol. 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.	137
origin()	C	Returns the module that threw the exception.	137
Parameters()	M	Returns the parameters of the error; these are already incorporated into the message text.	138
Rc()	I	Returns the return code remembered in the error object.	136

Global

An API global object serves to group global MQ Workflow API API calls.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description
Connect()	Initializes the API in the current thread; not supported in Java.
Disconnect()	Deinitializes the API in the current thread; not supported in Java.

Refer to "Accessor/mutator API calls" on page 106 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.	137
SetConfigurationID()	C	Sets the configuration ID to be used for profile access. Can only be set before the first profile usage; normally the Logon(). Only one configuration can be used per application process.	137

ImplementationData

An implementation data object represents the program implementation definitions.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an implementation data object.	94
Copy()	Allocates and initializes the storage for an implementation data object by copying.	99
Deallocate()	Deallocates the storage for an implementation data object.	99
destructor()	Destructs an implementation data object.	99
operator=()	Assigns an implementation data object to this one.	97
IsEmpty()	Indicates whether no implementation data information is available.	101
Kind()	States the actual kind of the implementation data, whether it is a DLL or an EXE.	101, 127

Refer to "Accessor/mutator API calls" on page 106 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.

Implementation data

Accessor methods	Type	Description	Page
CommandLineParameters()	C	Returns the command line parameters to be passed to the invoked program.	137
CommandLineParametersIsNull()	B	Indicates whether command line parameters are set.	142
DllOptions()	P	Returns the description of a DLL, if the implementation is a DLL.	140
ExeOptions()	P	Returns the description of an EXE, if the implementation is an EXE.	140
ExternalOptions()	P	Returns the description of external options, if the implementation is an external service.	140
options()	P	Returns the description of an EXE, a DLL, or an external service in Java.	140
Platform()	E	Returns the operating system platform this implementation data describes.	111, 125

ImplementationDataVector

An implementation data vector represents the result of a query for implementation data in the C-language.

Refer to "C-language vectors" on page 35 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates an implementation data vector object.
FirstElement()	Returns the first element of the implementation data vector.
NextElement()	Returns the next element of the implementation data vector.
Size()	Returns the number of elements in the implementation data vector.

InstanceMonitor

An instance monitor object represents a monitor in the API. It can be the monitor of a process instance, a monitor of an activity instance of type *Block*, or a monitor of an activity instance of type *Process*.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an instance monitor object.	94
Copy()	Allocates and initializes the storage for an instance monitor object by copying.	99
Deallocate()	Deallocates the storage for an instance monitor object.	99
destructor()	Destructs an instance monitor object.	99
Equal()	Compares two instance monitor objects.	97
operator=()	Assigns an instance monitor object to this one.	97
operator==()	Compares two instance monitor objects.	97

Refer to "Accessor/mutator API calls" on page 106 for detailed descriptions of accessor API calls. All properties are primary.

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 instance monitor. The activity instances contain both primary and secondary values.	138
ControlConnectorInstances()	M	Returns the control connector instances which are represented by the instance monitor.	138
PersistentOid()	P/C	Returns a representation of the object identification of the instance monitor.	137

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
ObtainBlockMonitor()	Returns the instance monitor for an activity instance of type *Block*; all languages but ActiveX. The activity instance is part of the set of activity instances represented by the instance monitor.	481

Instance monitor

Action methods	Description	Page
ObtainInstanceMonitor()	Returns the instance monitor for an activity instance of type *Block* or *Process* in ActiveX. The activity instance is part of the set of activity instances represented by the instance monitor.	481
ObtainProcessMonitor()	Returns the instance monitor for an activity instance of type *Process*; all languages but ActiveX. The activity instance is part of the set of activity instances represented by the instance monitor.	481
Refresh()	Refreshes the instance monitor from the MQ Workflow execution server.	484

Item

An item represents a work item, an activity instance notification, or a process instance notification. 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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an item object.	94
Copy()	Allocates and initializes the storage for an item object by copying.	99
Deallocate()	Deallocates the storage for an item object.	99
destructor()	Destructs an item object.	99
Equal()	Compares two items.	97
IsComplete()	Indicates whether the complete item information is available.	100
IsEmpty()	Indicates whether no item information is available.	101
Kind()	States the actual kind of the item, whether it is a work item or some kind of notification.	101, 131
operator=()	Assigns an item to this one.	97
operator==()	Compares two items.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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
Category()	P/C	Returns the process category of the item.	137
CategoryIsNull()	P/B	Indicates whether a category is set.	142
CreationTime()	P/D	Returns the creation time of the item.	110
Description()	P/C	Returns the description of the item.	137
DescriptionIsNull()	P/B	Indicates whether a description is set.	142
Documentation()	S/C	Returns the documentation of the item.	137
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	142
EndTime()	S/D	Returns the ending time of the item.	110
EndTimeIsNull()	S/B	Indicates whether an end time is set.	142
Icon()	P/C	Returns the icon associated with the item.	137
InContainerName()	S/C	Returns the name of the input container of the item.	137
LastModificationTime()	P/D	Returns the last time a primary attribute of the item was changed.	110

Accessor methods	Set/ Type	Description	Page
Name()	P/C	Returns the name of the item. In the C-language, 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.	137
OutContainerName()	S/C	Returns the name of the output container of the item.	137
Owner()	P/C	Returns the user ID of the owner of the item.	137
PersistentOid()	P/C	Returns a representation of the object identification of the item.	137
PersistentOidOfProcessInstance()	P/C	Returns the object ID of the associated process instance.	137
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the item.	137
ProcessInstanceName()	P/C	Returns the name of the process instance the item is part of.	137
ProcessInstanceState()	P/E	Returns the state of the process instance the item is part of.	111, 133
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	137
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the item is part of.	137
ReceivedAs()	P/E	Returns the reason why the item was received.	111, 112
ReceivedTime()	P/D	Returns the time when the item was received by the current owner.	110
StartTime()	S/D	Returns the start time of the item.	110
StartTimeIsNull()	S/B	Indicates whether a start time is set.	142

Mutator methods	Description	Page
Update()	Updates the item with the execution data sent by an MQ Workflow execution server. The object IDs of the item and of the object described by the execution data must match.	145

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes an item.	489
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance the item is part of.	492
ProcessInstance()	Retrieves the process instance the item is part of.	495
Refresh()	Retrieves the complete information of the item.	498
SetDescription()	Sets the description of the item.	500
SetName()	Sets the name of the item.	503
Transfer()	Transfers an item to the specified user.	505

ItemVector

An item vector represents the result of a query for items in the C-language.

Refer to "C-language vectors" on page 35 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 MQ Workflow provided message catalog.

Refer to "Accessor/mutator API calls" on page 106 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 indicated page.

Message

Accessor methods	Type	Description	Page
MessageText()	C	Returns an NLS regarding formatted message based on the message ID. Any parameters passed will be incorporated.	137

PersistentList

A persistent list represents a persistent list definition. **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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
IsEmpty()	Indicates whether no persistent list information is available.	101

Refer to "Accessor/mutator API calls" on page 106 for detailed descriptions of accessorAPI 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.	137
DescriptionIsNull()	B	Indicates whether a description is set.	142
Filter()	C	Returns the filter of the persistent list.	137
FilterIsNull()	B	Indicates whether a filter is set.	142
Name()	C	Returns the name of the persistent list.	137
OwnerOfList()	C	Returns the user ID of the owner of the persistent list.	137
OwnerOfListIsNull()	B	Indicates whether an owner is set; a public list does not have an owner.	142
PersistentOid()	C	Returns a representation of the object identification of the persistent list.	137
SortCriteria()	C	Returns the sort criteria of the persistent list.	137
SortCriteriaIsNull()	B	Indicates whether sort criteria are set.	142
Threshold()	I	Returns the threshold of the persistent list.	136

Accessor methods	Type	Description	Page
ThresholdIsNull()	B	Indicates whether a threshold is set.	142
Type()	E	Returns the type of the persistent list, whether it is a public or private list.	111, 132

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes the persistent list.	509
Refresh()	Refreshes the persistent list.	512
SetDescription()	Sets the description of the persistent list.	514
SetFilter()	Sets the filter of the persistent list.	517
SetSortCriteria()	Sets the sort criteria of the persistent list.	520
SetThreshold()	Sets the threshold of the persistent list.	522

Person

A person object represents the settings of the logged-on user.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a person object.	94
Copy()	Allocates and initializes the storage for a person object by copying.	99
Deallocate()	Deallocates the storage for a person object.	99
destructor()	Destructs a person object.	99
Equal()	Compares two persons.	97
operator=()	Assigns a person to this one.	97
operator==()	Compares two persons.	97
IsComplete()	Indicates whether the complete person information is available.	100
IsEmpty()	Indicates whether no person information is available.	101

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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.	138
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.	138
Description()	P/C	Returns the description of the person.	137
DescriptionIsNull()	P/B	Indicates whether a description is set.	142
FirstName()	P/C	Returns the first name of the person.	137
FirstNameIsNull()	P/B	Indicates whether a first name is set.	142
IsAbsent()	P/B	Indicates whether the person is absent.	109

Accessor methods	Set/ Type	Description	Page
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.	109
IsAdministrator()	S/B	Indicates whether the person is an administrator.	109
IsAuthorizedForAllCategories()	P/B	Indicates whether the person is said to be authorized for all categories either with basic and/or administration rights.	109
IsAuthorizedForAllCategoriesAsAdmin()	P/B	Indicates whether the person is said to be authorized for all categories as administrator.	109
IsAuthorizedForAllPersons()	P/B	Indicates whether the person is authorized to see the items of all persons.	109
IsAuthorizedForAuthorizationDefinition()	P/B	Indicates whether the person is authorized to define authorizations.	109
IsAuthorizedForOperationAdministration()	P/B	Indicates whether the person is authorized for operational administrations.	109
IsAuthorizedForProcessDefinition()	P/B	Indicates whether the person is authorized to define process models.	109
IsAuthorizedForStaffDefinition()	P/B	Indicates whether the person is authorized to define persons.	109
IsAuthorizedForTopologyDefinition()	P/B	Indicates whether the person is authorized to define topological data.	109
IsManager()	S/B	Indicates whether the person is a manager.	109

Accessor methods	Set/ Type	Description	Page
IsResetAbsence()	P/B	Indicates whether the absence flag should be reset when the person logs on.	109
LastName()	P/C	Returns the last name of the person.	137
LastNameIsNull()	P/B	Indicates whether a last name is set.	142
Level()	P/I	Returns the level of the person.	136
Manager()	S/C	Returns the user identification of the person's manager.	137
ManagerIsNull()	S/B	Indicates whether the person's manager is set.	142
MiddleName()	P/C	Returns the middle name of the person.	137
MiddleNameIsNull()	P/B	Indicates whether a middle name is set.	142
NamesOfManagedOrganizations()	S/M	Returns the names of organizations the person manages.	138
NamesOfRoles()	P/M	Returns the names of roles the person belongs to.	138
NamesOfRolesToCoordinate()	S/M	Returns the names of roles the person can coordinate.	138
OrganizationName()	P/C	Returns the name of the organization the person belongs to.	137
OrganizationNameIsNull()	P/B	Indicates whether an organization name is set.	142
PersonID()	P/C	Returns the person ID of the person.	137
PersonIDIsNull()	P/B	Indicates whether a person ID is set.	142

Accessor methods	Set/ Type	Description	Page
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.	138
PersonsAuthorizedForMe()	S/M	Returns the persons who are authorized for this person.	138
PersonsToStandInFor()	S/M	Returns the persons for whom this person stands in.	138
Phone()	P/C	Returns the phone number of the person.	137
PhoneIsNull()	P/B	Indicates whether a phone is set.	142
SecondPhone()	P/C	Returns the alternate phone number of the person.	137
SecondPhoneIsNull()	P/B	Indicates whether an alternate phone is set.	142
Substitute()	P/C	Returns the substitute of the person.	137
SubstituteIsNull()	P/B	Indicates whether a substitute is set.	142
SystemName()	P/C	Returns the home system of the person.	137
UserID()	P/C	Returns the user identification of the person.	137

Refer to “Action API calls” on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
Refresh()	Retrieves the complete person information from the server.	527
SetAbsence()	Sets the absent flag of the logged-on user to the specified value.	529
SetSubstitute()	Sets the substitute of the logged-on user to the specified value.	531

Point

A point object represents a bend point of a control connector.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a point object.	94
Copy()	Allocates and initializes the storage for a point object by copying.	99
Deallocate()	Deallocates the storage for a point object.	99
destructor()	Destructs a point object.	99
Equal()	Compares two point objects on the basis of their contents.	97
IsEmpty()	Indicates whether no point information is available.	101
operator=()	Assigns a point object to this one.	97
operator==()	Compares two point objects on the basis of their contents.	97

Refer to "Accessor/mutator API calls" on page 106 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.	136
YPosition()	I	Returns the y-coordinate of the point.	136

PointArray

A point array represents the result of a query for bend points in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

PointVector

A point vector represents the result of a query for points in the C-language.

Refer to "C-language vectors" on page 35 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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance object.	94
Copy()	Allocates and initializes the storage for a process instance object by copying.	99
Deallocate()	Deallocates the storage for a process instance object.	99
destructor()	Destructs a process instance object.	99
Equal()	Compares two process instances.	97
IsComplete()	Indicates whether the complete process instance information is available.	100
IsEmpty()	Indicates whether no process instance information is available.	101
operator=()	Assigns a process instance to this one.	97
operator==()	Compares two process instances.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by

issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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.	111, 113
Category()	P/C	Returns the category of the process instance.	137
CategoryIsNull()	P/B	Indicates whether a category is set.	142
CreationTime()	S/D	Returns the creation time of the process instance.	110
Creator()	S/C	Returns the creator of the process instance.	137
Description()	P/C	Returns the description of the process instance.	137
DescriptionIsNull()	P/B	Indicates whether a description is set.	142
Documentation()	S/C	Returns the documentation of the process instance.	137
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	142
EndTime()	S/D	Returns the end time of the process instance.	110
EndTimeIsNull()	S/B	Indicates whether an end time is set.	142
Icon()	P/C	Returns the icon associated with the process instance.	137
InContainerName()	S/C	Returns the name of the input container of the process instance.	137

Accessor methods	Set/ Type	Description	Page
InContainerNeeded()	P/B	Indicates whether an input container is needed to start the process instance. An input container is needed when • There is a mapping to some other container. • Staff assignment data is taken from it. • Notification related data is taken from it. • A transition or exit condition refers to a container element. • A description refers to a container element. • *Prompt for data at process start* is set for the process model.	109
LastModificationTime()	P/D	Returns the last time a primary attribute of the process instance was changed.	110
LastStateChangeTime()	P/D	Returns the last time the state of the process instance was changed.	110
Name()	P/C	Returns the name of the process instance.	137
NotificationTime()	S/D	Returns the notification time of the process instance.	110
NotificationTimeIsNull()	S/B	Indicates whether a notification time is set.	142
NotifiedPerson()	S/C	Returns the person who received the notification.	137
NotifiedPersonIsNull()	S/B	Indicates whether a notified person is set.	142
OrganizationName()	S/C	Returns the name of the organization of the process instance.	137
OrganizationNameIsNull()	S/B	Indicates whether an organization name is set.	142
OutContainerName()	S/C	Returns the name of the output container of the process instance.	137

Process instance

Accessor methods	Set/ Type	Description	Page
ParentName()	P/C	Returns the name of the parent process instance of this process instance.	137
ParentNameIsNull()	P/B	Indicates whether a parent name is set.	142
PersistentOid()	P/C	Returns a representation of the object identification of the process instance.	137
PersistentOidOfProcessTemplate()	P/C	Returns a representation of the object identification of the process template the process instance is derived from.	137
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the process instance.	137
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	142
ProcessTemplateName()	P/C	Returns the name of the process template the process instance is derived from.	137
RoleName()	S/C	Returns the name of the role of the process instance.	137
RoleNameIsNull()	S/B	Indicates whether a role is set.	142
Starter()	S/C	Returns the starter of the process instance.	137
StarterIsNull()	S/B	Indicates whether a starter is set.	142
StartTime()	S/D	Returns the start time of the process instance.	110
StartTimeIsNull()	S/B	Indicates whether a start time is set.	142
State()	P/E	Returns the state of the process instance.	111, 133
StateOfNotification()	S/E	Returns the notification state of the process instance.	111, 132
SuspensionExpirationTime()	P/D	Returns the suspension expiration time of the process instance.	110
SuspensionExpirationTimeIsNull()	P/B	Indicates whether the suspension expiration time is set.	142

Accessor methods	Set/ Type	Description	Page
SuspensionTime()	P/D	Returns the time the process instance was suspended.	110
SuspensionTimeIsNull()	P/B	Indicates whether the suspension time is set.	142
SystemGroupName()	P/C	Returns the name of the system group where the process instance runs.	137
SystemName()	P/C	Returns the name of the system where the process instance runs.	137
TopLevelName()	P/C	Returns the name of the top level process instance of this process instance.	137

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes the process instance.	535
InContainer()	Retrieves the input container of the process instance.	538
ObtainMonitor()	Retrieves the instance monitor for the process instance in ActiveX.	540
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance; all languages but ActiveX.	540
OutContainer()	Retrieves the output container of the process instance.	543
Refresh()	Retrieves the complete information of the process instance.	545
Restart()	Restarts the process instance.	548
Resume()	Resumes the execution of a suspended process instance.	550
SetDescription()	Sets the description of the process instance.	552
SetName()	Sets the name of the process instance.	555
Start()	Starts the process instance.	557
Start2()	Starts the process instance in Java and provides an input container.	557
Suspend()	Suspends the process instance.	560

Action methods	Description	Page
Suspend2()	Suspends the process instance in Java until the specified calendar date.	560
SuspendUntil()	Suspends the process instance until the specified time.	560
Terminate()	Terminates the process instance.	563

ProcessInstanceList

A process instance list represents a group of process instances. **All API calls of a persistent list are also applicable to process instance lists.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance list object.	94
Copy()	Allocates and initializes the storage for a process instance list object by copying.	99
Deallocate()	Deallocates the storage for a process instance list object.	99
destructor()	Destructs a process instance list object.	99
Equal()	Compares two process instance lists.	97
operator=()	Assigns a process instance list to this one.	97
operator==()	Compares two process instance lists.	97

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryProcessInstances()	Retrieves the process instances qualifying via the process instance list.	567

ProcessInstanceListArray

A process instance list array represents the result of a query for process instance lists in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41

Accessor methods	Description	Page
GetSize()	Returns the number of elements in the array.	41

Events	Description	Page
NewProcessInstanceList()	Adds a new process instance list to the array.	43
ProcessInstanceListRemove()	Removes the process instance list from the array.	43

ProcessInstanceListVector

A process instance list vector represents the result of a query for process instance lists in the C-language.

Refer to "C-language vectors" on page 35 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.

ProcessInstanceNotification

A process instance notification represents a notification raised for a process instance. **All API calls of an FmcjItem are also applicable to process instance notifications.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance notification object.	94
Copy()	Allocates and initializes the storage for a process instance notification object by copying.	99
Deallocate()	Deallocates the storage for a process instance notification object.	99
destructor()	Destructs a process instance notification object.	99
Kind()	In the C++ language, states that the object is a process instance notification.	101, 131

Process instance notification

Basic methods	Description	Page
operator=()	Assigns a process instance notification to this one.	97
operator==()	Compares two process instance notifications.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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
NotificationTime()	S/D	Returns the notification time of the process instance.	110

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	492

ProcessInstanceNotificationArray

A process instance notification array represents the result of a query for process instance notifications in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41

Accessor methods	Description	Page
GetSize()	Returns the number of elements in the array.	41

ProcessInstanceNotificationVector

A process instance notification vector represents the result of a query for process instance notifications in the C-language.

Refer to "C-language vectors" on page 35 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 the C-language.

Refer to "C-language vectors" on page 35 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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process template object.	94
Copy()	Allocates and initializes the storage for a process template object by copying.	99
Deallocate()	Deallocates the storage for a process template object.	99
destructor()	Destructs a process template object.	99
Equal()	Compares two process templates.	97
IsComplete()	Indicates whether the complete process template information is available.	100
IsEmpty()	Indicates whether no process template information is available.	101
operator=()	Assigns a process template to this one.	97
operator==()	Compares two process templates.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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.	111, 113
Category()	P/C	Returns the category of the process template.	137
CategoryIsNull()	P/B	Indicates whether a category is set.	142

Accessor methods	Set/ Type	Description	Page
CreationTime()	P/D	Returns the creation time of the process template.	110
Description()	P/C	Returns the description of the process template.	137
DescriptionIsNull()	P/B	Indicates whether a description is set.	142
Documentation()	S/C	Returns the documentation of the process template.	137
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	142
Icon()	P/C	Returns the icon associated with the process template.	137
InContainerName()	S/C	Returns the name of the input container of the process template.	137
InContainerNeeded()	P/B	Indicates whether an input container is needed to start an instance of the process template. An input container is needed when • 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. • *Prompt for data at process start* is set for the process model.	109
LastModificationTime()	P/D	Returns the last time a primary attribute of the process template was changed.	110
Name()	P/C	Returns the name of the process template.	137
OrganizationName()	S/C	Returns the name of the organization of the process template.	137
OrganizationNameIsNull()	S/B	Indicates whether an organization name is set.	142
OutContainerName()	S/C	Returns the name of the output container of the process template.	137
PersistentOid()	P/C	Returns a representation of the object identification of the process template.	137

Process template

Accessor methods	Set/ Type	Description	Page
ProcessAdmin()	S/C	Returns the user ID of the process administrator of an instance of the process template.	137
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	142
RoleName()	S/C	Returns the name of the role of the process template.	137
RoleNameIsNull()	S/B	Indicates whether a role is set.	142
ValidFromTime()	P/D	Returns the time when the process template becomes valid.	110

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
CreateAndStartInstance()	Creates and starts an instance of the process template.	571
CreateAndStartInstance2()	Creates and starts an instance of the process template in Java and provides an input container.	571
CreateInstance()	Creates an instance of the process template.	579
Delete()	Deletes the specified process template.	582
Delete2()	Deletes the specified process template versions in Java.	582
ExecuteProcessInstance()	Creates and executes an instance from the specified process template.	585
ExecuteProcessInstanceAsync()	Creates and executes an instance from the specified process template; an answer is not waited for.	585
InitialInContainer()	Retrieves the initially defined input container of the process template.	595
ProgramTemplate()	Retrieves the program template specified by the passed name.	597
Refresh()	Retrieves the complete information of the process template.	600

ProcessTemplateList

A process template list represents a group of process templates. **All API calls of a persistent list are also applicable to process template lists.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process template list object.	94
Copy()	Allocates and initializes the storage for a process template list object by copying.	99
Deallocate()	Deallocates the storage for a process template list object.	99
Equal()	Compares two process template lists.	97
destructor()	Destructs a process template list object.	99
operator=()	Assigns a process template list to this one.	97
operator==()	Compares two process template lists.	97

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryProcessTemplates()	Retrieves the process templates qualifying via the process template list.	603

ProcessTemplateListArray

A process template list array represents the result of a query for process template lists in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

Events	Description	Page
NewProcessTemplateList()	Adds a new process template list to the array.	43
ProcessTemplateListRemove()	Removes the specified process template list from the array.	43

ProcessTemplateListVector

A process template list vector represents the result of a query for process template lists in the C-language.

Process template list vector

Refer to "C-language vectors" on page 35 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 the C-language.

Refer to "C-language vectors" on page 35 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 program template.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a program data object.	94
Copy()	Allocates and initializes the storage for a program data object by copying.	99
Deallocate()	Deallocates the storage for a program data object.	99
destructor()	Destructs a program data object.	99

Basic methods	Description	Page
Equal()	Compares two program data objects if they belong to the same work item.	97
IsEmpty()	Indicates whether no program data information is available yet.	101
operator=()	Assigns a program data object to this one.	97
operator==()	Compares two program data objects if they belong to the same work item.	97

Refer to "Accessor/mutator API calls" on page 106 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
AsStream()	C	Returns the program data as a binary stream.	137
Description()	C	Returns the description of the implementing program.	137
DescriptionIsNull()	B	Indicates whether a description is set.	142
ExecutionMode()	E	States whether the program can participate in global transactions or not.	111, 122
ExecutionUser()	E	Returns the user on whose behalf the program is to be executed.	111, 123
FromStream()	P	In ActiveX, constructs a transient program data object from the stream passed.	140
Icon()	C	Returns the icon associated with the implementing program.	137
Implementations()	M	Returns the implementation definitions of the program.	138
InContainer()	P	Returns the input container of the program.	140
IsUnattended()	B	States whether the program can run unattended.	109
OutContainer()	P	Returns the output container of the program.	140

Program data

Accessor methods	Type	Description	Page
ProgramTrusted()	B	States whether the program can be trusted. Only a trusted program can receive its program ID.	109
StreamLength()	C	In the C and C++ languages, returns the length of the buffer needed to hold the program data in stream format.	137

ProgramTemplate

A program template object represents the program implementation definitions.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a program template object.	94
Copy()	Allocates and initializes the storage for a program template object by copying.	99
Deallocate()	Deallocates the storage for a program template object.	99
destructor()	Destructs a program template object.	99
Equal()	Compares two program template objects on the basis of their names and the process template they belong to.	97
IsEmpty()	Indicates whether no program template information is available yet.	101
operator=()	Assigns a program template object to this one.	97
operator==()	Compares two program template objects on the basis of their names and the process template they belong to.	97

Refer to "Accessor/mutator API calls" on page 106 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
AsStream()	C	Returns the program template as a binary stream.	137

Accessor methods	Type	Description	Page
Description()	C	Returns the description of the implementing program.	137
DescriptionIsNull()	B	Indicates whether a description is set.	142
ExecutionMode()	E	States whether the program can participate in global transactions or not.	111, 122
ExecutionUser()	E	Returns the user on whose behalf the program is to be executed.	137, 123
FromStream()	P	In ActiveX, constructs a transient program template object from the stream passed.	140
Icon()	C	Returns the icon associated with the implementing program.	137
Implementations()	M	Returns the implementation definitions of the program.	138
InitialInContainer()	P	Returns the initially defined input container of the program.	140
InContainerAccess()	B	States whether the input container is accessed by the program.	109
IsUnattended()	B	States whether the program can run unattended.	109
InitialOutContainer()	P	Returns the initially defined output container of the program.	140
OutContainerAccess()	B	States whether the output container is accessed by the program.	109
ProgramTrusted()	B	States whether the program can be trusted. Only a trusted program can receive its program ID.	109
StreamLength()	C	In the C and C++ languages, returns the length of the buffer needed to hold the program template in stream format.	137
StructuresFromActivity()	B	States whether the program can handle any container passed to it.	109
ValidFromTime()	P/D	Returns the time when the process template and thus the program template becomes valid.	110

Refer to “Action API calls” on page 150 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.	607
Execute2()	Requests the execution of the program by the system's program execution server.	607
ExecuteAsync()	Requests the execution of the program by the system's program execution server; an answer is not waited for.	607
ExecuteAsync2()	Requests the execution of the program by the system's program execution server; an answer is not waited for.	607
ExecuteAsyncWithOptions()	Requests the execution of the program by the system's program execution server; an answer is not waited for. The priority of the program can be specified.	607
ExecuteWithOptions()	Requests the execution of the program by the system's program execution server. The priority of the program can be specified.	607

ReadOnlyContainer

A read-only container represents a container that can only be read, for example, an input data container of a work item or an output container of a process instance; all languages but ActiveX. **All API calls of a container are applicable to read-only containers.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
AsReadWriteContainer()	In the C-language and Java, converts the read-only container into a read/write container.	98
constructor()	Constructs a read-only container object.	94
Copy()	Allocates and initializes the storage for a read-only container object by copying.	99
Deallocate()	Deallocates the storage for a read-only container object.	99
Equal()	Compares two read-only containers.	97
destructor()	Destructs a read-only container object.	99
operator=()	Assigns a read-only container to this one.	97
operator==()	Compares two read-only containers.	97

Basic methods	Description	Page
operator FmcjReadWriteContainer()	In C++, converts the read-only container into a read/write container.	98

ReadOnlyContainerHolder

A read-only container holder represents an object in Java that can contain a read-only container, for example, an output container of an executed process instance.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a read-only container holder object. Optionally, an already existing read-only container can be passed.	94

Refer to "Accessor/mutator API calls" on page 106 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 (O). The API call declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
value()	O	Returns the read-only container contained in the holder object.	139

ReadWriteContainer

A read/write container represents a container that can be read and written, for example, an input container of a process instance or an output container of a work item; all languages but ActiveX. **All API calls of a container are applicable to read/write containers.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
AsReadOnlyContainer()	In the C-language and Java, converts the read/write container into a read-only container.	98

Basic methods	Description	Page
constructor()	Constructs a read/write container object.	94
Copy()	Allocates and initializes the storage for a read/write container object by copying.	99
Deallocate()	Deallocates the storage for a read/write container object.	99
Equal()	Compares two read/write containers.	97
destructor()	Destructs a read/write container object.	99
operator=()	Assigns a read/write container to another one.	97
operator==()	Compares two read/write containers.	97
operator FmcjReadOnlyContainer()	In C++, converts the read/write container into a read-only container.	98

Refer to "Accessor/mutator API calls" on page 106 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 the C-language. The leaf element is part of an array and of type BINARY.	73
SetArrayFloatValue()	F	Sets the value of the specified container leaf element in the C-language. The leaf element is part of an array and of type FLOAT.	74
SetArrayLongValue()	I	Sets the value of the specified container leaf element in the C-language. The leaf element is part of an array and of type LONG.	74
SetArrayStringValue()	C	Sets the value of the specified container leaf element in the C-language. The leaf element is part of an array and of type STRING.	74
SetBinaryValue()	N	Sets the value of the specified container leaf element in the C-language. The leaf element is of type BINARY.	73
SetBuffer()	N	Sets the value of the specified container leaf element in Java. The leaf element is of type BINARY.	76

Accessor methods	Type	Description	Page
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.	76
SetDouble()	F	Sets the value of the specified container leaf element in Java. The leaf element is of type FLOAT.	76
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.	76
SetFloatValue()	F	Sets the value of the specified container leaf element in the C-language. The leaf element is of type FLOAT.	74
SetLong()	I	Sets the value of the specified container leaf element in Java. The leaf element is of type LONG.	76
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.	76
SetLongValue()	I	Sets the value of the specified container leaf element in the C-language. The leaf element is of type LONG.	74
SetString()	N	Sets the value of the specified container leaf element in Java. The leaf element is of type STRING.	76
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.	76
SetStringValue()	C	Sets the value of the specified container leaf element in the C-language. The leaf element is of type STRING.	74
SetValue()	N/F/C/I	Sets the value of the specified container leaf element in the C++ language.	75

Result

A result object represents the result of a API call call in the C++ and C-language.

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
destructor	Destructs the C++ representation of the result object.	99

Refer to "Accessor/mutator API calls" on page 106 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 an NLS regarding formatted message.	137
ObjectOfCurrentThread()	P	Returns the result object associated with the thread from where this API call is called.	139
Origin()	C	Returns the origin of the result, that is, file, line, function.	137
Parameters()	M	Returns the parameters of the result; these are already incorporated in the message text.	138
Rc()	I	Returns the return code remembered in the result object.	136

Service

A service object represents common aspects of MQ Workflow service objects. **All API calls of a service are also applicable to execution services.**

Refer to "Accessor/mutator API calls" on page 106 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.

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
IsLoggedOn()	B	Indicates whether a user logged on via this service object. This API call tells you the logon status known by the client. When issuing an action call, the session may, however, be not found because it expired or because you reconstructed a session from a different system group.	109
SetTimeout()	I	Sets the time the client will wait for a server to answer. The time is to be specified in milliseconds.	142
SystemGroupName()	C	Returns the name of the system group where the server resides.	137
SystemName()	C	Returns the name of the system where the server resides.	137
Timeout()	I	Returns the time the client will wait for a server to answer.	136
UserID()	C	Returns the user identification of the logged-on user.	137

Refer to “Action API calls” on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
Refresh()	Refreshes information from the server, especially the logged-on status.	613
SetPassword()	Sets the password of the logged-on user.	615
UserSettings()	Retrieves the user settings of the logged-on user.	617

StringArray

A string array represents a list of strings in ActiveX.

Refer to “ActiveX arrays” on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	40
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

Accessor methods	Description	Page
RemoveAll()	Removes all elements.	42
RemoveAt()	Removes the element at the indicated position.	42
SetAt()	Sets the element at the indicated position.	42

StringVector

In the C-language, a string vector serves to represents 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-language vectors" on page 35 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.
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.

Accessor methods	Description
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 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a symbol layout object.	94
Copy()	Allocates and initializes the storage for a symbol layout object by copying.	99
Deallocate()	Deallocates the storage for a symbol layout object.	99
destructor()	Destructs a symbol layout object.	99
Equal()	Compares two symbol layout objects on the basis of their contents.	97
IsEmpty()	Indicates whether no symbol layout information is available.	101
operator=()	Assigns a symbol layout object to this one.	97
operator==()	Compares two symbol layout objects on the basis of their contents.	97

Refer to "Accessor/mutator API calls" on page 106 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.	136
XPositionOfName()	I	Returns the x-coordinate of the name associated to the icon.	136

Accessor methods	Type	Description	Page
YPosition()	I	Returns the y-coordinate of the named icon.	136
YPositionOfName()	I	Returns the y-coordinate of the name associated to the icon.	136

Workitem

A work item represents an activity instance assigned to a user in order to be worked on. **All API calls of an Item are also applicable to work items.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a work item object.	94
Copy()	Allocates and initializes the storage for a work item object by copying.	99
Deallocate()	Deallocates the storage for a work item object.	99
destructor()	Destructs a work item object.	99
Kind()	In the C++ language, states that the object is a work item.	101, 131
operator=()	Assigns a work item to this one.	97
operator==()	Compares two work items.	97

Refer to "Accessor/mutator API calls" on page 106 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. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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.	101, 118
ErrorReason()	S/O	Returns an error object describing the reason why the associated activity instance is in state InError.	139
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	142
ExitCondition()	S/C	Returns the exit condition of the work item.	137
ExpirationTime()	S/D	Returns the expiration time of the work item.	110
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	142
FirstNotificationTime()	S/D	Returns the time the first notification for the work item is to occur or has occurred.	110
FirstNotificationTimeIsNull()	S/B	Indicates whether a first notification time is set.	142
Implementation()	P/C	Returns the name of the implementing program of the associated activity instance.	137
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	142
ManualExitMode()	S/B	Returns whether the exit mode of the work item is manual.	109
ManualStartMode()	S/B	Returns whether the start mode of the work item is manual.	109
PersistentOidOfActivityInstance()	P/C	Returns the object ID of the associated activity instance.	137
Priority()	P/I	Returns the priority of the work item.	136
SecondNotificationTime()	S/D	Returns the time the second notification for the work item is to occur or has occurred.	110
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	142
Staff()	S/M	Returns all persons owning a work item for the associated activity instance.	138

Accessor methods	Set/ Type	Description	Page
StartCondition()	S/C	Returns the start condition of the work item.	137
State	P/E	Returns the state of the work item.	111, 128
StateOfNotification()	S/E	Returns the notification state of the work item.	111, 114
SupportTools()	P/M	Returns the support tools associated with the work item.	138
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	142

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
ActivityInstance()	Retrieves the associated activity instance.	624
CancelCheckOut()	Cancels the check out of the work item.	627
CheckIn()	Checks in the work item.	629
CheckOut()	Checks out the work item.	632
Finish()	Finishes a manual exit work item.	638
ForceFinish()	Force finishes the work item.	640
ForceFinishWithContainer()	Force finishes the work item and passes a container.	341
ForceFinish2()	In Java, force finishes the work item and passes a container.	341
ForceRestart()	Force restarts the work item.	643
ForceRestartWithContainer()	Force restarts the work item and passes a container.	344
ForceRestart2()	In Java, force restarts the work item and passes a container.	344
InContainer()	Retrieves the input container of the work item.	646
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	492
OutContainer()	Retrieves the output container of the work item.	648
Restart()	Restarts the work item.	650
Start()	Starts the work item.	652
StartTool()	Starts the specified support tool.	655
Terminate()	Terminates the work item.	657

WorkitemArray

A work item array represents the result of a query for work items in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

WorkitemVector

A workitem vector represents the result of a query for work items in the C-language.

Refer to "C-language vectors" on page 35 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates the storage for a workitem vector object.
FirstElement()	Returns the first element of the workitem vector.
NextElement()	Returns the next element of the workitem vector.
Size()	Returns the number of elements in the workitem vector.

Worklist

A worklist represents a group of items. **All API calls of a persistent list are also applicable to worklists.**

Refer to "Basic API calls" on page 93 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a worklist object.	94
Copy()	Allocates and initializes the storage for a worklist object by copying.	99
Deallocate()	Deallocates the storage for a worklist object.	99
destructor()	Destructs a worklist object.	99
Equal()	Compares two worklists.	97
operator=()	Assigns a worklist to another one.	97
operator==()	Compares two worklists.	97

Refer to "Accessor/mutator API calls" on page 106 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.	109

Refer to "Action API calls" on page 150 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications qualifying via the worklist.	661
QueryItems()	Retrieves all items qualifying via the worklist.	665
QueryProcessInstanceNotifications()	Retrieves the process instance notifications qualifying via the worklist.	668
QueryWorkitems()	Retrieves the work items qualifying via the worklist.	671

WorklistArray

A worklist array represents the result of a query for worklists in ActiveX.

Refer to "ActiveX arrays" on page 40 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	41
GetSize()	Returns the number of elements in the array.	41

Events	Description	Page
NewWorklist()	Adds a new worklist to the array.	43
WorklistRemove()	Removes the specified worklist from the array.	43

WorklistVector

A worklist vector represents the result of a query for worklists in the C-language.

Refer to "C-language vectors" on page 35 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.

Part 7. API action and activity implementation calls

The following chapters describe the MQ Workflow application programming interfaces for action or activity implementation API calls in alphabetical order.

Each entry contains a functional description of the API 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 MQ Workflow server a session must have been established with.

API interface declaration
Shows the required file declarations.

ActiveX signature
Shows the ActiveX syntax of the API call.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

C-language signature
Shows the C-language syntax of the API call.

C++ language signature
Shows the C++ language syntax of the API call.

Java signature Shows the Java syntax of the API call.

Parameters Describes each of the parameters together with an indicator whether the parameter is an input or output parameter.

Return type Describes the value returned by the call.

Return codes/ FmcException
Lists all possible return codes which may be raised by this call.

Examples Points to an example of the call.

Chapter 35. Activity instance actions

An FmcjActivityInstance or 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 255 for a complete list of API calls.

ForceFinish()

This API call ends the execution of the specified activity instance because it is known to have completed (action call).

An activity instance implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. An activity instance 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*.

Optionally, an output container can be specified to denote the result of processing. If none is specified, the output container available at the execution server is taken. For example, the output container defined with initial values.

The activity instance and, if it exists, the single non-disabled work item are then put into the *ForceFinished* state. The starter is set to the logged-on user. The exit condition is considered to be true and navigation proceeds.

Depending on the "delete finished items" option, associated work items are deleted.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long ForceFinish()

long ForceFinishWithContainer( Container * outputContainer )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceForceFinish( FmcjActivityInstanceHandle hdlInstance,
                                 FmcjContainerHandle        outputContainer)
```

C++ language signatures

```
APIRET ForceFinish()

APIRET ForceFinish( FmcjContainer const & outputContainer )
```

Java signature

```
public abstract
void   forceFinish() throws FmcException

public abstract
void   forceFinish2( Container outputContainer ) throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

outputContainer
Input. The output container to become the result of the activity instance execution. A 0 handle can be passed in the C-language.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.
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 does 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_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.
FMC_ERROR_WRONG_STATE(120)
The activity instance 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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceRestart()

This API call forces MQ Workflow to enable the restart of the specified activity instance (action call).

An activity instance implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. An activity instance implemented by a process must be in the states *Ready, Executed, InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

Optionally, an input container can be specified to denote the input to be used when the activity instance or its associated work item is (re)started. If none is specified, the input container available at the execution server is taken.

The activity instance and the logged-on user's work item are then reset into the *Ready* state. If there is no work item for the logged-on user, it is created. All other work items associated with the activity instance are set into the *Disabled* state. Note that an automatic activity instance must now be started manually.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long ForceRestart()

long ForceRestartWithContainer( Container * inputContainer )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceForceRestart(
       FmcjActivityInstanceHandle hdlInstance,
       FmcjContainerHandle        inputContainer   )
```

C++ language signature

```
APIRET ForceRestart()

APIRET ForceRestart( FmcjContainer const & inputContainer )
```

Java signature

```
public abstract
void   forceRestart() throws FmcException

public abstract
void   forceRestart2( Container inputContainer ) throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

inputContainer
Input. The input container to be used for restarting the activity instance. A 0 handle can be passed in the C-language.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.

FMC_ERROR_WRONG_STATE(120)
The activity instance 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the activity instance from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long InContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceInContainer( FmcjActivityInstanceHandle    hdlInstance
                                 FmcjReadOnlyContainerHandle * input     )
```

C++ language signature

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java signature

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance 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.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()/ObtainInstanceMonitor

This API call retrieves a monitor for the process instance the activity instance is part of from the MQ Workflow execution server (action call).

When the deep option is specified, all activity instances of type Block are resolved, that is, their monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the 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 instance monitor handle already points to some object.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
InstanceMonitor*
ObtainInstanceMonitor( boolean deep, long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceObtainProcessMonitor(
       FmcjActivityInstanceHandle       hdlInstance,
       bool                             deep,
       FmcjInstanceMonitorHandle *      monitor)
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false ) const
```

Java signature

```
public abstract
InstanceMonitor obtainProcessMonitor( boolean deep )
throws FmcException
```

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 currently ignored.
hdlInstance	Input. The activity instance whose instance monitor for the containing process instance is to be retrieved.
monitor	Input/Output. The address of the handle to the instance monitor respectively the instance monitor object to be set.
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.

InstanceMonitor*/InstanceMonitor
A pointer to the instance monitor respectively the instance monitor for 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the activity instance from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long OutContainer( Container * output )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceOutContainer( FmcjActivityInstanceHandle    hdlInstance,
                                  FmcjReadOnlyContainerHandle * output     )
```

C++ language signature

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java signature

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

output Input/Output. The output container.

Return type

long/ APIRET The return code of calling this method - see below.

ReadOnlyContainer
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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the activity instance from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceRefresh( FmcjActivityInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance object to be refreshed.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SubProcessInstance()

This API call retrieves the process instance which is implementing the activity instance from the MQ 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, it is not checked whether the process instance handle already points to some object.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
ProcessInstance* SubProcessInstance( long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceSubProcessInstance(
       FmcjActivityInstanceHandle   hdlInstance,
       FmcjProcessInstanceHandle *  instance )
```

C++ language signature

```
APIRET SubProcessInstance( FmcjProcessInstance & instance ) const
```

Java signature

```
public abstract
ProcessInstance subProcessInstance() throws FmcException
```

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 respectively 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)

A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)

Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)

The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This API call terminates an activity instance implemented by a program or process (action call).

If the activity instance 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*, *Suspended*, or *Terminating*. If the activity instance 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*.

When the activity instance is implemented by a program and processed under the control of a program execution agent or user-defined program execution server, a message is sent to inform about the termination request. The program execution agent tries to kill fenced activity implementations.

An activity instance implemented by a process is terminated together with all its non-autonomous subprocesses with respect to control autonomy.

The activity instance is then put into the *Terminating* or *Terminated* state.

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. Navigation can be explicitly continued by a user with process administration rights, that is, ForceFinish() or ForceRestart() repair actions can be called.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process administration authorization
- Be the starter of the associated work item
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX signature

```
long Terminate()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceTerminate( FmcjActivityInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

Parameters

hdlInstance Input. The handle of the activity instance 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.

FMC_ERROR_WRONG_STATE(120)
The activity instance 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 36. Activity instance notification actions

An FmcjActivityInstanceNotification or 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 the C++ language, FmcjActivityInstanceNotification is thus a subclass of the FmcjItem class and inherits all properties and methods. In the Java language, ActivityInstanceNotification is thus a subclass of the Item class and inherits all properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all functions are explicitly defined on ActivityInstanceNotification. Note, however, that they are described as Item actions.

An activity instance notification is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on an activity instance notification. See "ActivityInstanceNotification" on page 260 for a complete list of API calls.

ActivityInstance()

This API call retrieves the activity instance the activity instance notification is associated to from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

In C++, when the activity 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 activity instance handle already points to some object.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstanceNotification

ActiveX signature

```
ActivityInstance* ActivityInstance( long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationActivityInstance(
       FmcjActivityInstanceNotificationHandle hdlItem,
       FmcjActivityInstanceHandle *           instance )
```

C++ language signature

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java signature

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

Parameters

hdlItem Input. The handle of the activity instance notification object to be queried.

instance Input/Output. The activity 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 of calling this API call - see return codes below.

ActivityInstance*/ ActivityInstance

A pointer to the activity instance or the activity instance the activity instance notification is associated to.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist. Or, the transient activity instance notification object recreated from its OID is not an activity instance notification; it is a process 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)

A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)

Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 to the activity instance the notification has been created for. It is then started on the program execution agent associated to the logged-on user.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be the activity instance notification owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstanceNotification

ActiveX signature

```
long StartTool( BSTR toolName )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationStartTool(
                    FmcjActivityInstanceNotificationHandle hdlItem,
                    char const *                           toolName )
```

C++ language signature

```
APIRET StartTool( string const & toolName ) const
```

Java signature

```
public abstract
void   startTool( String toolName ) throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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_WRONG_KIND(501)
The transient activity instance notification object recreated from its OID is not an activity instance notification; it is a process instance notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 37. Container activity implementation API calls

An FmcjContainer or Container object represents a data container of a process template, process instance, work item, activity implementation, or support tool. A container can be a read-only input container or a read/write input or output container.

The API calls defined on 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.

An FmcjContainer or Container object represents the common aspects of read-only or read/write containers. In the C++ language, FmcjContainer is thus the superclass of the FmcjReadOnlyContainer and FmcjReadWriteContainer classes and provides for all common properties and methods. In the Java language, Container is thus a superclass of the ReadOnlyContainer and ReadWriteContainer classes and provides for all common properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported. All methods are available on the Container class.

The following sections describe the activity implementation functions which are used for communication between an activity implementation or support tool and a program execution agent. See “Container” on page 266 for a complete list of API calls on containers.

InContainer()

This API call retrieves the input container from the MQ Workflow program execution agent (activity implementation call).

It can be used from within an activity implementation or support tool.

Usage note

- See “Activity implementation API calls” on page 151 for general information.

Authorization

Be an activity implementation or support tool

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX	IBM MQSeries Workflow 3.1
C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx resepctively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long InContainer()
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerInContainer(
        FmcjReadOnlyContainerHandle * input )
```

C++ language signature

```
static APIRET InContainer( FmcjReadOnlyContainer & input )
```

Java signature

```
public abstract
       ReadOnlyContainer ExecutionAgent.inContainer()
throws FmcException
```

Parameters

input Input/Output. The address of the input container handle respectively the input container of the activity implementation or support tool to be set.

Return type

long/ APIRET

The return code of calling this API call - see return codes below.

ReadOnlyContainer

The input container of the activity implementation or support tool.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 called from within an activity implementation or support tool or the program execution agent is not active.

FMC_ERROR_COMMUNICATION(13)
The specified program execution agent cannot be reached.

FMC_ERROR_INTERNAL(100)
An MQ 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-language example see "Programming an executable (C-language)" on page 801
- For a C++ example see "Programming an executable (C++)" on page 802

OutContainer()

This API call retrieves the output container from the MQ Workflow program execution agent (activity implementation call).

It can be used from within an activity implementation.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Be an activity implementation

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long OutContainer()
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerOutContainer(
        FmcjReadWriteContainerHandle * output )
```

C++ language signature

```
static APIRET OutContainer( FmcjReadWriteContainer & output )
```

Java signature

```
public abstract
       ReadWriteContainer ExecutionAgent.outContainer()
throws FmcException
```

Parameters

output	Input/Output. The address of the output container handle respectively the output container of the activity implementation to be set.

Return type

long/ APIRET	The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 called from within an activity implementation or the program execution agent is not active.

FMC_ERROR_TOOL_FUNCTION(128)
A support tool cannot access an output container.

FMC_ERROR_COMMUNICATION(13)
The specified program execution agent cannot be reached.

FMC_ERROR_INTERNAL(100)
An MQ 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-language example see “Programming an executable (C-language)” on page 801
- For a C++ example see “Programming an executable (C++)” on page 802

RemoteInContainer()

This API call retrieves the input container from the MQ Workflow program execution agent (activity implementation call).

It can be used from within a program started by an activity implementation or support tool.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Valid program identification

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long RemoteInContainer( BSTR programID )
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerRemoteInContainer(
        char const *                  programID,
        FmcjReadOnlyContainerHandle * input )
```

C++ language signature

```
static APIRET RemoteInContainer(
       string const &               programID,
       FmcjReadOnlyContainer &      input )
```

Java signature

```
public abstract
   ReadOnlyContainer ExecutionAgent.remoteInContainer( String programID )
throws FmcException
```

Parameters

input Input/Output. The address of the input container handle respectively the input container of the activity implementation or support tool to be set.

programID Input. The program identification by which the activity implementation or support tool is known to the program execution agent.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

ReadOnlyContainer

The input container of the activity implementation or support tool.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 called from within an activity implementation or the program execution agent is not active.

FMC_ERROR_COMMUNICATION(13)

The specified program execution agent cannot be reached.

FMC_ERROR_INTERNAL(100)

An MQ 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 MQ Workflow program execution agent (activity implementation call).

It can be used from within a program started by an activity implementation.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Valid program identification

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long RemoteOutContainer( BSTR programID )
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerRemoteOutContainer(
        char const *                  programID,
        FmcjReadWriteContainerHandle * output )
```

C++ language signature

```
static APIRET RemoteOutContainer(
       string const &                 programID,
       FmcjReadWriteContainer &       output )
```

Java signature

```
public abstract
   ReadWriteContainer ExecutionAgent.remoteOutContainer( String programID )
throws FmcException
```

Parameters

output Input/Output. The address of the output container handle respectively 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 agent.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 called from within an activity implementation or the program execution agent is not active.

FMC_ERROR_TOOL_FUNCTION(128)

A support tool cannot access an output container.

FMC_ERROR_COMMUNICATION(13)

The specified program execution agent cannot be reached.

FMC_ERROR_INTERNAL(100)
An MQ 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 MQ Workflow program execution agent (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 MQ Workflow execution server until the activity implementation ends. It is kept transiently by the program execution agent.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Be an activity implementation

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long SetOutContainer()
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerSetOutContainer(
        FmcjReadWriteContainerHandle const output )
```

C++ language signature

```
static APIRET SetOutContainer( FmcjReadWriteContainer const & output )
```

Java signature

```
public abstract
  void ExecutionAgent.setOutContainer( ReadWriteContainer output )
throws FmcException
```

Parameters

output Input. The output container handle respectively the output container of the activity implementation to be passed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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_CONTAINER(509)
The container passed is not a valid output container for the activity implementation; wrong schema or version.

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 called from within an activity implementation or the program execution agent is not active.
FMC_ERROR_TOOL_FUNCTION(128)
A support tool cannot set the output container.
FMC_ERROR_COMMUNICATION(13)
The specified program execution agent cannot be reached.
FMC_ERROR_INTERNAL(100)
An MQ 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-language example see "Programming an executable (C-language)" on page 801
- For a C++ example see "Programming an executable (C++)" on page 802

SetRemoteOutContainer()

This API call returns the output container to the MQ Workflow program execution agent (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 MQ Workflow execution server until the activity implementation ends. It is kept transiently by the program execution agent.

Usage note
- See "Activity implementation API calls" on page 151 for general information.

Authorization

Valid program identification

Required connection

None but active MQ Workflow program execution agent.

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjccon.h respectively fmcjcrun.h

C++ fmcjpcon.hxx respectively fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionAgent

ActiveX signature

```
long SetRemoteOutContainer( BSTR programID )
```

C-language signature

```
APIRET  FMC_APIENTRY FmcjContainerSetRemoteOutContainer(
        char const *                      programID,
        FmcjReadWriteContainerHandle const output )
```

C++ language signature

```
static APIRET SetRemoteOutContainer(
       string const &                  programID,
       FmcjReadWriteContainer const &  output )
```

Java signature

```
public abstract
void   ExecutionAgent.setRemoteOutContainer( String             programID,
                                             ReadWriteContainer output   )
throws FmcException
```

Parameters

output Input. The output container handle respectively the output container of the activity implementation to be passed.

programID Input. The program identification by which the activity implementation is known to the program execution agent.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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_CONTAINER(509)
The container passed is not a valid output container for the activity implementation; wrong schema or version.

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 called from within an activity implementation or the program execution agent is not active.

FMC_ERROR_TOOL_FUNCTION(128)
A support tool cannot set the output container.

FMC_ERROR_COMMUNICATION(13)
The specified program execution agent cannot be reached.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

Chapter 38. Execution service actions

An FmcjExecutionService or ExecutionService object represents a session between a user and an MQ Workflow execution server so that Runtime services may be asked for.

The execution service object essentially provides for the basic API calls to set up a communication path to the specified MQ 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 MQ 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 the C++ language, 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 the C-language, 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. In ActiveX, inheritance is not supported so that all functions are explicitly defined on ExecutionService. Note, however, that they are described as Service actions.

The following sections describe the actions which can be applied on an execution service. See “ExecutionService” on page 281 for a complete list of API calls.

CreateProcessInstanceList()

This API call creates a process instance list on the MQ 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.

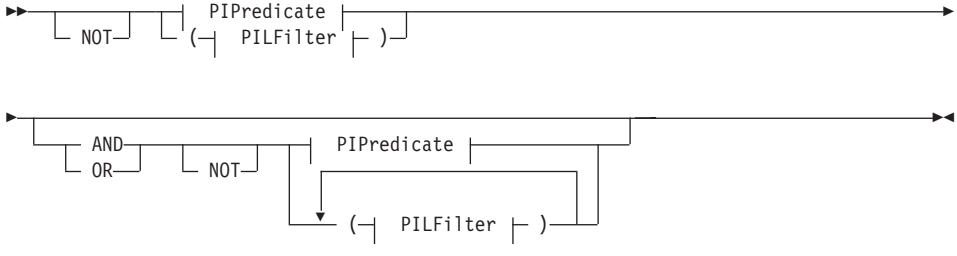
The following rules apply for specifying a description:
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

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 xiii).

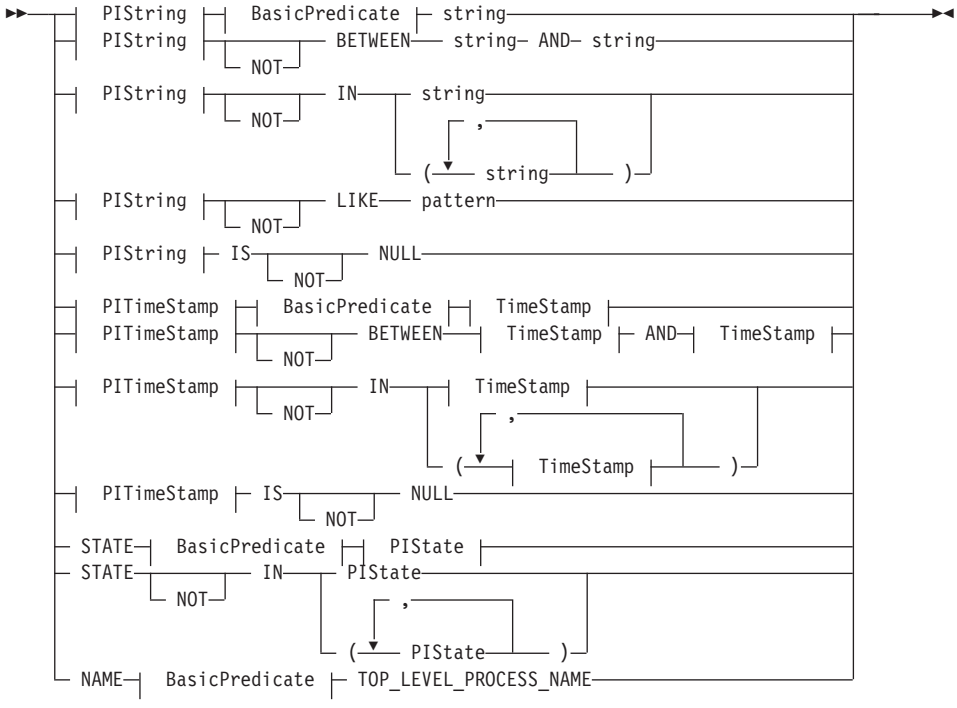
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

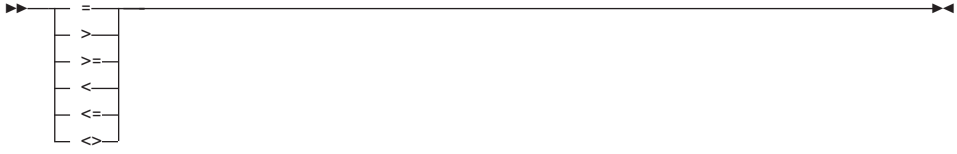
PILFilter



PIPredicate



BasicPredicate

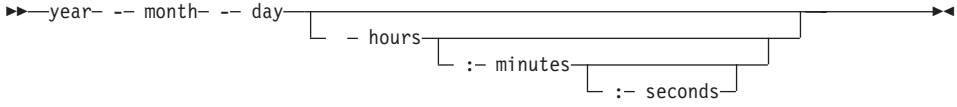


PIState

PIString

PITimeStamp

TimeStamp

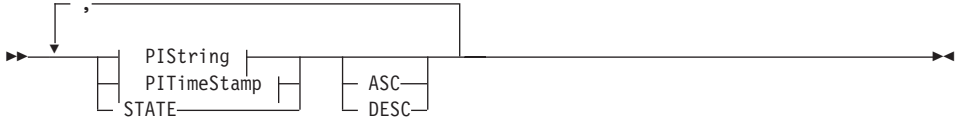


A process instance list sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PIState diagram.

PILOrderBy



Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long CreateProcessInstanceList(
       BSTR                              name,
       long                              type,
       BSTR                              owner,
       boolean                           ownerIsNull,
       BSTR                              description,
       boolean                           descriptionIsNull,
       BSTR                              filter,
       boolean                           filterIsNull,
       BSTR                              sortCriteria,
       boolean                           sortCriteriaIsNull,
       long                              threshold,
       boolean                           thresholdIsNull )
```

C-language signature

```
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++ language signature

```
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 signature

```
public abstract
ProcessInstanceList createProcessInstanceList(
       String                          name,
       TypeOfList                      type,
       String                          owner,
       String                          description,
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

Parameters

description Input. A user-defined description of the process instance list.

descriptionIsNull
Input. Indicates whether a description is provided for the list.

filter Input. The filter criteria which characterize the process instances to be contained in the process instance list.

filterIsNull Input. Indicates whether a filter is provided for the list.

name Input. A user-defined name for the process instance list.

newList Input/Output. The newly created process instance list.

owner Input. The owner of the list when the type is private. Ignored for public lists.

ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.

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 instances in the process instance list.

sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.

threshold Input. The threshold which defines the maximum number of process instances in the process instance list to be passed to the client.

thresholdIsNull
Input. Indicates whether a threshold is provided for the list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code of calling 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.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Create a process instance list (ActiveX)" on page 759.
- For a C-language example see "Create a process instance list (C-language)" on page 760.
- For a C++ example see "Create a process instance list (C++)" on page 762.
- For a Java example see "Create a process instance list (Java)" on page 764.

CreateProcessTemplateList()

This API call creates a process template list on the MQ 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:

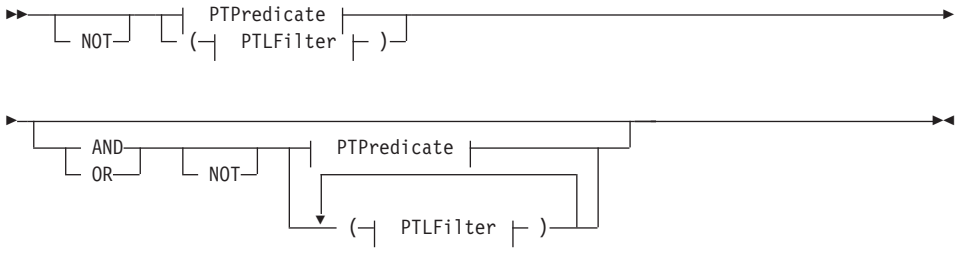
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

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 xiii).

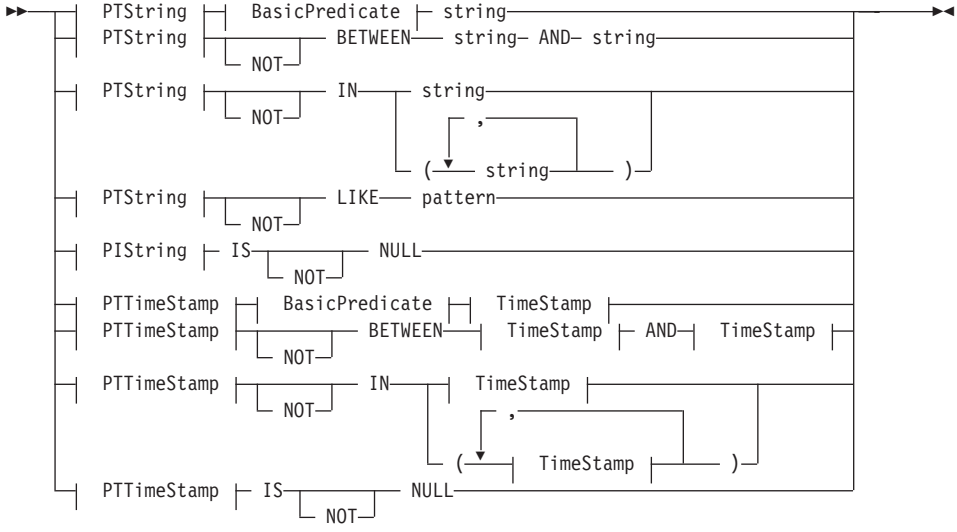
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PTLFilter



PTPredicate

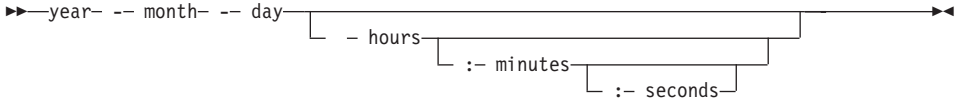


BasicPredicate

PTString

PTTimeStamp

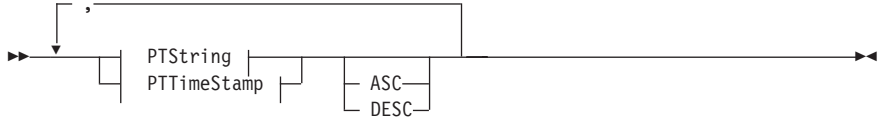
TimeStamp



A process template list sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTLOrderBy



Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long CreateProcessTemplateList(
       BSTR                              name,
       long                              type,
       BSTR                              owner,
       boolean                           ownerIsNull,
       BSTR                              description,
       boolean                           descriptionIsNull,
       BSTR                              filter,
       boolean                           filterIsNull,
       BSTR                              sortCriteria,
       boolean                           sortCriteriaIsNull,
       long                              threshold,
       boolean                           thresholdIsNull )
```

C-language signature

```
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 )
```

C++ language signature

```
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 signature

```
public abstract
ProcessTemplateList createProcessTemplateList(
       String                          name,
       TypeOfList                      type,
       String                          owner,
       String                          description,
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

Parameters

description Input. A user-defined description of the process template list.

descriptionIsNull
Input. Indicates whether a description is provided for the list.

filter Input. The filter criteria which characterize the process templates in the process template list.

filterIsNull Input. Indicates whether a filter is provided for the list.

name Input. A user-defined name for the process template list.

newList Input/Output. The newly created process template list.

owner Input. The owner of the list when the type is private. Ignored for public lists.

ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.

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 templates in the process template list.

sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.

threshold Input. The threshold which defines the maximum number of process templates in the process template list.

thresholdIsNull
Input. Indicates whether a threshold is provided for the list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Create a process instance list (ActiveX)" on page 759.
- For a C-language example see "Create a process instance list (C-language)" on page 760.
- For a C++ example see "Create a process instance list (C++)" on page 762.
- For a Java example see "Create a process instance list (Java)" on page 764.

CreateWorklist()

This API call creates a worklist on the MQ 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:

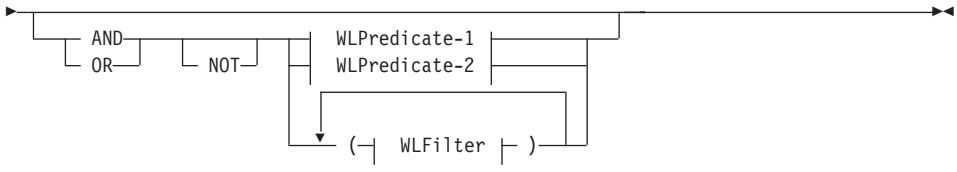
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

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 xiii).

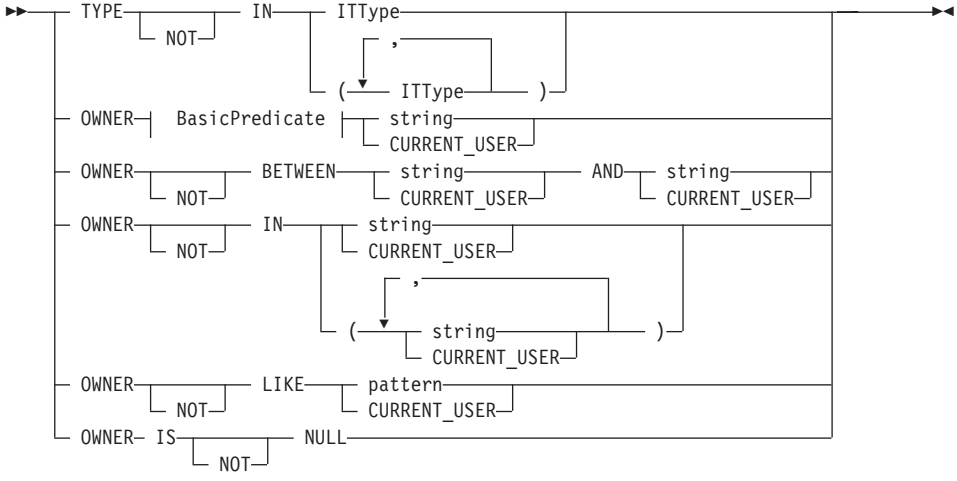
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

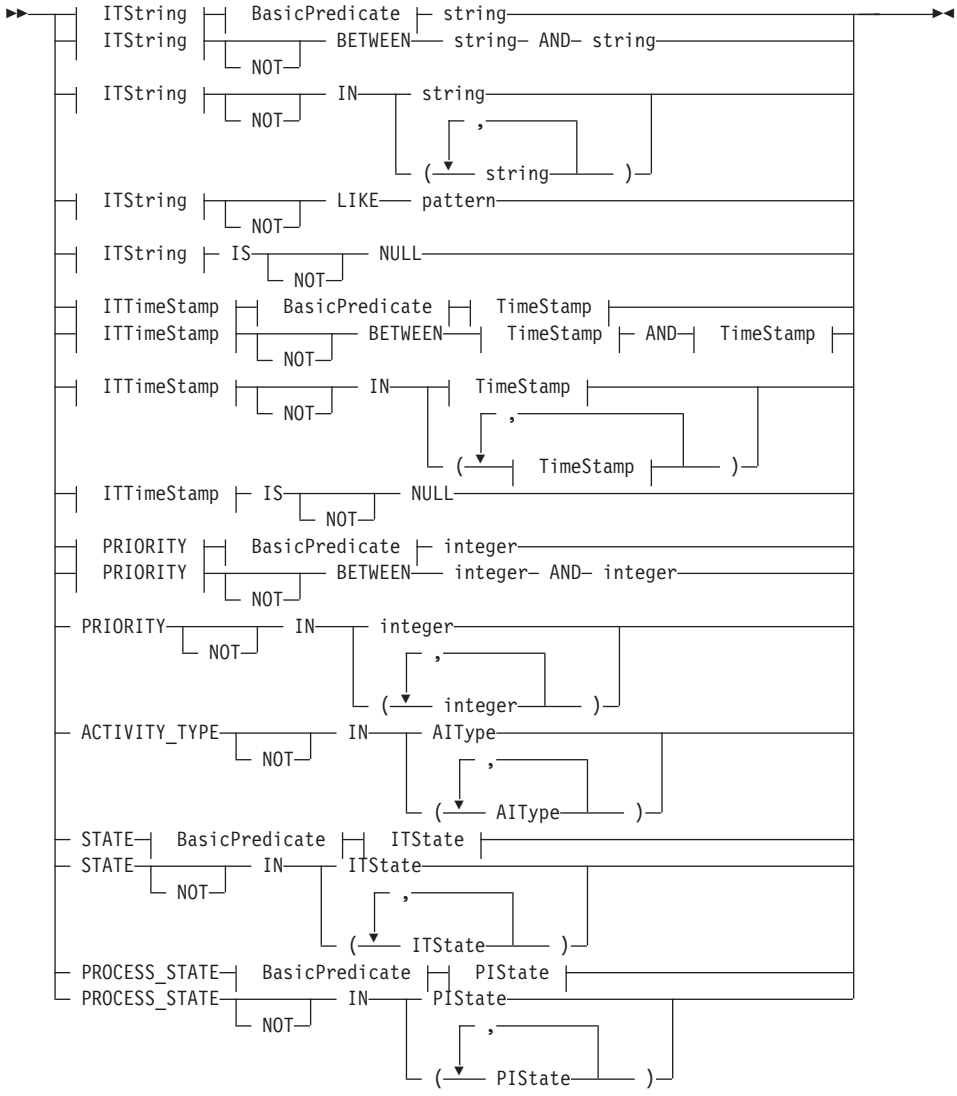
WLFilter



WLPredicate-1



WLPredicate-2



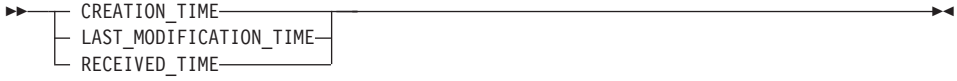
AIType

BasicPredicate

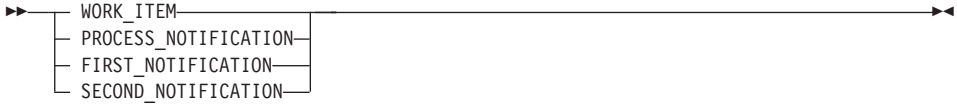
ITState

ITString

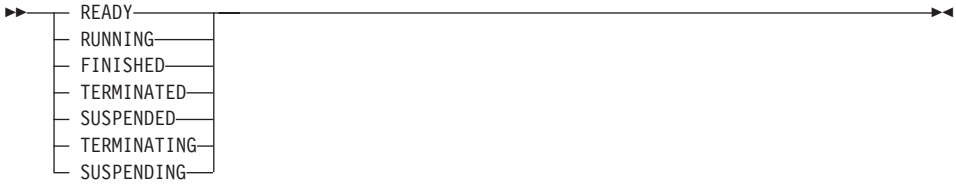
ITTimeStamp



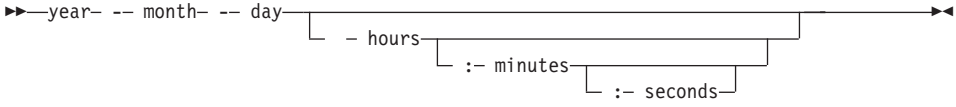
ITType



PIState



TimeStamp



A worklist sort criterion is specified as a character string.

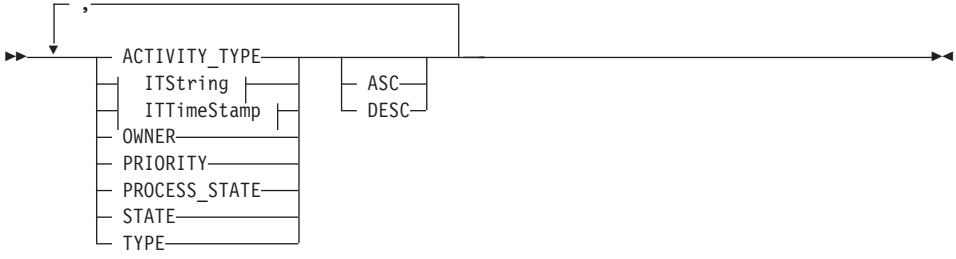
Note: The default sort order is ascending.

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 respectively the PIState diagram.

WLOrderBy



Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long CreateWorklist(
       BSTR                              name,
       long                              type,
       BSTR                              owner,
       boolean                           ownerIsNull,
       BSTR                              description,
       boolean                           descriptionIsNull,
       BSTR                              filter,
       boolean                           filterIsNull,
       BSTR                              sortCriteria,
       boolean                           sortCriteriaIsNull,
       long                              threshold,
       boolean                           thresholdIsNull )
```

C-language signature

```
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++ language signature

```
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 signature

```
public abstract
WorkList createWorkList(
       String                          name,
       TypeOfList                      type,
       String                          owner,
       String                          description,
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

Parameters

description Input. A user-defined description of the worklist.

descriptionIsNull
Input. Indicates whether a description is provided for the list.

filter Input. The filter criteria which characterize the items in the worklist.

filterIsNull Input. Indicates whether a filter is provided for the list.

name Input. A user-defined name for the worklist.

newList Input/Output. The newly created worklist.

owner Input. The owner of the list when the type is private. Ignored for public lists.

ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.

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 in the worklist.

sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.

threshold Input. The threshold which defines the maximum number of items in the worklist.

thresholdIsNull
Input. Indicates whether a threshold is provided for the list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Create a process instance list (ActiveX)" on page 759.
- For a C-language example see "Create a process instance list (C-language)" on page 760.
- For a C++ example see "Create a process instance list (C++)" on page 762.
- For a Java example see "Create a process instance list (Java)" on page 764.

Logoff()

This API call allows the application to finish the specified user session with an MQ Workflow execution server (action call).

When logoff has been successfully executed, no further client/server calls are accepted using this execution service object.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long Logoff()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogoff(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET Logoff()
```

Java signature

```
public abstract
void   logoff() throws FmcException
```

Parameters

service Input. A handle to the service object representing the session with the execution server.

Return type
long/ APIRET

The return code of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Part 9. Examples and scenarios" on page 753.

Logon()

This API call allows an application to establish a user session with an MQ Workflow execution server (action call).

A successful Logon() is the prerequisite for using all other action and program execution management API calls of the MQ Workflow API.

You either log on by specifying a user ID and a password or you log on by specifying user credentials which are then verified by your authentication exit.

The user ID to log on with, respectively the user ID returned by your authentication exit, must be a registered MQ Workflow user.

When the execution server supports *unified logon* and when you log on with a user ID and password, 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 a *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.

Furthermore, the *present* mode indicates to MQ Workflow that the session can handle unsolicitedmessages pushed by the execution server - see "The push data access model" on page 20 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. Note that using a present here session mode also requests to shut down the program execution agent.

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:

ActiveX	SessionMode
C-language	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.

ActiveX	SessionMode_Default
C-language	Fmc_SM_Default
C++	FmcjService::Default respectively FmcjExecutionService::Default
JAVA	SessionMode.DEFAULT

Present Indicates that you want to operate in a present session mode.

ActiveX	SessionMode_Present
C-language	Fmc_SM_Present
C++	FmcjService::Present respectively FmcjExecutionService::Present
JAVA	SessionMode.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.

ActiveX	SessionMode_PresentHere
C-language	Fmc_SM_PresentHere
C++	FmcjService::PresentHere respectively FmcjExecutionService::PresentHere
JAVA	SessionMode.PRESENT_HERE

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:

ActiveX	AbsenceIndicator
C-language	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.

	ActiveX	AbsenceIndicator_NotSet
	C-language	Fmc_SA_NotSet
	C++	FmcjService::NotSet respectively FmcjExecutionService::NotSet
	JAVA	AbsenceIndicator.NOT_SET
Reset	Indicates that your absence setting is to be reset; you are back.	
	ActiveX	AbsenceIndicator_Reset
	C-language	Fmc_SA_Reset
	C++	FmcjService::Reset respectively FmcjExecutionService::Reset
	JAVA	AbsenceIndicator.RESET
Leave	Indicates that your absence setting should stay as is; you are either absent or not.	
	ActiveX	AbsenceIndicator_Leave
	C-language	Fmc_SA_Leave
	C++	FmcjService::Leave respectively FmcjExecutionService::Leave
	JAVA	AbsenceIndicator.LEAVE

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be a registered MQ Workflow user

Required connection

None

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long Logon                ( BSTR userID,     BSTR password )

long LogonWithOptions     ( BSTR             userID,
                            BSTR             password,
                            SessionMode      sessionMode,
                            AbsenceIndicator absenceIndicator )

long LogonWithCredentials ( VARIANT *        userCredentials,
                            SessionMode      sessionMode,
                            AbsenceIndicator absenceIndicator
                            BSTR             userName )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogon (
       FmcjExecutionServiceHandle        service,
       char const *                      userID,
       char const *                      password,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator )

APIRET FMC_APIENTRY FmcjExecutionServiceLogonWithCredentials(
       FmcjExecutionServiceHandle        service,
       FmcjBinary const *                userCredentials,
       unsigned long                     userCredentialsLength,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator,
       char const *                      userName )
```

C++ language signature

```
APIRET Logon(
       string const &                    userID,
       string const &                    password,
       SessionMode                       sessionMode = Present,
       AbsenceIndicator                  absenceIndicator = NotSet )

APIRET Logon(
       FmcjBinary const *                userCredentials,
       unsigned long                     userCredentialsLength,
       SessionMode                       sessionMode = Present,
       AbsenceIndicator                  absenceIndicator = NotSet,
       string const *                    userName = 0 )
```

Java signature

```
public abstract
void   logon ( String userID,   String password )

public abstract
void   logon2( String           userID,
               String           password,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator ) throws FmcException

public abstract
void   logon3( Byte[]           userCredentials  ) throws FmcException

public abstract
void   logon4( Byte[]           userCredentials,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator,
               String           userName         ) throws FmcException
```

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.

userCredentials
Input. The user credentials to be passed to a user-provided authentication exit.

userCredentialsLength
Input. The length of the binary user credentials string.

userID Input. The user ID of the user on whose behalf a logon is to be made. Can be empty for unified logon.

userName Input. An optional user name to be passed to a user-provided authentication exit.

Return type

long/ APIRET
The return code of calling 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_AUTHENTICATION(513)
Your authentication exit rejected the logon request. See the first parameter in the result object for the error reason.

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_LOGON_DENIED(512)
The logon request has been denied by your authentication exit.

FMC_ERROR_NEWER_VERSION(505)
The version of the client is newer than the server version, that is, not supported.

FMC_ERROR_NOT_AUTHORIZED(119)
A user-provided authentication exit or entry points in the DLL to pass the credentials to are not found. Or, the authentication exit returns a recoverable error.

FMC_ERROR_PASSWORD(12)
Incorrect password.

FMC_ERROR_PROFILE(124)
The configuration profile or profile entries (system group, system) cannot be found.

FMC_ERROR_USERID_UNKNOWN(10)
No user ID registered with MQ 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Part 9. Examples and scenarios" on page 753.

Passthrough()

This API call can be used by an activity implementation to establish a user session with an MQ Workflow execution server from within this program (activity-implementation call).

When successfully executed, a session to the same execution server is set up from where 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.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Activity implementation started by MQ Workflow

Required connection

None but active MQ Workflow program execution agent

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long Passthrough()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServicePassthrough(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET Passthrough()
```

Java signature

```
public abstract
void passthrough()  throws FmcException
```

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 of calling 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 agent is not active.

FMC_ERROR_TOOL_FUNCTION(128)
Passthrough cannot be called from a support tool or from a program started by the program execution server.

FMC_ERROR_USERID_UNKNOWN(10)
The user who started the work item does no longer exist.
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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Programming an executable (C-language)" on page 801.
- For a C++ example see "Programming an executable (C++)" on page 802.

PEAShutDown()

This API call allows to shutdown the program execution agent associated to the logged on user (program execution management API call call).

The program execution agent is then shut down whether activity implementations are still running or not. Be careful to wait for any running activity implementations so that their result is correctly passed to the execution server.

Usage note
- See "Program execution management API calls" on page 155 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long PEAShutDown()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServicePEAShutDown(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET PEAShutDown()
```

Java signature

```
public abstract
void programExecutionAgentShutDown() throws FmcException
```

Parameters

service Input. The handle of the execution service object to identify the user and the program execution agent to be shutdown.

Return type

long/ APIRET The return code of calling 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_DOES_NOT_EXIST(118)
A program execution agent for the logged-on user is not running.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to issue this call from within an activity implementation or support tool.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

PEAStartUp()

This API call is used to start a program execution agent associated to the logged-on user (program-execution-management call).

The program execution agent is then started on the same node where this calling application runs. A single program execution agent per user is supported. All user's work, whether from this session or from others, is send to this program execution agent.

The program execution agent is **not** automatically shut down when the user session(s) ends; it must be possible for the program execution agent to wait for activity implementations to complete.

If you are told that the program execution agent already runs on a different node, you can issue a shutdown and try again. Be careful to wait for any running activity implementations.

Usage note

- See "Program execution management API calls" on page 155 for general information.

Authorization

Valid user session

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long PEAStartUp()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServicePEAStartUp(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET PEAStartUp()
```

ActiveX signature

```
public abstract
void programExecutionAgentStartUp() throws FmcException
```

Parameters

service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code of calling 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_STARTED(111)
A program execution agent for the logged-on user is already running.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to issue this call from within an activity implementation or support tool.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

QueryActivityInstanceNotifications()

This API call retrieves the activity instance notifications the user has access to from the MQ Workflow execution server (action call).

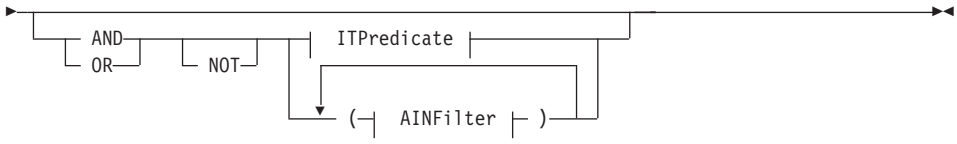
In C and C++, any activity instance notifications retrieved are appended to the supplied vector. If you want to read the current activity instance notifications only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language, respectively 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:

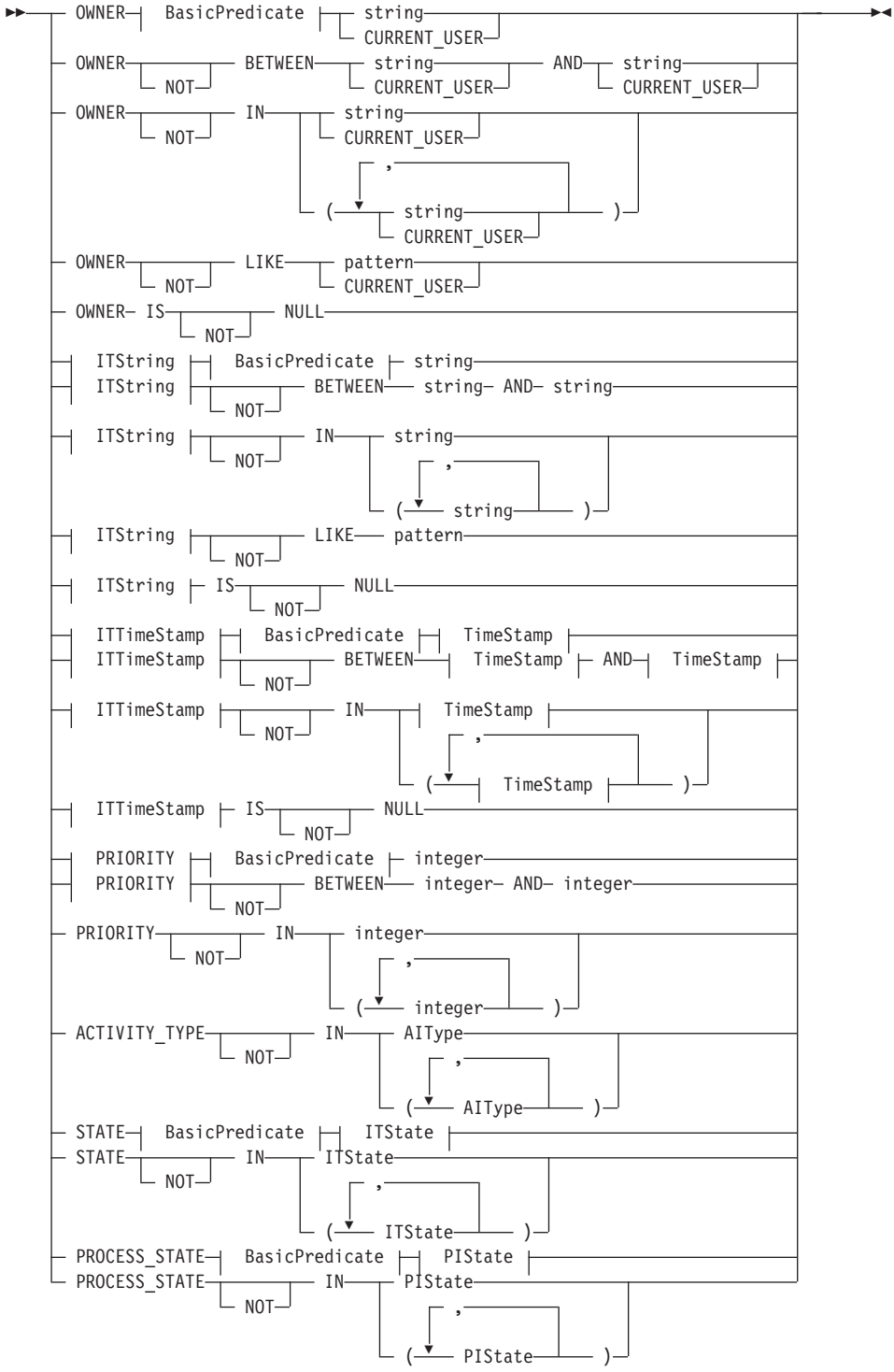
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

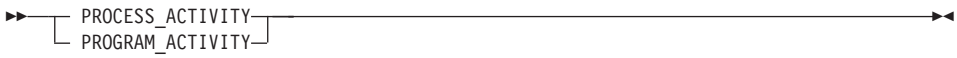
AINFilter



ITPredicate


OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
,
(
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
,
(
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
,
(
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
,
(
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
,
(
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
,
(
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
,
(
PIState
)

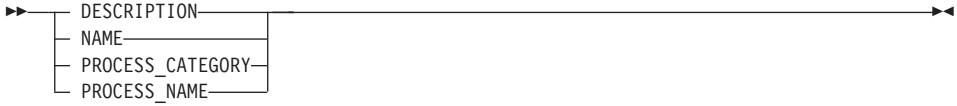
AIType



BasicPredicate

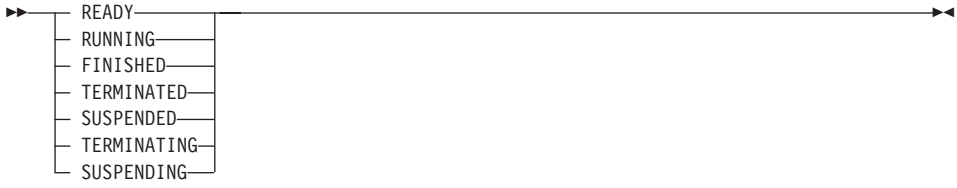
ITState

ITString

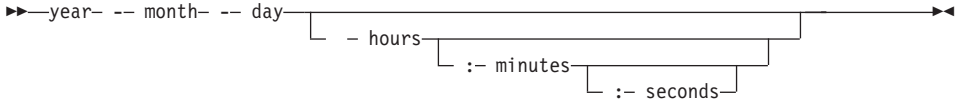


ITTimeStamp

PIState



TimeStamp



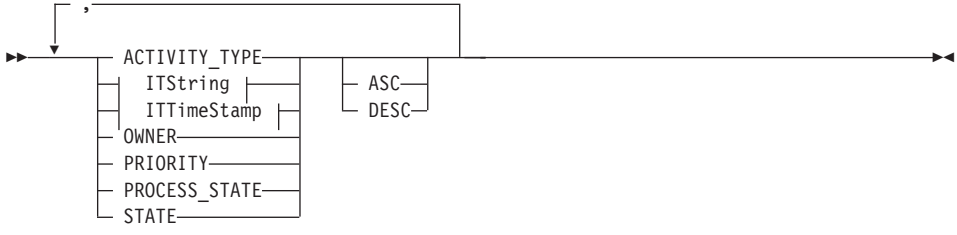
Activity instance notifications can be sorted. An activity instance notification sort criterion is specified as a character string.

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 respectively the PIState 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
- StartTime
- State
- SupportTools

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryActivityInstanceNotifications(
       FmcjExecutionServiceHandle                     service,
       char const *                                   filter,
       char const *                                   sortCriteria,
       unsigned long const *                          threshold,
       FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++ language signature

```
APIRET QueryActivityInstanceNotifications(
       string const *                              filter,
       string const *                              sortCriteria,
       unsigned long const *                       threshold,
       vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java signature

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications(
       String                                      filter,
       String                                      sortCriteria,
       Integer                                     threshold   )
throws FmcException
```

Parameters

filter Input. The filter criteria which characterize the activity instance notifications to be retrieved.

notifications Input/Output. The qualifying vector of activity 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 activity instance notifications found.

threshold Input. The threshold which defines the maximum number of activity instance notifications to be returned to the client.

Return type

APIRET The return code of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 783.
- For a C++ example see “Query process instances (C++)” on page 784.
- For a Java example see “Query process instances (Java)” on page 786.

QueryItems()

This API call retrieves the work items or notifications the user has access to from the MQ Workflow execution server (action call).

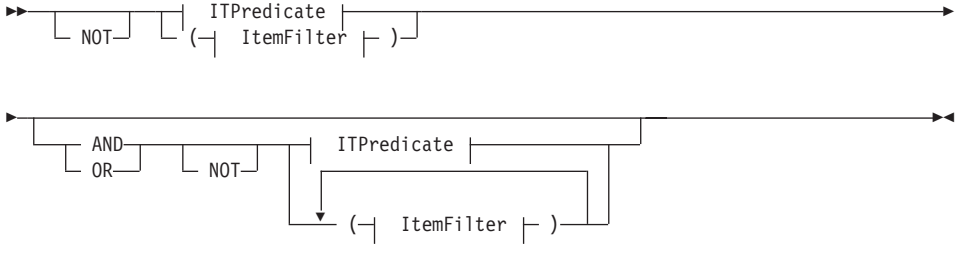
In C and C++, any items retrieved are appended to the supplied vector. If you want to read the current items only, you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

The items to be retrieved can be characterized by a filter. An item filter is specified as a character string.

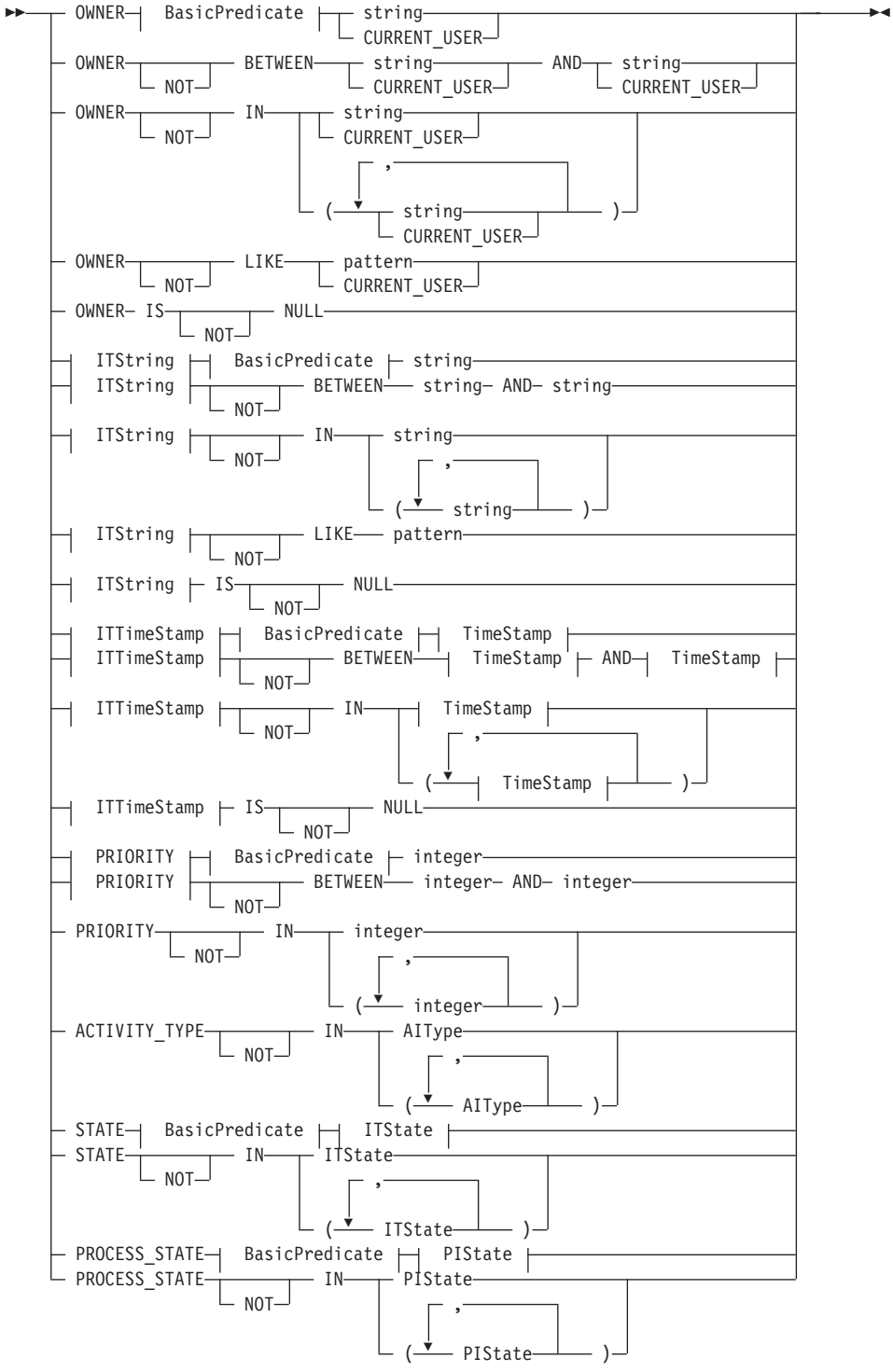
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

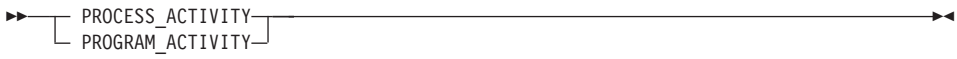
ItemFilter



ITPredicate


OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
,
(
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
,
(
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
,
(
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
,
(
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
,
(
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
,
(
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
,
(
PIState
)

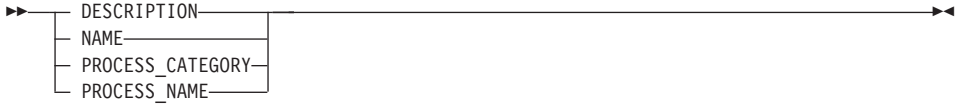
AIType



BasicPredicate

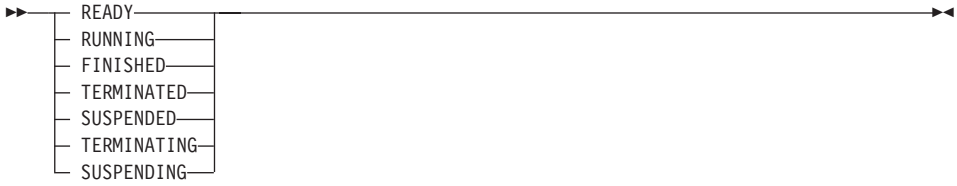
ITState

ITString

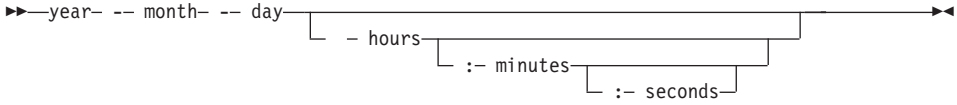


ITTimeStamp

PIState



TimeStamp



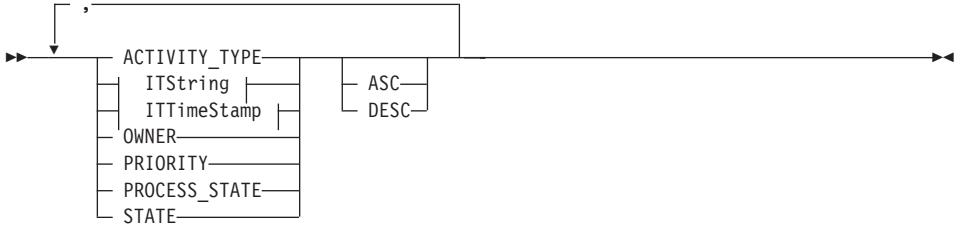
Items can be sorted. An item sort criterion is specified as a character string.

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 respectively the PIState 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 note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryItems(
       FmcjExecutionServiceHandle  service,
       char const *                filter,
       char const *                sortCriteria,
       unsigned long const *       threshold,
       FmcjItemHandle *            items )
```

C++ language signature

```
APIRET QueryItems(
       string const *             filter,
       string const *             sortCriteria,
       unsigned long const *      threshold,
       vector<FmcjItem> &         items ) const
```

Java signature

```
public abstract
Item[] queryItems(
       String                     filter,
       String                     sortCriteria,
       Integer                    threshold   ) throws FmcException
```

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 of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query process instances (C-language)" on page 783.
- For a C++ example see "Query process instances (C++)" on page 784.
- For a Java example see "Query process instances (Java)" on page 786.

QueryProcessInstanceLists()

This API call retrieves the process instance lists the user has access to from the MQ Workflow execution server (action call).

In C and C++, 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 call this API call. This means that you should set the vector handle to 0 in the C-language, respectively erase all elements of the vector in the C++ API. In ActiveX, the process instance list array on the ExecutionService is updated.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long QueryProcessInstanceLists()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceLists(
       FmcjExecutionServiceHandle            service,
       FmcjProcessInstanceListVectorHandle * lists )
```

C++ language signature

```
APIRET QueryProcessInstanceLists(
       vector<FmcjProcessInstanceList> & lists ) const
```

Java signature

```
public abstract
ProcessInstanceList[] queryProcessInstanceLists() throws FmcException
```

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 of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query worklists (ActiveX)” on page 770.
- For a C-language example see “Query worklists (C-language)” on page 771.
- For a C++ example see “Query worklists (C++)” on page 774.
- For a Java example see “Query worklists (Java)” on page 776.

QueryProcessInstanceNotifications()

This API call retrieves the process instance notifications the user has access to from the MQ Workflow execution server (action call).

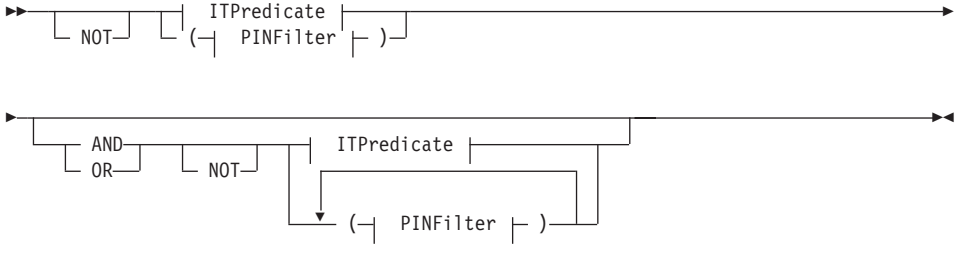
In C and C++, any process instance notifications retrieved are appended to the supplied vector. If you want to read the current process instance notifications only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language respectively 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.

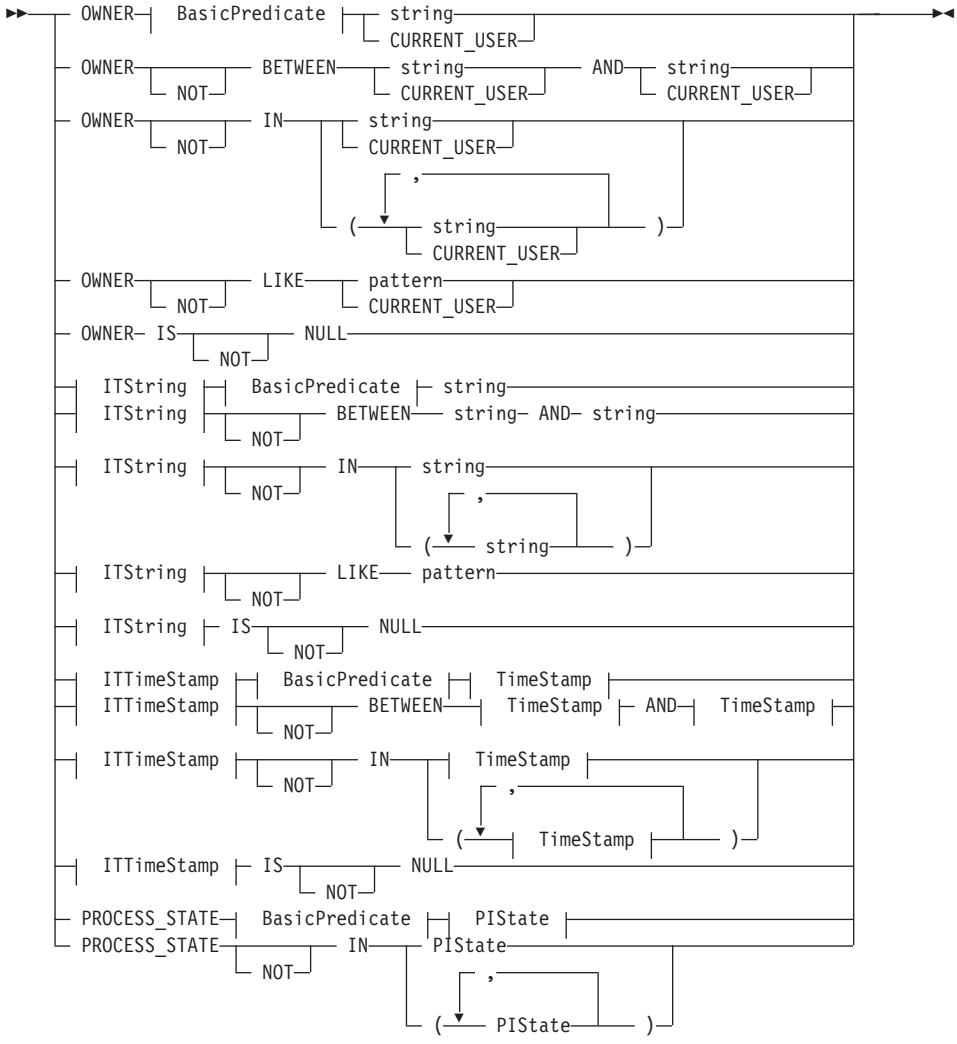
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

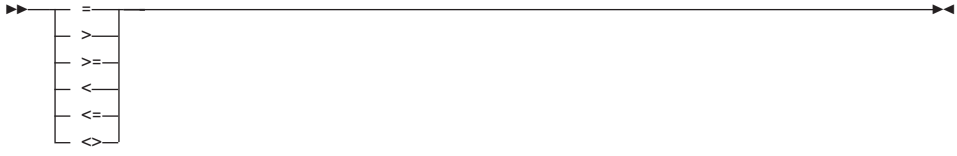
PINFilter



ITPredicate

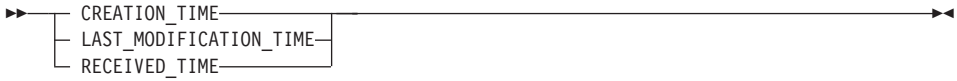


BasicPredicate

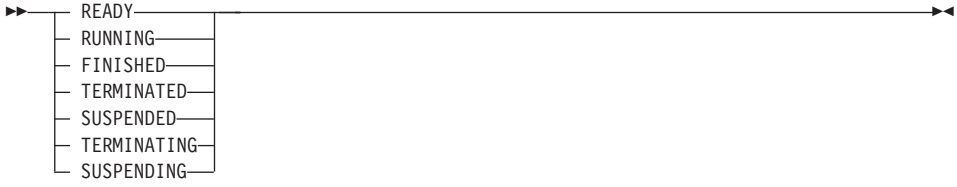


ITString

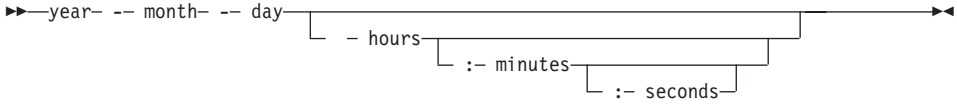
ITTimeStamp



PIState



TimeStamp



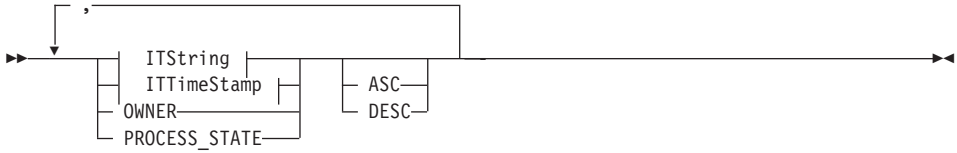
Process instance notifications can be sorted. A process instance notification sort criterion is specified as a character string.

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 respectively the PIState 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 note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX not supported

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceNotifications(
       FmcjExecutionServiceHandle                    service,
       char const *                                  filter,
       char const *                                  sortCriteria,
       unsigned long const *                         threshold,
       FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++ language signature

```
APIRET QueryProcessInstanceNotifications(
       string const *                            filter,
       string const *                            sortCriteria,
       unsigned long const *                     threshold,
       vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java signature

```
public abstract
ProcessInstanceNotification[] queryProcessInstanceNotifications(
       String                     filter,
       String                     sortCriteria,
       Integer                    threshold   ) throws FmcException
```

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 of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 783.
- For a C++ example see “Query process instances (C++)” on page 784.
- For a Java example see “Query process instances (Java)” on page 786.

QueryProcessInstances()

This API call retrieves the current process instances the user has access to from the MQ Workflow execution server (action call).

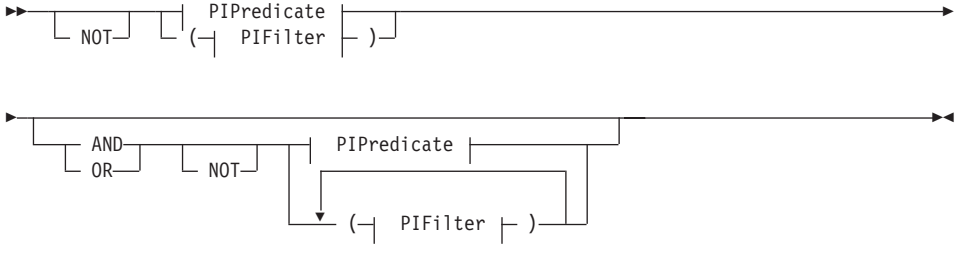
In C and C++ any process instances retrieved are appended to the supplied vector. If you want to read the current process instances only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

A filter on process instances is specified as a character string containing a filter predicate:

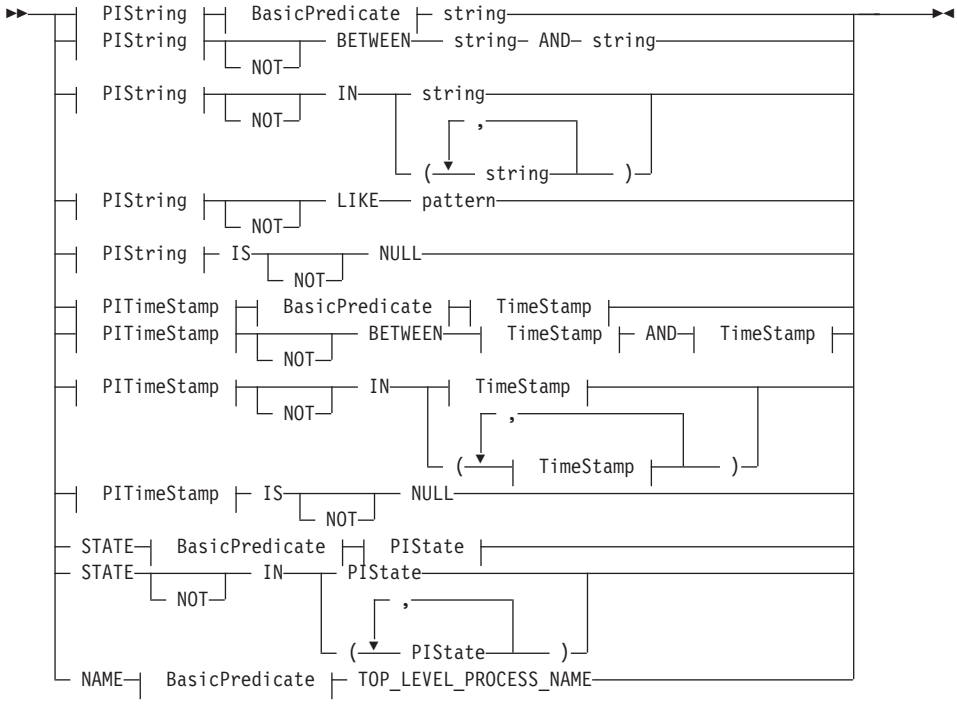
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

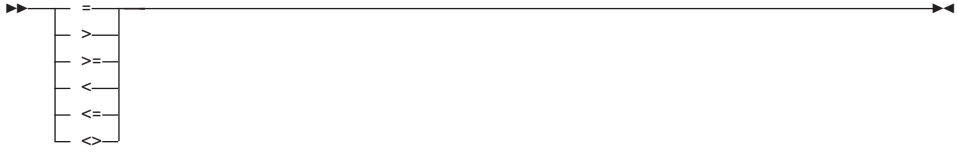
PIFilter



PIPredicate



BasicPredicate

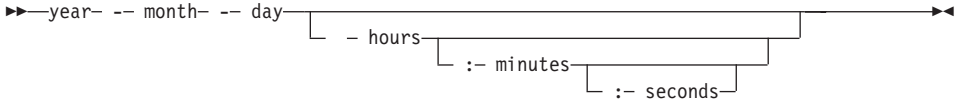


PIState

PIString

PITimeStamp

TimeStamp

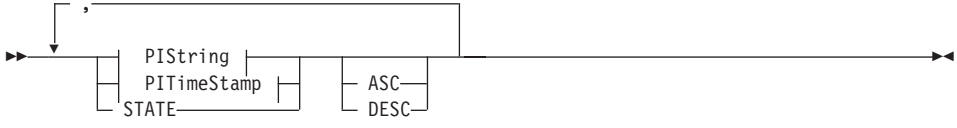


Process instances can be sorted. A process instance sort criterion is specified as a character string.

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 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
- State
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstances(
       FmcjExecutionServiceHandle        service,
       char const *                      filter,
       char const *                      sortCriteria,
       unsigned long const *             threshold,
       FmcjProcessInstanceVectorHandle * instances )
```

C++ language signature

```
APIRET QueryProcessInstances(
       string const *                  filter,
       string const *                  sortCriteria,
       unsigned long const *           threshold,
       vector<FmcjProcessInstance> &   instances ) const
```

Java signature

```
public abstract
ProcessInstance[]  queryProcessInstances(
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

Parameters

filter Input. The filter criteria which characterize the process instances to be retrieved.

instances Input/Output. The qualifying vector of process instances.

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 instances found.

threshold Input. The threshold which defines the maximum number of process instances to be returned to the client.

Return type

APIRET The return code of calling 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.

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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query process instances (C-language)" on page 783.
- For a C++ example see "Query process instances (C++)" on page 784.
- For a Java example see "Query process instances (Java)" on page 786.

QueryProcessTemplateLists()

This API call retrieves the current process template lists the user has access to from the MQ Workflow execution server (action call).

In C and C++, 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 call this API call. This means that you should set the vector handle to 0 in the C-language respectively erase all elements of the vector in the C++ API. In ActiveX, the process template list array on the ExecutionService is updated.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long QueryProcessTemplateLists()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplateLists(
       FmcjExecutionServiceHandle           service,
       FmcjProcessTemplateListVectorHandle * lists )
```

C++ language signature

```
APIRET QueryProcessTemplateLists(
       vector<FmcjProcessTemplateList> &  lists ) const
```

Java signature

```
public abstract
ProcessTemplateList[] queryProcessTemplateLists() throws FmcException
```

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 of calling 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 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query worklists (ActiveX)” on page 770.
- For a C-language example see “Query worklists (C-language)” on page 771.
- For a C++ example see “Query worklists (C++)” on page 774.
- For a Java example see “Query worklists (Java)” on page 776.

QueryProcessTemplates()

This API call retrieves the current process templates from the MQ Workflow execution server (action call).

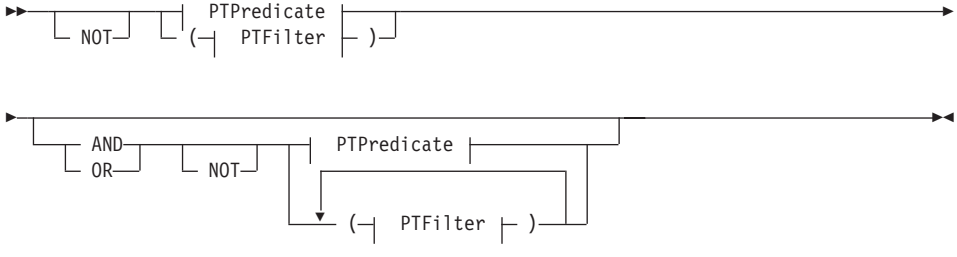
In C and C++, any process templates retrieved are appended to the supplied vector. If you want to read the current process templates only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

A filter on process templates is specified as a character string containing a filter predicate:

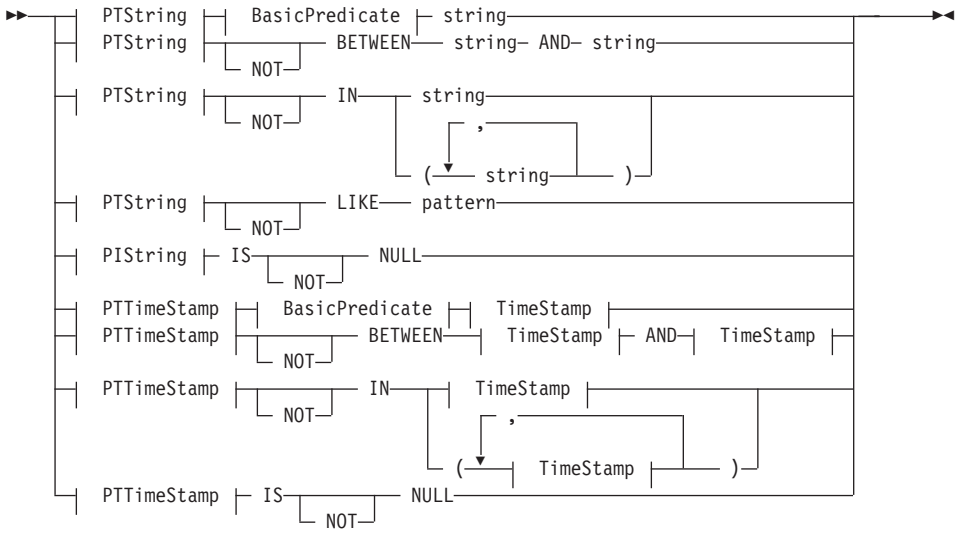
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PTFilter



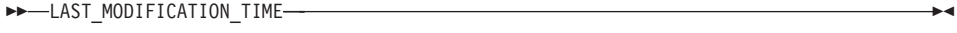
PTPredicate



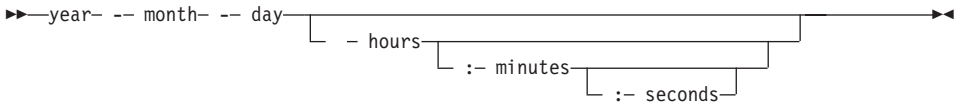
BasicPredicate

PTString

PTTimeStamp



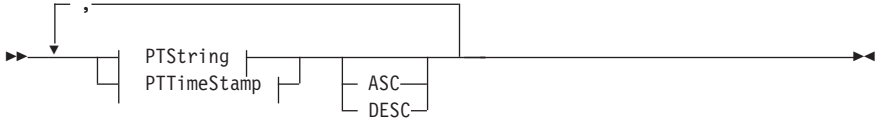
TimeStamp



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 note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplates(
       FmcjExecutionServiceHandle        service,
       char const *                      filter,
       char const *                      sortCriteria,
       unsigned long const *             threshold,
       FmcjProcessTemplateVectorHandle * templates )
```

C++ language signature

```
APIRET QueryProcessTemplates(
       string const *                filter,
       string const *                sortCriteria,
       unsigned long const *         threshold,
       vector<FmcjProcessTemplate> & templates ) const
```

Java signature

```
public abstract
ProcessTemplates[] queryProcessTemplates(
       String                        filter,
       String                        sortCriteria,
       Integer                       threshold   ) throws FmcException
```

Parameters

filter	Input. The filter criteria which characterize the process templates 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 process templates found.
templates	Input/Output. The qualifying vector of process templates.
threshold	Input. The threshold which defines the maximum number of process templates to be returned to the client.

Return type

APIRET
The return code of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 783 .
- For a C++ example see “Query process instances (C++)” on page 784.
- For a Java example see “Query process instances (Java)” on page 786.

QueryWorkitems()

This API call retrieves the work items the user has access to from the MQ Workflow execution server (action call).

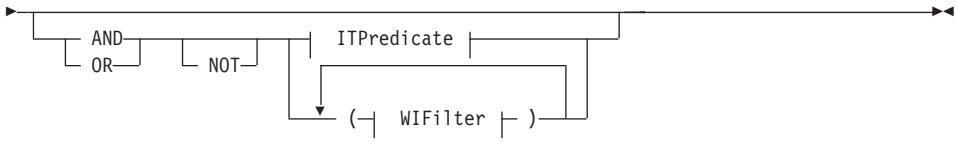
In C and C++, any work items retrieved are appended to the supplied vector. If you want to read the current work items only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language respectively erase all elements of the vector 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:

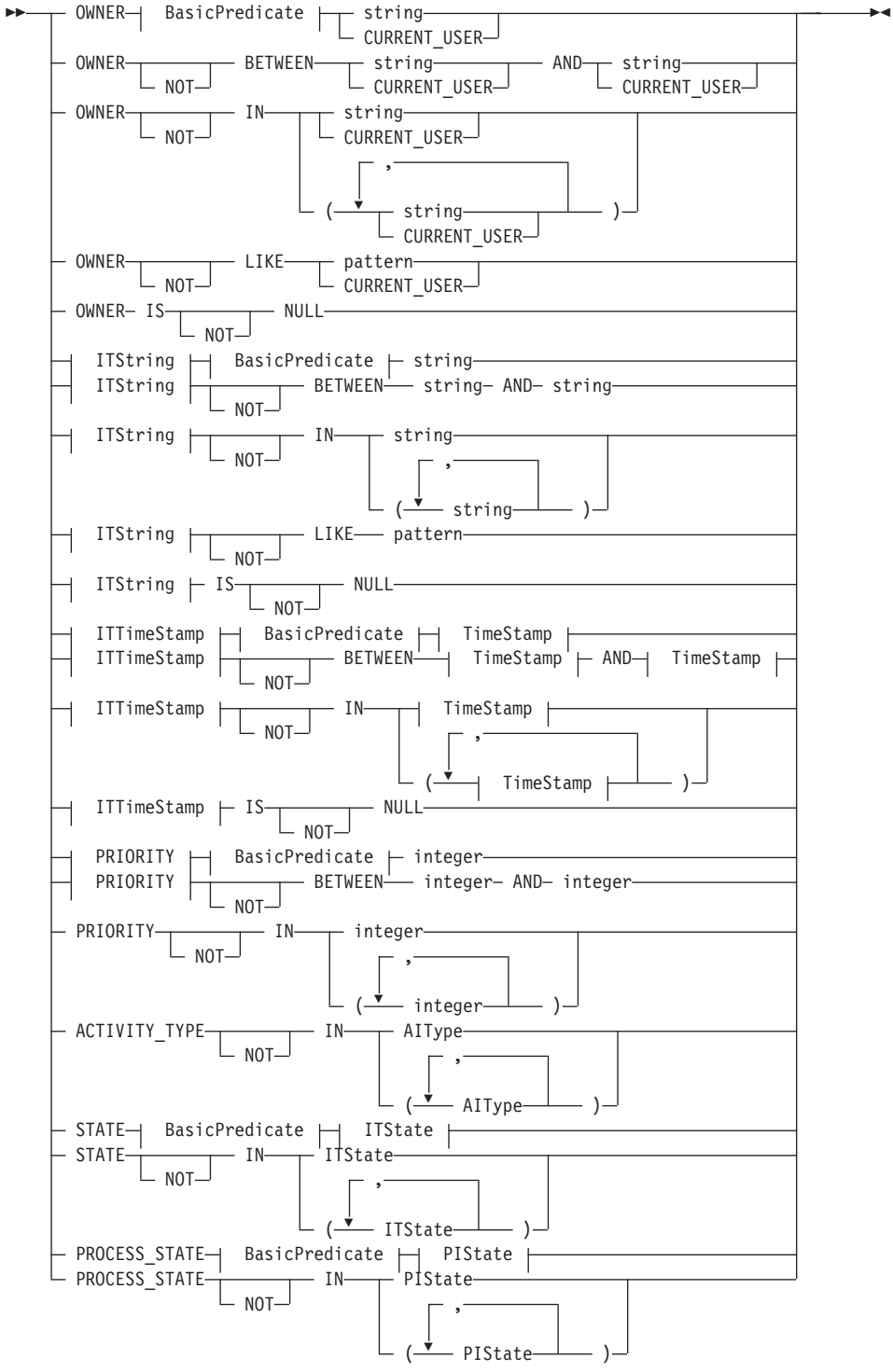
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. 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. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

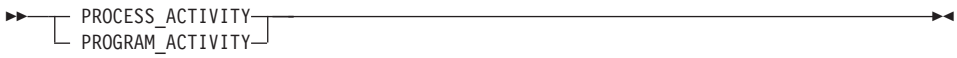
WIFilter



ITPredicate


OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
(
,
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
(
,
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
(
,
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
(
,
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
(
,
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
(
,
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
(
,
PIState
)

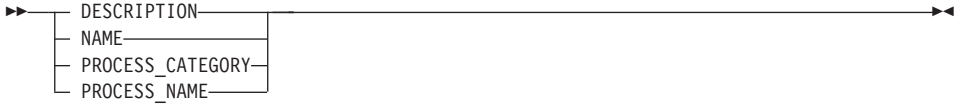
AIType



BasicPredicate

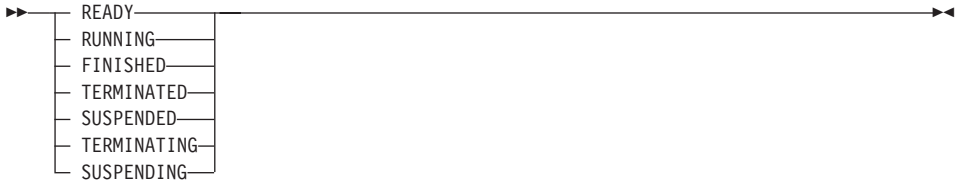
ITState

ITString

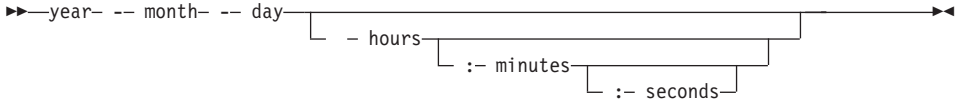


ITTimeStamp

PIState



TimeStamp



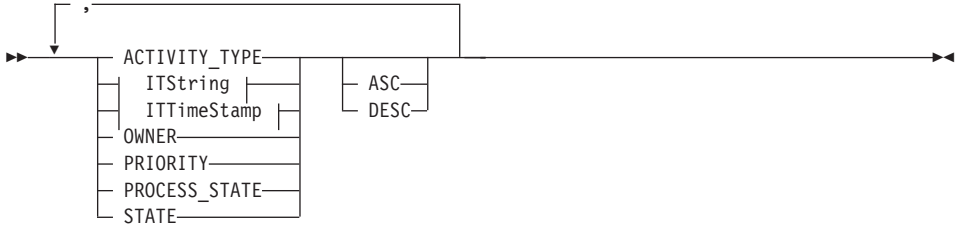
Work items can be sorted. A work item sort criterion is specified as a character string.

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 respectively the PIState 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 note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorkitems(
       FmcjExecutionServiceHandle  service,
       char const *                filter,
       char const *                sortCriteria,
       unsigned long const *       threshold,
       FmcjWorkitemVectorHandle *  workitems )
```

C++ language signature

```
APIRET QueryWorkitems(
       string const *                filter,
       string const *                sortCriteria,
       unsigned long const *         threshold,
       vector<FmcjWorkitem> &        workitems ) const
```

Java signature

```
public abstract
WorkItem[] queryWorkItems(
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

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 of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query process instances (C-language)" on page 783.
- For a C++ example see "Query process instances (C++)" on page 784.
- For a Java example see "Query process instances (Java)" on page 786.

You query work items similar to querying process instances.

QueryWorklists()

This API call retrieves the worklists the user has access to from the MQ Workflow execution server (action call).

In C and C++, 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 call this API call. This means that you should set the vector handle to 0 in the C-language respectively erase all elements of the vector in the C++ API. In ActiveX, the worklist array on the ExecutionService array is updated.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long QueryWorklists()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorklists(
       FmcjExecutionServiceHandle  service,
       FmcjWorklistVectorHandle *  lists )
```

C++ language signature

```
APIRET QueryWorklists( vector<FmcjWorklist> & lists ) const
```

Java signature

```
public abstract
WorkList[] queryWorkLists()  throws FmcException
```

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 of calling 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 MQ 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_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query worklists (ActiveX)" on page 770.
- For a C-language example see "Query worklists (C-language)" on page 771.
- For a C++ example see "Query worklists (C++)" on page 774.
- For a Java example see "Query worklists (Java)" on page 776.

Receive()

This API call allows for receiving data pushed by an MQ 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, that is, point to a buffer that contains all zeros (0x00). 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 280 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:

ActiveX	not supported
C-language	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-language	Fmc_DART_NotSet
C++	FmcjExecutionData::NotSet

Terminate(2) Indicates that receiving data can end.

C-language	Fmc_DART_Terminate
C++	FmcjExecutionData::Terminate

ItemDeleted(1000)
Indicates that a work item, an activity instance notification, or a process instance notification has been deleted.

C-language	Fmc_DART_ItemDeleted
C++	FmcjExecutionData::ItemDeleted

Workitem(1002)
Indicates that a work item has been created or updated.

C-language Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

ActivityInstanceNotification(1003)
Indicates that an activity instance notification has been created or updated.

C-language Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

ProcessInstanceNotification(1004)
Indicates that a process instance notification has been created or updated.

C-language Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

ExecuteInstanceResponse(1100)
Indicates that the execution data contains the response on an ExecuteProcessInstance() request.

C-language Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

ExecuteProgramResponse(1101)
Indicates that the execution data contains the response on an ExecuteProgram() request.

C-language Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server (present session mode)

API interface declarations

ActiveX not applicable

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA not supported

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceReceive(
       FmcjExecutionServiceHandle   service,
       FmcjCorrelID *               correlID,
       FmcjExecutionDataHandle *    data,
       signed long                  timeout )
```

C++ language signature

```
APIRET Receive( FmcjCorrelID *      correlID,
                FmcjExecutionData & data,
                signed long         timeout  ) const
```

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 MQ 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 of calling 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.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 MQ Workflow execution server from within this program (activity-implementation call).

An activity implementation started by an MQ Workflow program execution agent can request services from that program execution agent without further identification. It is known by the program execution agent.

When the activity implementation decides to distribute work among other programs and starts those programs as separate operating system processes, then those processes are unknown by the program execution agent and cannot request services. The activity implementation can, however, ask the program execution agent for its program identification and pass that identification to the programs started. That is, the programs started receive the authorization to talk to the program execution agent as long as the actual activity implementation is alive.

The started programs can then request services from the program execution agent by themselves by specifying this program identification.

When successfully executed, a session to the same execution server is set up from where 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.

Usage note

- See "Activity implementation API calls" on page 151 for general information.

Authorization

Valid program identification

Required connection

None but MQ Workflow program execution agent must be active

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long RemotePassthrough( BSTR programID )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceRemotePassthrough(
       FmcjExecutionServiceHandle  service )
       char const *                programID )
```

C++ language signature

```
APIRET RemotePassthrough( string const & programID )
```

Java signature

```
public abstract
void remotePassthrough( String programID ) throws FmcException
```

Parameters

programID Input. The program identification by which the actually started activity implementation is known to the program execution agent.

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 of calling 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 agent is not active.

FMC_ERROR_TOOL_FUNCTION(128)
Passthrough cannot be called from a program started by a support tool or from a program started by the program execution server.

FMC_ERROR_USERID_UNKNOWN(10)
The user who started the work item does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetPersonAbsent()

This API call sets the absence indication of the specified 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 note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX signature

```
long SetPersonAbsent( BSTR userID, boolean newValue )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceSetPersonAbsent(
       FmcjExecutionServiceHandle  service,
       char const *                userID,
       bool                        newValue )
```

C++ language signature

```
APIRET SetPersonAbsent( string const * userID   = 0,
                        bool           newValue = true )
```

Java signature

```
public abstract
void   setPersonAbsent() throws FmcException

public abstract
void   setPersonAbsent2( String userID, boolean newValue )
throws FmcException
```

Parameters

service Input. The handle of the service object where a person's absence is to be set.

newValue Input. True, if the person is denoted as absent, else false.

userID Input. The user ID of the person whose absence is to be set. When no user ID is provided in C++, the absence of the logged-on user is set.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_USERID_UNKNOWN(10)
The specified user ID is not known.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 MQ 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, that is, points to a buffer that contains all zeros (0x00), then a correlation ID is returned which can be used to explicitly receive this data.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

None

API interface declarations

ActiveX	not supported
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	not supported

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceTerminateReceive(
       FmcjExecutionServiceHandle  service,
       FmcjCorrelID *              correlID )
```

C++ language signature

```
APIRET TerminateReceive( FmcjCorrelID * correlID = 0 )
```

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 of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 39. Instance monitor actions

An InstanceMonitor object represents a monitor for a process instance, an activity instance of type *Block*, or an activity instance of type *Process*.

The following sections describe the actions which can be applied on an instance monitor. See "InstanceMonitor" on page 294 for a complete list of API calls.

ObtainInstanceMonitor()/ ObtainBlockMonitor()/ ObtainProcessMonitor()

This API call retrieves the instance monitor for the specified activity instance from the MQ Workflow execution server (action call). If the requested instance monitor has already been retrieved from the server, the monitor found in the API cache is returned to the caller.

When the monitor for a process instance is retrieved, the specified activity instance must be of type *Process* and be part of this instance monitor.

When the monitor for a block is retrieved, the specified activity instance must be of type *Block* and be part of this instance monitor.

When the deep option is specified, nested activity instances of type *Block* are resolved, that is, their instance monitors are also read from the server.

Note: Deep is currently not supported.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.InstanceMonitor

ActiveX signature

```
InstanceMonitor *
ObtainInstanceMonitor( long *             returnCode,
                       ActivityInstance * activity,
                       boolean            deep    )
```

C-language signature

```
FmcjInstanceMonitorHandle FMC_APIENTRY
    FmcjInstanceMonitorObtainBlockMonitor(
                       FmcjInstanceMonitorHandle  hdlMonitor,
                       FmcjActivityInstanceHandle activity  )

FmcjInstanceMonitorHandle FMC_APIENTRY
    FmcjInstanceMonitorObtainProcessMonitor(
                       FmcjInstanceMonitorHandle  hdlMonitor,
                       FmcjActivityInstanceHandle activity,
                       bool                       deep      )
```

C++ language signature

```
FmcjInstanceMonitor *
       ObtainBlockMonitor ( FmcjActivityInstance const & activity ) const

APIRET ObtainBlockMonitor ( FmcjActivityInstance const & activity,
                            FmcjInstanceMonitor &        monitor ) const

FmcjInstanceMonitor *
       ObtainProcessMonitor(FmcjActivityInstance const & activity,
                            bool                         deep = false) const

APIRET ObtainProcessMonitor(FmcjActivityInstance const & activity,
                            FmcjInstanceMonitor &        monitor,
                            bool                         deep = false) const
```

Java signature

```
public abstract
InstanceMonitor obtainBlockMonitor  ( ActivityInstance activity )
                throws FmcException

public abstract
InstanceMonitor obtainProcesskMonitor( ActivityInstance activity,
                                       boolean          deep    )
                throws FmcException
```

Parameters

activity Input. The activity instance of type *Block* or *Process* whose instance monitor is to be retrieved.

deep Input. An indicator whether monitors of activity instances of type *Block* are to be resolved, that is, their monitors are also to be retrieved. Note that deep is currently not supported.

hdlMonitor Input. The instance monitor containing the activity instance of type *Block* or *Process*.

monitor Input/Output. The instance monitor retrieved.

returnCode Input/Output. The result of calling this API call - see return codes below.

Return type

APIRET The result of calling this API call - see return codes below.

FmcjInstanceMonitor*/Handle/InstanceMonitor
The instance monitor respectively a pointer of handle to the instance monitor.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 instance monitor or does 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_WRONG_KIND(501)
The specified activity instance is not of type *Block* or *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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the instance monitor from the MQ Workflow execution server (action call).

All information about the instance monitor is retrieved.

When the deep option is specified, then activity instances of type *Block* are resolved, that is, their instance monitors are also refreshed from the server.

Note: Deep is currently not supported.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.InstanceMonitor

ActiveX signature

```
long Refresh ( boolean deep )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjInstanceMonitorRefresh(
                        FmcjInstanceMonitorHandle  hdlMonitor,
                        bool                       deep      )
```

C++ language signature

```
APIRET Refresh( bool deep = false )
```

Java signature

```
public abstract
void Refresh( boolean deep ) throws FmcException
```

Parameters

deep Input. An indicator whether activity instances of type *Block*

are to be resolved, that is, their monitors are also to be provided. Note, deep is currently ignored.

hdlMonitor Input. The instance monitor to be refreshed.

Return type

long The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 40. Item actions

An FmcjItem or 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 the C++ language, FmcjItem is thus the superclass of the FmcjWorkitem, FmcjActivityInstanceNotification, and FmcjProcessInstanceNotification classes and provides for all common properties and methods. In the Java language, Item is thus a superclass of the WorkItem, ActivityInstanceNotification, and ProcessInstanceNotification classes and provides for all common properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all methods are explicitly defined on the appropriate classes. Note, however, that they are described here as Item actions.

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 296 for a complete list of API calls.

Delete()

This API call deletes the specified item from the MQ 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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the item owner
- Work item authorization for the item owner
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX signature

```
long ActivityInstanceNotification.Delete()

long ProcessInstanceNotification.Delete()

long Workitem.Delete()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemDelete( FmcjItemHandle hdlItem )

#define FmcjActivityInstanceNotificationDelete FmcjItemDelete
#define FmcjProcessInstanceNotificationDelete  FmcjItemDelete
#define FmcjWorkitemDelete                     FmcjItemDelete
```

C++ language signature

```
APIRET Delete()
```

Java signature

```
public abstract
void   delete() throws FmcException
```

Parameters

hdlItem Input. The handle of the item to be deleted.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()/ObtainInstanceMonitor

This API call retrieves the instance monitor for the process instance the item is part of from the MQ Workflow execution server (action call).

When the deep option is specified, then activity instances of type Block are resolved, that is, their monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the 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 instance monitor handle already points to some object.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX signature

```
InstanceMonitor*
ActivityInstanceNotification.ObtainInstanceMonitor(
                             long *    returnCode,
                             boolean   deep      )

InstanceMonitor*
ProcessInstanceNotification.ObtainInstanceMonitor(
                            long *    returnCode,
                            boolean   deep      )

InstanceMonitor*
Workitem.ObtainInstanceMonitor(
                            long *    returnCode,
                            boolean   deep      )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemObtainProcessMonitor(
                    FmcjItemHandle               hdlItem,
                    bool                         deep,
                    FmcjInstanceMonitorHandle *  monitor)

#define FmcjActivityInstanceNotificationObtainProcessMonitor
        FmcjItemObtainProcessMonitor
#define FmcjProcessInstanceNotificationObtainProcessMonitor
        FmcjItemObtainProcessMonitor
#define FmcjWorkitemObtainProcessMonitor
        FmcjItemObtainProcessMonitor
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false ) const
```

Java signature

```
public abstract
InstanceMonitor obtainProcesseMonitor( boolean deep )
throws FmcException
```

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 currently ignored.

hdlItem Input. The item whoseinstance monitor is to be retrieved.

monitor Input/Output. The address of the handle to the monitor respectively the 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 of calling this API call - see return codes below.

InstanceMonitorHandle*/ InstanceMonitor

A pointer to the instance monitor or the 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)

A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)

Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ProcessInstance()

This API call retrieves the process instance the item is a part of from the MQ 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 note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX signature

```
ProcessInstance*
ActivityInstanceNotification.ProcessInstance( long * returnCode)

ProcessInstancer*
ProcessInstanceNotification.ProcessInstance( long * returnCode)

ProcessInstance*
Workitem.ProcessInstance( long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemProcessInstance(
       FmcjItemHandle               hdlItem,
       FmcjProcessInstanceHandle *  instance )

#define FmcjActivityInstanceNotificationProcessInstance
        FmcjItemProcessInstance
#define FmcjProcessInstanceNotificationProcessInstance
        FmcjItemProcessInstance
#define FmcjWorkitemProcessInstance
        FmcjItemProcessInstance
```

C++ language signature

```
APIRET ProcessInstance( FmcjProcessInstance & instance ) const
```

Java signature

```
public abstract
ProcessInstance processInstance() throws FmcException
```

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 of calling 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.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the item from the MQ Workflow execution server (action call).

All information about the item, primary and secondary, is retrieved.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the item owner
- Work item authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX signature

```
long ActivityInstanceNotification.Refresh()

long ProcessInstanceNotification.Refresh()

long Workitem.Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemRefresh( FmcjItemHandle hdlItem )

#define FmcjActivityInstanceNotificationRefresh FmcjItemRefresh
#define FmcjProcessInstanceNotificationRefresh  FmcjItemRefresh
#define FmcjWorkitemRefresh                     FmcjItemRefresh
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

Parameters

hdlItem Input. The handle of the item object to be refreshed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

Be the item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx

JAVA com.ibm.workflow.api.Item

ActiveX signature

```
long ActivityInstanceNotification.SetDescription(
                                  BSTR      description,
                                  boolean   isNull     )

long ProcessInstanceNotification.SetDescription(
                                  BSTR       description,
                                  boolean    isNull     )

long Workitem.SetDescription(     BSTR      description,
                                  boolean   isNull     )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemSetDescription(
                           FmcjItemHandle  hdlItem,
                           char const *    description )

#define FmcjActivityInstanceNotificationSetDescription
        FmcjItemSetDescription
#define FmcjProcessInstanceNotificationSetDescription
        FmcjItemSetDescription
#define FmcjWorkitemSetDescription
        FmcjItemSetDescription
```

C++ language signature

```
APIRET SetDescription( string const * description )
```

Java signature

```
public abstract
void   setDescription( String description ) throws FmcException
```

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.

isNull Input. If set to *True,* indicates that any description of the item is to be reset.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 respectively 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 note

- See "Action API calls" on page 150 for general information.

Authorization

Be the item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx

JAVA com.ibm.workflow.api.Item

ActiveX signature

```
long ActivityInstanceNotification.SetName( BSTR name )

long ProcessInstanceNotification.SetName( BSTR name )

long Workitem.SetName( BSTR name )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemSetName( FmcjItemHandle  hdlItem,
                                     char const *    name )

#define FmcjActivityInstanceNotificationSetName FmcjItemSetName
#define FmcjProcessInstanceNotificationSetName  FmcjItemSetName
#define FmcjWorkitemSetName                     FmcjItemSetName
```

C++ language signature

```
APIRET SetName( string const * name )
```

Java signature

```
public abstract
void   setName( String name ) throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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*, *Suspended*, or *Terminated* and the associated process instance in states *Running*, *Suspending*, or *Suspended*. Work items in states InError or Terminated can only be transferred to the process administrator.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Workitem authority for the persons to transfer from/to
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX signature

```
long ActivityInstanceNotification.Transfer( BSTR userID )

long ProcessInstanceNotification.Transfer( BSTR userID )

long Workitem.Transfer( BSTR userID )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjItemTransfer( FmcjItemHandle hdlItem,
                                      char const *   userID )

#define FmcjActivityInstanceNotificationTransfer FmcjItemTransfer
#define FmcjProcessInstanceNotificationTransfer  FmcjItemTransfer
#define FmcjWorkitemTransfer                     FmcjItemTransfer
```

C++ language signature

```
APIRET Transfer( string const & userID )
```

Java signature

```
public abstract
void   transfer( String userID ) throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 41. Persistent list actions

An FmcjPersistentList or 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 the C++ language, FmcjPersistentList is thus the superclass of the FmcjProcessInstanceList, FmcjProcessTemplateList, and FmcjWorklist classes and provides for all common properties and methods. In the Java language, PersistentList is thus a superclass of the ProcessInstanceList, ProcessTemplateList, and Worklist classes and provides for all common properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all methods are explicitly defined on the appropriate classes. Note, however, that they are described here as PersistentList actions.

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 300 for a complete list of API calls.

Delete()

This API call deletes the specified persistent list from the MQ Workflow execution server (action call).

The transient representation of the persistent list is not impacted; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.Delete()

long ProcessTemplateList.Delete()

long Worklist.Delete()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListDelete( FmcjPersistentListHandle hdlList )

#define FmcjProcessInstanceListDelete FmcjPersistentListDelete
#define FmcjProcessTemplateListDelete FmcjPersistentListDelete
#define FmcjWorklistDelete            FmcjPersistentListDelete
```

C++ language signature

```
APIRET Delete()
```

Java signature

```
public abstract
void   delete() throws FmcException
```

Parameters

hdlList Input. The handle of the persistent list to be deleted.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh

This API call refreshes the persistent list from the MQ Workflow execution server (action call).

All information about the persistent list is retrieved, for example, its description, its filter, or its sort criteria.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.Refresh()

long ProcessTemplateList.Refresh()

long Worklist.Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListRefresh( FmcjPersistentListHandle hdlList )

#define FmcjProcessInstanceListRefresh FmcjPersistentListRefresh
#define FmcjProcessTemplateListRefresh FmcjPersistentListRefresh
#define FmcjWorklistRefresh            FmcjPersistentListRefresh
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

Parameters

hdlList Input. The handle of the persistent list to be refreshed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.SetDescription(
                         BSTR         description,
                         boolean      isNull     )

long ProcessTemplateList.SetDescription(
                         BSTR         description,
                         boolean      isNull     )

long Worklist.SetDescription(
                         BSTR         description,
                         boolean      isNull     )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListSetDescription( FmcjPersistentListHandle hdlList,
                                  char const *             description )

#define FmcjProcessInstanceListSetDescription
        FmcjPersistentListSetDescription
#define FmcjProcessTemplateListSetDescription
        FmcjPersistentListSetDescription
#define FmcjWorklistSetDescription
        FmcjPersistentListSetDescription
```

C++ language signature

```
APIRET SetDescription( string const * description )
```

Java signature

```
public abstract
void   setDescription( String description ) throws FmcException
```

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.

isNull Input. If set to *True*, indicates that any description is to be removed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.SetFilter(
                         BSTR         filter,
                         boolean      isNull     )

long ProcessTemplateList.SetFilter(
                         BSTR         filter,
                         boolean      isNull     )

long Worklist.SetFilter( BSTR         filter,
                         boolean      isNull     )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListSetFilter( FmcjPersistentListHandle  hdlList,
                             char const *              filter )

#define FmcjProcessInstanceListSetFilter FmcjPersistentListSetFilter
#define FmcjProcessTemplateListSetFilter FmcjPersistentListSetFilter
#define FmcjWorklistSetFilter            FmcjPersistentListSetFilter
```

C++ language signature

```
APIRET SetFilter( string const * filter )
```

Java signature

```
public abstract
void   setFilter( String filter ) throws FmcException
```

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.
isNull	Input. If set to *True*, indicates that any filter is to be removed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )

long ProcessTemplateList.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )

long Worklist.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListSetSortCriteria( FmcjPersistentListHandle hdlList,
                                   char const *             sortCriteria )

#define FmcjProcessInstanceListSetSortCriteria
        FmcjPersistentListSetSortCriteria
#define FmcjProcessTemplateListSetSortCriteria
        FmcjPersistentListSetSortCriteria
#define FmcjWorklistSetSortCriteria
        FmcjPersistentListSetSortCriteria
```

C++ language signature

```
APIRET SetSortCriteria( string const * sortCriteria )
```

Java signature

```
public abstract
void   setSortCriteria( String sortCriteria ) throws FmcException
```

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).

isNull Input. If set to *True*, indicates that any sort criteria are to be removed.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX signature

```
long ProcessInstanceList.SetThreshold(
                         long         threshold,
                         boolean      isNull     )

long ProcessTemplateList.SetThreshold(
                         long         threshold,
                         boolean      isNull     )

long Worklist.SetThreshold(
                         long         threshold,
                         boolean      isNull     )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListSetThreshold( FmcjPersistentListHandle hdlList,
                                unsigned long const *    threshold )

#define FmcjProcessInstanceListSetThreshold  FmcjPersistentListSetThreshold
#define FmcjProcessITemplateListSetThreshold FmcjPersistentListSetThreshold
#define FmcjWorklistSetThreshold             FmcjPersistentListSetThreshold
```

C++ language signature

```
APIRET SetThreshold( unsigned long const * threshold )
```

Java signature

```
public abstract
void   setThreshold( Integer threshold ) throws FmcException
```

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).

isNull Input. If set to *True*, indicates that any threshold is to be erased.

Return type

long/ APIRET The return code of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 42. Person actions

An FmcjPerson or a Person object represents an MQ 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 301 for a complete list of API calls.

Refresh()

This API call refreshes the person from the MQ Workflow execution server (action call).

All information about the person, primary and secondary, is retrieved.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Person

ActiveX signature

```
long Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjPersonRefresh( FmcjPersonHandle hdlPerson )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Person

ActiveX signature

```
long SetAbsence( boolean newValue )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjPersonSetAbsence(
       FmcjPersonHandle  hdlPerson,
       bool              newValue )
```

C++ language signature

```
APIRET SetAbsence( bool newValue )
```

Java signature

```
public abstract
void   setAbsence( boolean newValue ) throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 MQ Workflow user ID other than the logged-on user. If no substitute is provided, the substitute of the logged-on user is erased.

Note: Changing the substitute can result in changes of persons who are authorized to access the (work) items of the logged-on user. You must refresh the person object to read the updated definitions.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Person

ActiveX signature

```
long SetSubstitute( BSTR substitute, boolean isNull )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjPersonSetSubstitute(
                    FmcjPersonHandle  hdlPerson,
                    char const *      substitute )
```

C++ language signature

```
APIRET SetSubstitute( string const * substitute )
```

Java signature

```
public abstract
void   setSubstitute( String substitute ) throws FmcException
```

Parameters

hdlPerson	Input. The handle of the person object whose substitute is to be set.
isNull	Input. If set to *True*, any substitute specification is removed.
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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 MQ 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 43. Process instance actions

An FmcjProcessInstance or 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 MQ Workflow.

The following diagram provides an overview on the possible process instance states and the actions which are allowed in those states, provided that the appropriate authority has been granted and that more specific requirements stated in the API calls descriptions have been fulfilled.

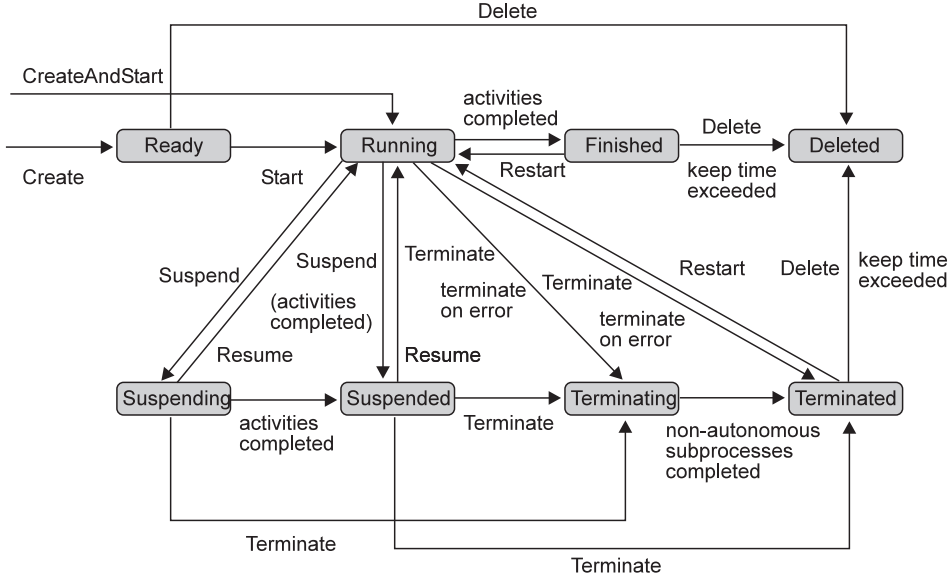


Figure 23. Process instance states

The following sections describe the actions which can be applied on a process instance. See "ProcessInstance" on page 307 for a complete list of API calls.

Delete()

This API call deletes the specified process instance from the MQ 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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Process administration authorization
- Be the process instance creator
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Delete()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceDelete( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Delete()
```

Java signature

```
public abstract
void   delete() throws FmcException
```

Parameters

hdlInstance Input. The handle of the process instance to be deleted.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the process instance from the MQ 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 responsible for the ownership of objects, that is, it is not checked whether the container handle already points to some object.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long InContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceInContainer(
       FmcjProcessInstanceHandle      hdlInstance,
       FmcjReadWriteContainerHandle * input )
```

C++ language signature

```
APIRET InContainer( FmcjReadWriteContainer & input )
```

Java signature

```
public abstract
ReadWriteContainer inContainer() throws FmcException
```

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 respectively the input container of the process instance to be set.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()

This API call obtains a monitor for the process instance from the MQ Workflow execution server (action call).

When the deep option is specified, then activity instances of type Block are resolved, that is, their monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the 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 instance monitor handle already points to some object.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
InstanceMonitor* ObtainMonitor( long *    returnCode,
                                boolean   deep         )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceObtainProcessMonitor(
                    FmcjProcessInstanceHandle   hdlInstance,
                    bool                        deep,
                    FmcjInstanceMonitorHandle * monitor )
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false )
```

Java signature

```
public abstract
InstanceMonitor obtainProcessMonitor( boolean deep ) throws FmcException
```

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 currently ignored.

hdlInstance Input. The handle of the process instance object whose monitor is to be retrieved.

monitor Input/Output. The address of the monitor handle respectively the monitor of the process instance to be set.

returnCode Input/Output. A pointer to the result of the method call - see return codes below.

Return type

APIRET The return code of calling this API call - see return codes below.

InstanceMonitor*/ProcessInstanceMonitor
A pointer to the instance monitor respectively the 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the process instance from the MQ 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 responsible for the ownership of objects, that is, it is not checked whether the container handle already points to some object.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long OutContainer( Container * output )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceOutContainer(
       FmcjProcessInstanceHandle     hdlInstance,
       FmcjReadOnlyContainerHandle * output )
```

C++ language signature

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java signature

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

Parameters

hdlInstance Input. The handle of the process instance object whose output container is to be retrieved.

output Input/Output. The address of the output container or of its handle respectively the output container of the process instance to be set.

Return type

long/ APIRET The result of calling this API call - see return codes below.

ReadOnlyContainer
The output 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the process instance from the MQ Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRefresh( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

Parameters

hdlInstance Input. The handle of the process instance object to be refreshed.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This API call restarts the process instance on the MQ 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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Restart()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRestart( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Restart()
```

Java signature

```
public abstract
void   restart() throws FmcException
```

Parameters

hdlInstance Input. The handle of the process instance object to be restarted.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Resume( boolean deep )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceResume( FmcjProcessInstanceHandle hdlInstance,
                           bool                      deep )
```

C++ language signature

```
APIRET Resume( bool deep )
```

Java signature

```
public abstract
void   resume( boolean deep ) throws FmcException
```

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 calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long SetDescription( BSTR description, boolean isNull )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceSetDescription(
       FmcjProcessInstanceHandle hdlInstance,
       char const *              description )
```

C++ language signature

```
APIRET SetDescription( string const * description )
```

Java signature

```
public abstract
void   setDescription( String  description ) throws FmcException
```

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.

isNull Input. If set to *True*, any description is removed.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long SetName( BSTR name )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceSetName( FmcjProcessInstanceHandle hdlInstance,
                            char const *              name )
```

C++ language signature

```
APIRET SetName( string const & name )
```

Java signature

```
public abstract
void   setName( String name ) throws FmcException
```

Parameters

hdlInstance	Input. The handle of the process instance object whose name is to be set.
name	Input. The name to be set.

Return type

long/ APIRET	The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Start()

long StartWithContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceStart( FmcjProcessInstanceHandle    hdlInstance,
                          FmcjReadWriteContainerHandle input )
```

C++ language signatures

```
APIRET Start()

APIRET Start( FmcjReadWriteContainer const & input )
```

Java signature

```
public abstract
void   start() throws FmcException

public abstract
void   start2( ReadWriteContainer input ) throws FmcException
```

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 calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

FMC_ERROR_INVALID_CONTAINER(509)
The passed container is invalid for the process instance; wrong schema or version.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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, suspending non-autonomous subprocesses, or checked-out work items. When the activity implementations completed their executions and the non-autonomous subprocesses reached the *Suspended* state, and when the checked-out work items are checked in, the process instance is put into the *Suspended* state.

Optionally, a date may be specified up to when the process instance is suspended. The date is to be specified in local time. The process instance is then automatically resumed, together with the non-autonomous subprocesses, if the deep option had been specified.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Suspend( boolean deep )

long SuspendUntilDateTime( DateAndTime * time,
                           boolean       deep )
```

C-language signatures

```
APIRET FMC_APIENTRY FmcjProcessInstanceSuspend(
       FmcjProcessInstanceHandle  hdlInstance,
       bool                       deep )

APIRET FMC_APIENTRY FmcjProcessInstanceSuspendUntil(
       FmcjProcessInstanceHandle  hdlInstance,
       FmcjCDateTime const *      time,
       bool                       deep )
```

C++ language signatures

```
APIRET Suspend( bool deep )

APIRET Suspend( FmcjDateTime const & time, bool deep )
```

Java signature

```
public abstract
void   suspend( boolean deep ) throws FmcException

public abstract
void   suspend2( Calendar time, boolean deep ) throws FmcException
```

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 respectively a pointer to the date/time up to when the process instance is to be suspended.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 terminating non-autonomous subprocesses. 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 note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX signature

```
long Terminate()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceTerminate( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

Parameters

hdlInstance Input. The handle of the process instance object to be terminated.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 44. Process instance list 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 the C++ language, 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 the C-language, 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. In ActiveX, inheritance is not supported so that all methods are explicitly defined on ProcessInstanceList. Note, however, that they are described as PersistentList actions.

The following sections describe the actions which can be applied on a process instance list. See “ProcessInstanceList” on page 312 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 MQ Workflow execution server (action call).

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 and C++, 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 call this API call.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstanceList

ActiveX signature

```
long QueryProcessInstances()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceListQueryProcessInstances(
       FmcjProcessInstanceListHandle      hdlList,
       FmcjProcessInstanceVectorHandle *  instances )
```

C++ language signature

```
APIRET QueryProcessInstances(
       vector<FmcjProcessInstance> & instances ) const
```

Java signature

```
public abstract
ProcessInstance[] queryProcessInstances() throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.
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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query worklists (C-language)" on page 771
- For a C++ example see "Query worklists (C++)" on page 774

Chapter 45. Process template actions

An FmcjProcessTemplate or 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 lately. 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 315 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 MQ 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 note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
ProcessInstance* CreateAndStartInstance(
       BSTR                          name,
       boolean                       nameIsNull,
       BSTR                          reserved1,
       BSTR                          reserved2,
       boolean                       keepName,
       long *                        returnCode  )

ProcessInstance* CreateAndStartInstanceWithCnr(
       BSTR                          name,
       boolean                       nameIsNull,
       BSTR                          reserved1,
       BSTR                          reserved2,
       Container *                   input,
       boolean                       keepName,
       long *                        returnCode  )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateAndStartInstance(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjProcessInstanceHandle *   newInstance )
```

C++ language signatures

```
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 signature

```
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 message

```
<!-- ProcessTemplateCreateAndStart ================== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
          ( ProcTemplName,
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgInstName              (#PCDATA) >
<!ELEMENT KeepName                  (#PCDATA) >
              <!--  Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
```

XML message continued

```
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
          ( ProcessInstance
          | Exception ) >
<!ELEMENT ProcessInstance
          ( ProcInstID,
            ProcInstName,
            ProcInstParentName?,
            ProcInstTopLevelName,
            ProcInstDescription?,
            ProcInstState,
            LastStateChangeTime,
            LastModificationTime,
            ProcTemplID,
            ProcTemplName,
            Icon,
            Category? ) >
```

XML message continued

```
<!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) >
```

XML message continued

```
<!--==========================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ==========================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

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.

nameIsNull Input. Indicates whether a name is specified for 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 of calling this API call - see return codes below.

ProcessInstance*/ ProcessInstance
A pointer to the newly created and started process instance respectively 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist or is no longer valid.

FMC_ERROR_INVALID_CONTAINER(509)
The passed container is invalid for the process instance to be started; wrong schema or version.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.

FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.
FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.
FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.
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 - MQ Workflow XML request sent to MQ Workflow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateCreateAndStartInstance>
   <ProcTemplName>OnlineCreditRequest</ProcTemplName>
   <ProgInstName>Credit Request #658321<ProgInstName>
   <KeepName>true</KeepName>
   <ProcInstInputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
   </ProcInstInputData>
 </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

XML example - MQ Workflow response sent from MQ Workflow to the reply queue:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <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>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
 </ProcessTemplateCreateAndStartInstanceResponse>
</WfMessage>
```

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:

- 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 MQ 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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
ProcessInstance*  CreateInstance(
       BSTR                        name,
       boolean                     nameIsNull,
       BSTR                        reserved1,
       BSTR                        reserved2,
       boolean                     keepName,
       long *                      returnCode  )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateInstance(
       FmcjProcessTemplateHandle   hdlTemplate,
       char const *                name,
       char const *                reserved1,
       char const *                reserved2,
       bool                        keepName,
       FmcjProcessInstanceHandle * newInstance )
```

C++ language signature

```
APIRET CreateInstance(
       string const *        name,
       string const *        reserved1,
       string const *        reserved2,
       FmcjProcessInstance & newInstance,
       bool                  keepName = false ) const
```

Java signature

```
public abstract
ProcessInstance createInstance(
                String        name,
                String        reserved1,
                String        reserved2,
                boolean       keepName ) throws FmcException
```

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.

nameIsNull Input. Indicates whether a name is specified for 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.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance*/ ProcessInstance
A pointer to the newly created process instance respectively 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 exisiting process templates only.

There are no impacts on your transient representation of the process template; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process modeling authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
long Delete( boolean   pastVersions,
             boolean   currentVersion,
             boolean   futureVersions )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessTemplateDelete( FmcjProcessTemplateHandle hdlTemplate,
                           bool                      pastVersions,
                           bool                      currentVersion,
                           bool                      futureVersions )
```

C++ language signature

```
APIRET Delete( bool pastVersions  = true,
               bool currentVersion= true,
               bool futureVersions= true )
```

Java signature

```
public abstract
void   delete() throws FmcException

public abstract
void   delete2( boolean pastVersions,
                boolean currentVersion,
                boolean futureVersions ) throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 do 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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
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).

Note that the program execution agent(s) must have been started so that the activity implementations are executed.

This API call can be called synchronously and asynchronously. When called synchronously, the process instance should be short running 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, that is, contain all zeros (0x00).

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 MQ 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 respectively an empty container is returned.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
ProcessInstance* ExecuteProcessInstance(
       Container *                  output,
       BSTR                         name,
       boolean                      nameIsNull,
       BSTR                         reserved1,
       BSTR                         reserved2,
       boolean                      keepName,
       long *                       returnCode  )

ProcessInstance* ExecuteProcessInstanceWithCnr(
       Container *                  input,
       Container *                  output,
       BSTR                         name,
       boolean                      nameIsNull,
       BSTR                         reserved1,
       BSTR                         reserved2,
       boolean                      keepName,
       long *                       returnCode  )
```

C-language signatures

```
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++ language signatures

```
APIRET ExecuteProcessInstance(
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const

APIRET ExecuteProcessInstance(
       FmcjReadWriteContainer const &  input,
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const

APIRET ExecuteProcessInstanceAsync(
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false,
       FmcjCorrelID  *                 correlID = 0,
       string const *                  userContext = 0 )

APIRET ExecuteProcessInstanceAsync(
        FmcjReadWriteContainer const & input,
        string const *                 name = 0,
        string const *                 reserved1 = 0,
        string const *                 reserved2 = 0,
        bool                           keepName = false,
        FmcjCorrelID  *                correlID = 0,
        string const *                 userContext = 0 )
```

Java signature

```
public abstract
ProcessInstance executeProcessInstance(  ReadOnlyContainerHolder output,
                                         String                  name,
                                         String                  reserved1,
                                         String                  reserved2,
                                         Boolean                 keepName
) throws FmcException

public abstract
ProcessInstance executeProcessInstance2( ReadWriteContainer      input,
                                         ReadOnlyContainerHolder output,
                                         String                  name,
                                         String                  reserved1,
                                         String                  reserved2,
                                         Boolean                 keepName
) throws FmcException
```

XML message

```
<!-- 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? ) >
```

XML message continued

```
<!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) >
```

XML message continued

```
<!--============================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ============================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

Parameters

correlID	Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response.
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 executed.

nameIsNull Input. Indicates whether a name is specified for the process instance to be executed.

newInstance Input/Output. The executed process instance.

output Output. A pointer or an object to contain the output container of the 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.

userContext Input. A user-defined context which can be used for correlation.

Return type

APIRET The return code of calling 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.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist or is no longer valid.

FMC_ERROR_INVALID_CONTAINER(509)
The specified container is invalid for process instance execution; wrong schema or version.

FMC_ERROR_INVALID_CORRELATION_ID(506)
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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.

FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.

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 - MQ Workflow XML request sent to MQ Workflow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateExecute>
    <ProcTemplName>OnlineCreditRequest</ProcTemplName>
    <ProcInstName>Credit Request #658321</ProcInstName>
    <KeepName>true</KeepName>
    <ProcInstInputData>
      </CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage>
```

XML example - MQ Workflow XML response sent from MQ Workflow to the reply queue:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <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>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
    <ProcInstOutputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
```

```
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstOutputData>
  </ProcessTemplateExecuteResponse>
</WfMessageHeader>
```

InitialInContainer()

This API call retrieves the input container associated with the process template from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
long InitialInContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateInitialInContainer(
       FmcjProcessTemplateHandle      hdlTemplate,
       FmcjReadWriteContainerHandle * input )
```

C++ language signature

```
APIRET InitialInContainer( FmcjReadWriteContainer & input )
```

Java signature

```
public abstract
ReadWriteContainer initialInContainer() throws FmcException
```

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 respectively the input container of the process template to be set.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ProgramTemplate()

This API call retrieves the program template identified by the passed name from the MQ 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, then they are initial containers. Such, 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 note

- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
ProgramTemplate * ProgramTemplate( BSTR programName, long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateProgramTemplate(
       FmcjProcessTemplateHandle   hdlTemplate,
       char const *                programName,
       FmcjProgramTemplateHandle * program )
```

C++ language signature

```
APIRET ProgramTemplate( string const &         programName,
                        FmcjProgramTemplate &  program    ) const
```

Java signature

```
public abstract
ProgramTemplate  programTemplate( String programName )
throws FmcException
```

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.

returnCode Input/Output. The result of calling this method - see below.

Return type

APIRET/long* The result of calling this API call - see return codes below.

ProgramTemplate/ProgramTemplate*
The program template respectively 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 MQ Workflow 3.2.1 has been installed.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the process template from the MQ Workflow execution server (action call).

All information about the process template - primary and secondary - is retrieved.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX signature

```
long Refresh()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessTemplateRefresh( FmcjProcessTemplateHandle hdlTemplate )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

Parameters

hdlTemplate Input. The handle of the process template object to be refreshed.

Return type

long/ APIRET The result of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 46. Process template list 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 the C++ language, FmcjProcessTemplateList is thus a subclass of the FmcjPersistentList class and inherits all properties and methods. In the Java language, ProcessTemplateList is thus a subclass of the PersistentList class and inherits all properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all functions are explicitly defined on ProcessTemplateList. Note, however, that they are described as PersistentList actions.

The following sections describe the actions which can be applied on a process template list. See “ProcessTemplateList” on page 318 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 MQ 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

In C and C++, 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 call this API call.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplateList

ActiveX signature

```
long QueryProcessTemplates()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateListQueryProcessTemplates(
       FmcjProcessTemplateListHandle     hdlList,
       FmcjProcessTemplateVectorHandle * templates )
```

C++ language signature

```
APIRET QueryProcessTemplates(
       vector<FmcjProcessTemplate> & templates ) const;
```

Java signature

```
public abstract
ProcessTemplate[] queryProcessTemplates() throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query worklists (C-language)" on page 771
- For a C++ example see "Query worklists (C++)" on page 774

Chapter 47. Program template actions

An FmcjProgramTemplate or 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 wherin it is contained. This means that it is versioned via the containing process template.

The following sections describe the actions which can be applied on a program template. See "ProgramTemplate" on page 322 for a complete list of API calls.

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 called synchronously and asynchronously. When called synchronously, the program should be short running 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, that is, contain all zeros (0x00).

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 accessed by the program but none is provided, then 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 calling 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.

Specification of a priority influences OS/390 Workload management. The priority must be a value between 0 and 999.

Notes:

1. Passthrough() cannot be called from a program executed on the PES.
2. The output container is an input/output parameter. This means for Java, that it is passed as an input parameter and that it is returned as the return value of the execute method; the input parameter is not changed.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be logged on

Required connection

MQ Workflow program execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProgramTemplate
COBOL	fmcjvars.cpy, fmcjperf.cpy

ActiveX signature

```
long Execute()

long ExecuteWithOptions( Container *  input,
                         Container *  output
                         long         priority )
```

C-language signatures

```
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++ language signatures

```
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 signatures

```
public abstract
ReadWriteContainer execute() throws FmcException

public abstract
ReadWriteContainer execute2( ReadWriteContainer input,
                             ReadWriteContainer output,
                             long               priority )
throws FmcException
```

Parameters

correlID Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response.

hdlTemplate Input. The handle of the program template object to be executed.

input Input. The input container of the program.

output Input/Output. The output container of the program.

priority Input. The priority of the program to be executed.

userContext 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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_EXIT_ERROR(32204)

A program execution server exit reported an error.

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 or version 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_SPECIFICATION(816)
The specified priority must be in the range 0<=priority<=999.

FMC_ERROR_INVALID_USER_CONTEXT(819)
The specified user context is longer than 254 characters.

FMC_ERROR_LOCAL_USER_REQUIRED(32203)
The program must be executed by a local user.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

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_NOT_AUTHORIZED(32202)
The user is not authorized to execute 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 MQ 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_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.

FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.

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.

Chapter 48. Service actions

An FmcjService or Service object represents the common aspects of MQ Workflow service objects.

In the C++ language, FmcjService is the superclass of the FmcjExecutionService class and provides for all common properties and methods. In the Java language, Service is thus a superclass of the ExecutionService class and provides for all common properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all methods are explicitly defined on ExecutionService. Note, however, that they are described here as Service actions.

The following sections describe the actions which can be applied on a service. See "Service" on page 328 for a complete list of API calls.

Refresh()

This API call refreshes the logon status from the server (action call).

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service

ActiveX signature

```
long ExecutionService.Refresh()
```

C-language signature

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceRefresh( FmcjServiceHandle  service )

#define FmcjExecutionServiceRefresh FmcjServiceRefresh
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void refresh() throws FmcException
```

Parameters

service Input. A handle to the service object representing the session with an MQ Workflow server.

Return type

APIRET/long The return code of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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.

Note: If you intend to work in a multi-platform environment or switch between codepages, 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 codepages.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service

ActiveX signature

```
long ExecutionService.SetPassword( BSTR newPassword )
```

C-language signature

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceSetPassword( FmcjServiceHandle service,
                               char const *      newPassword )

#define FmcjExecutionServiceSetPassword FmcjServiceSetPassword
```

C++ language signature

```
APIRET SetPassword( string const & newPassword ) const
```

Java signature

```
public abstract
void   setPassword( String newPassword ) throws FmcException
```

Parameters

newPassword	Input. The new password to be used.
service	Input. A handle to the service object representing the session with an MQ Workflow server.

Return type

long/ APIRET	The return code of calling 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 does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_PASSWORD(12)
The password does not comply with the MQ 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

UserSettings()

This API call returns all settings of the logged on user (action call).

An empty object respectivly a null pointer is returned if no user has logged on yet via this service object.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service

ActiveX signature

```
IDispatch* ExecutionService.UserSettings( long * returnCode )
```

C-language signature

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceUserSettings( FmcjServiceHandle  service,
                                FmcjPersonHandle * user )

#define FmcjExecutionServiceUserSettings FmcjServiceUserSettings
```

C++ language signature

```
APIRET UserSettings( FmcjPerson & user ) const
```

Java signature

```
public abstract
Person userSettings() throws FmcException
```

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 MQ Workflow server.

user Input/Output. The person object to contain respectively the address of the person handle to point to the settings of the logged on user.

Return type

APIRET The return code of calling 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)

A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)

Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)

The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)

Timeout has occurred.

Chapter 49. Work item actions

An FmcjWorkitem or 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 the C++ language, FmcjWorkitem is thus a subclass of the FmcjItem class and inherits all properties and methods. In the Java language, WorkItem is thus a subclass of the Item class and inherits all properties and methods. Similarly, in the C-language, 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. In ActiveX, inheritance is not supported so that all functions are explicitly defined on Workitem. Note, however, that common methods are described as Item actions.

A work item is uniquely identified by its object identifier.

The following diagrams provide an overview on the possible work item states and the actions which are allowed in those states, provided that the appropriate authority has been granted and that more specific requirements stated in the API calls descriptions have been fulfilled. Note that the actions and possible states are dependent on the process instance state, the work item is a part of.

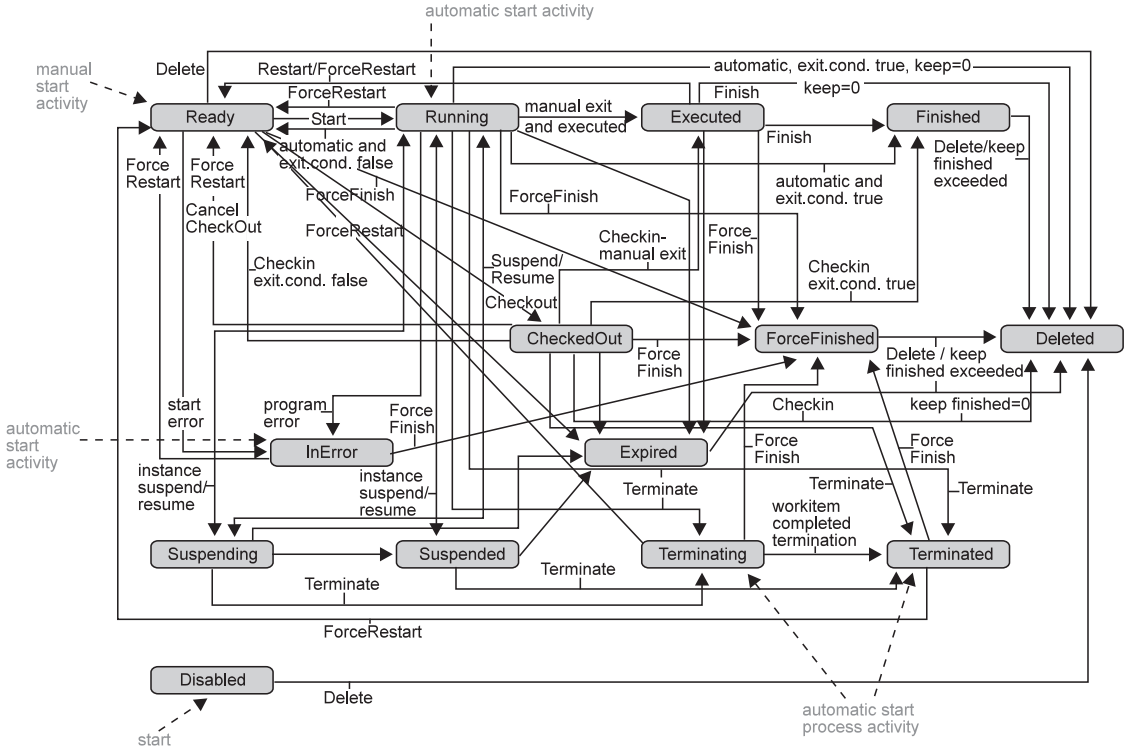


Figure 24. Work item states - process instance state running

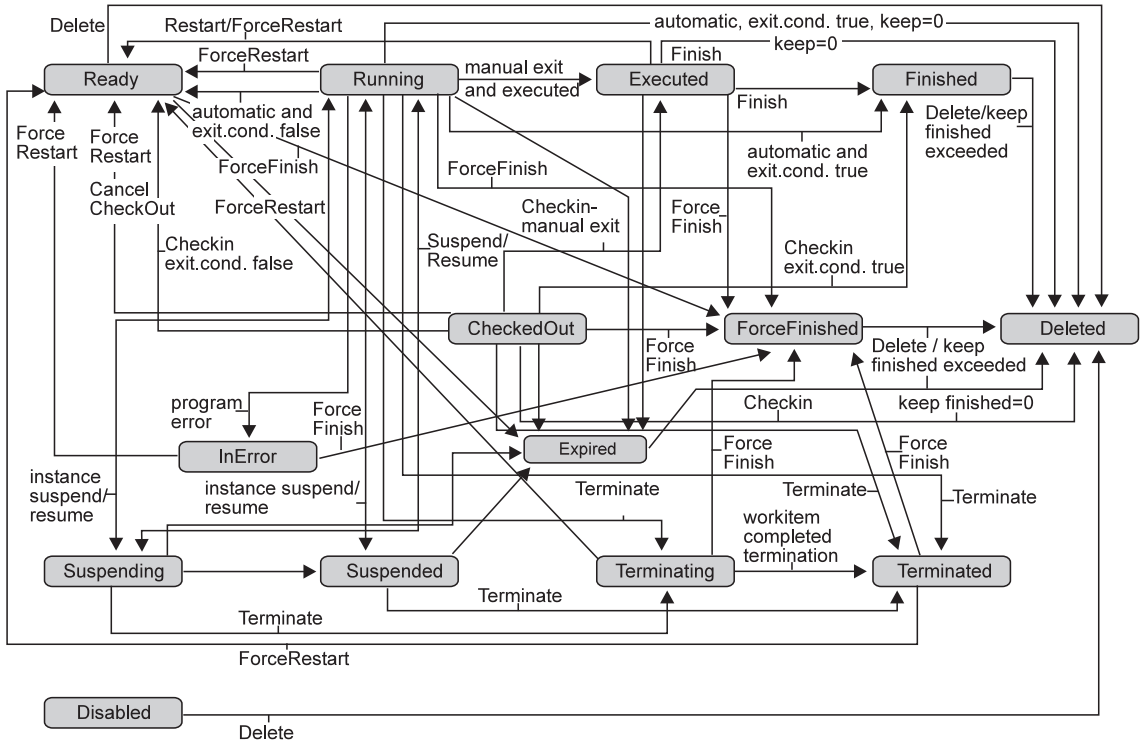


Figure 25. Work item states - process instance state suspending or suspended

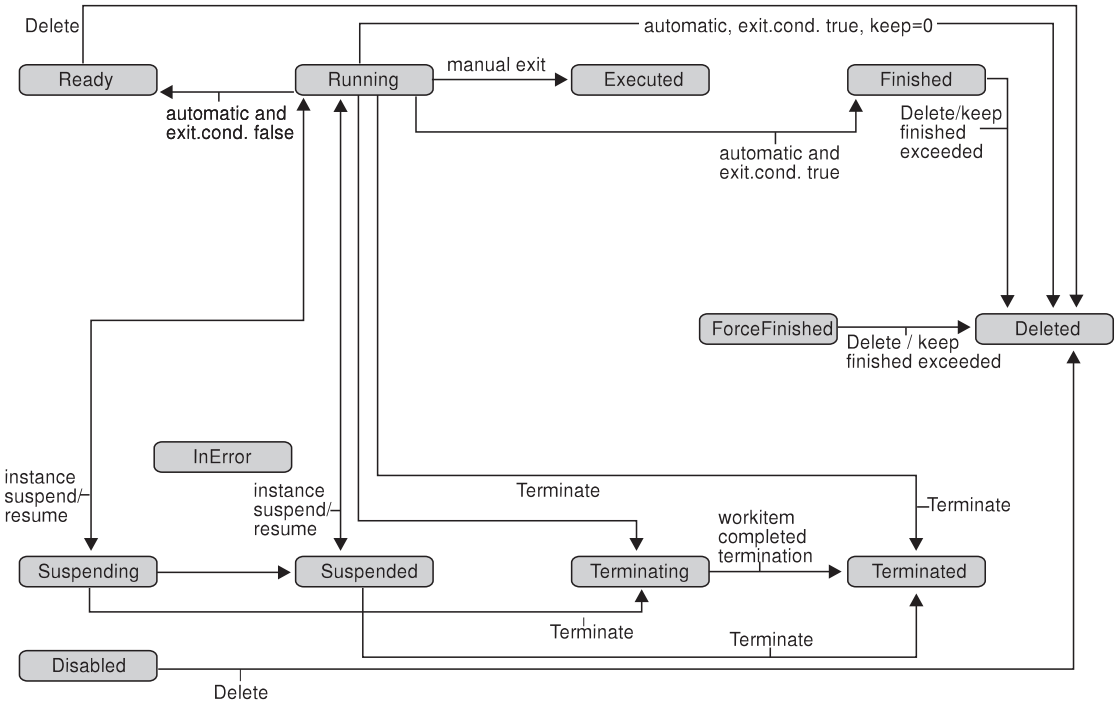


Figure 26. Work item states - process instance state terminating or terminated

The following sections describe the actions which can be applied on a work item. See "Workitem" on page 332 for a complete list of API calls.

ActivityInstance()

This API call retrieves the activity instance the work item is associated to from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

In C++, when the activity 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 activity instance handle already points to some object.

Usage note

- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
ActivityInstance* ActivityInstance( long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemActivityInstance(
       FmcjWorkitemHandle              hdlWorkitem,
       FmcjActivityInstanceHandle *    instance )
```

C++ language signature

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java signature

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

Parameters

hdlWorkitem	Input. The handle of the work item object to be queried.
instance	Input/Output. The activity 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 of calling this API call - see return codes below.

ActivityInstance*/ ActivityInstance
A pointer to the activity instance or the activity instance the work item is associated to.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 or activity instance does not exist. The activity instance may not exist when the transient work item object has been recreated from its OID and when it is not a work item but a process 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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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.

Note that all sibling work items set into the *Disabled* state by the previous Checkout() request are also reset into the *Ready* state.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long CancelCheckOut()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemCancelCheckOut( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET CancelCheckOut()
```

Java signature

```
public abstract
void  cancelCheckOut() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
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).

The work item must be in the *CheckedOut* state and the associated process instance must be in the *Running* or *Suspending* state.

Checking in a work item tells MQ Workflow that user processing has finished and workflow processing under the control of MQ Workflow can continue. The return code of the user processing and, optionally, the output container values are passed back to MQ 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. The return code is automatically set into the _RC data member of the output container if this field has not been set explicitly.

When an output container is specified, then that container must be a valid container for the work item, that is, it must contain the correct schema and

version definitions. In other words, it must be the (updated) output container retrieved with the *CheckOut()* request or the output container retrieved for the work item, and so on.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long CheckIn( Container * output, long returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemCheckIn( FmcjWorkitemHandle          hdlWorkitem,
                     FmcjReadWriteContainerHandle output,
                     long                        returnCode )
```

C++ language signature

```
APIRET CheckIn( FmcjReadWriteContainer const * output,
                long                           returnCode )
```

Java signature

```
public abstract
void   checkIn( ReadWriteContainer output,
                int                returnCode ) throws FmcException
```

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 of calling 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

FMC_ERROR_INVALID_CONTAINER(509)

The specified output container is invalid; wrong schema or version.

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 transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 implemented as a program and be in the *Ready* state. The associated process instance must be in the *Running* state.

Checkout then means that processing is not done by MQ Workflow's inherent program-invocation mechanism. MQ Workflow assumes that processing is done by user-specific means and changes the state of the work item to *CheckedOut*.

The caller can request program definitions for specific operating system platforms. The following enumeration types can be used to specify the requested program data.

ActiveX	WorkitemProgramRetrieval
C-language	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.

ActiveX	WIProgramRetrieval_NotSet
C-language	Fmc_WS_NotSet
C++	FmcjWorkitem::NotSet
JAVA	ProgramRetrieval.NOT_SET

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.

ActiveX	WIProgramRetrieval_CommonDataOnly
C-language	Fmc_WS_CommonDataOnly
C++	FmcjWorkitem::CommonDataOnly
JAVA	ProgramRetrieval.COMMON_DATA_ONLY

SpecifiedDefinitions
returns the program definition for the specified platform. A platform must be specified.

ActiveX	WIProgramRetrieval_SpecifiedDefinitions
C-language	Fmc_WS_SpecifiedDefinitions
C++	FmcjWorkitem::SpecifiedDefinitions
JAVA	ProgramRetrieval.SPECIFIED_DEFINITIONS

AllDefinitions
returns all available program definitions. Any platform specification is ignored.

ActiveX	WIProgramRetrieval_AllDefinitions
C-language	Fmc_WS_AllDefinitions
C++	FmcjWorkitem::AllDefinitions
JAVA	ProgramRetrieval.ALL_DEFINITIONS

The following enumeration types can be used to specify the platform for which program definitions are to be retrieved.

ActiveX	ImplementationDataBasis
C-language	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.

ActiveX	Basis_NotSpecified
C-language	Fmc_DP_NotSet
C++	FmcjImplementationData::NotSpecified
JAVA	Basis.NOT_SPECIFIED

OS2 indicates that the program definition for the OS/2 platform is requested.

ActiveX	Basis_OS2
C-language	Fmc_DP_OS2
C++	FmcjImplementationData::OS2
JAVA	Basis.OS2

AIX indicates that the program definition for the AIX platform is requested.

ActiveX	Basis_AIX
C-language	Fmc_DP_AIX
C++	FmcjImplementationData::AIX
JAVA	Basis.AIX

HPUX indicates that the program definition for the HP-UX platform is requested.

ActiveX	Basis_HPUX
C-language	Fmc_DP_HPUX
C++	FmcjImplementationData::HPUX
JAVA	Basis.HPUX

Windows95 indicates that the program definition for the Windows 95, 98, or Me platform is requested.

ActiveX	Basis_Windows95
C-language	Fmc_DP_Windows95
C++	FmcjImplementationData::Windows95
JAVA	Basis.WINDOWS_95

WindowsNT indicates that the program definition for the Windows NT or Windows 2000 platform is requested.

ActiveX	Basis_WindowsNT
C-language	Fmc_DP_WindowsNT

	C++	FmcjImplementationData::WindowsNT
	JAVA	Basis.WINDOWS_NT
OS390	indicates that the program definition for the OS/390(R) platform is requested.	
	ActiveX	Basis_OS390
	C-language	Fmc_DP_OS390
	C++	FmcjImplementationData::OS390
	JAVA	Basis.OS390
Solaris	indicates that the program definition for the Solaris platform is requested.	
	ActiveX	Basis_Solaris
	C-language	Fmc_DP_Solaris
	C++	FmcjImplementationData::Solaris
	JAVA	Basis.Solaris

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
ProgramData CheckOut( WorkitemProgramRetrieval requestedData,
                      ImplementationDataBasis  platform,
                      long *                   returnCode   )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemCheckOut( FmcjWorkitemHandle                hdlWorkitem,
                      enum FmcjWorkitemProgramRetrieval requestedData,
                      enum FmcjImplementationDataBasis  platform,
                      FmcjProgramDataHandle *           programData  )
```

C++ language signature

```
APIRET CheckOut( ProgramRetrieval                requestedData,
                 FmcjImplementationData::Basis   platform,
                 FmcjProgramData &               programData  )
```

Java signature

```
public abstract
ReadOnlyContainer checkOut() throws FmcException

public abstract
ProgramData       checkOut2(
                  ProgramRetrieval  requestedData,
                  Basis             platform     ) throws FmcException
```

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 respectively 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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

FMC_ERROR_INVALID_SPECIFICATION(816)
Invalid combination of checkout parameters.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_PROGRAM_NOT_DEFINED(1012)
No program defined for the requested platform.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long Finish()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemFinish( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Finish()
```

Java signature

```
public abstract
void   finish() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceFinish()

This API call ends the execution of a work item which is known to have completed (action call).

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*.

Optionally, an output container can be specified to denote the result of processing. If none is specified, the output container available at the execution server is taken. For example, the output container defined with initial values.

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 note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long ForceFinish()

long ForceFinishWithContainer( Container * outputContainer )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemForceFinish( FmcjWorkitemHandle hdlWorkitem )

APIRET FMC_APIENTRY
FmcjWorkitemForceFinishWithContainer( FmcjWorkitemHandle   hdlWorkitem,
                                      FmcjContainerHandle  outputContainer )
```

C++ language signature

```
APIRET ForceFinish()

APIRET ForceFinish( FmcjContainer const & outputContainer )
```

Java signature

```
public abstract
void   forceFinish() throws FmcException

public abstract
void   forceFinish2( Container outputContainer ) throws FmcException
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

outputContainer
Input. The output container to be set as the result of the call; can be a read/write or read-only container.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceRestart()

This API call forces MQ Workflow to enable the restart of a work item (action call).

A work item implemented by a program must be in states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in states *Ready*, *Executed*, *InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

Optionally, an input container can be specified to denote the input to be used for restarting the work item. If none is specified, the input container available at the execution server is taken.

The work item is then reset into the *Ready* state. Note that automatic activity instances must now be started manually.

Usage note
- See "Action API calls" on page 150 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

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX signature

```
long ForceRestart()

long ForceRestartWithContainer( Container * inputContainer )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemForceRestart(
                                FmcjWorkitemHandle hdlWorkitem )

APIRET FMC_APIENTRY FmcjWorkitemForceRestartWithContainer(
                                FmcjWorkitemHandle  hdlWorkitem,
                                FmcjContainerHandle inputContainer )
```

C++ language signature

```
APIRET ForceRestart()

APIRET ForceRestart( FmcjContainer const & inputContainer )
```

Java signature

```
public abstract
void   forceRestart() throws FmcException

public abstract
void   forceRestart2( Container inputContainer ) throws FmcException
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

inputContainer
Input. The input container to be used when restarting the work item.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the work item from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

One of:

- Be the work item owner
- Work item authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long InContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemInContainer( FmcjWorkitemHandle            hdlWorkitem,
                         FmcjReadOnlyContainerHandle * input )
```

C++ language signature

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java signature

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

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.

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the work item from the MQ Workflow execution server (action call).

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

One of:
- Be the work item owner
- Work item authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long OutContainer( Container * output )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemOutContainer( FmcjWorkitemHandle             hdlWorkitem,
                          FmcjReadWriteContainerHandle * output )
```

C++ language signature

```
APIRET OutContainer( FmcjReadWriteContainer & output ) const
```

Java signature

```
public abstract
ReadWriteContainer outContainer() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.
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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This API call asks MQ 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 note
- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long Restart()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemRestart( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Restart()
```

Java signature

```
public abstract
void   restart() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 agent associated to the logged-on user.

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 note

- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long Start()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemStart( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Start()
```

Java signature

```
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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

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 to the activity instance the work item is derived from. It is then started on the program execution agent associated to the logged-on user.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX signature

```
long StartTool( BSTR toolName )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemStartTool( FmcjWorkitemHandle hdlWorkitem,
                       char const *       toolName   )
```

C++ language signature

```
APIRET StartTool( string const & toolName ) const
```

Java signature

```
public abstract
void   startTool( String toolName ) throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.
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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.
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 MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
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*.

When the work item is implemented by a program and processed under the control of a program execution agent or user-defined program execution server, a message is sent to inform about the termination request. The program execution agent tries to kill fenced activity implementations.

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.

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. Navigation can be explicitly continued by a user with process administration rights, that is, ForceFinish() or ForceRestart() repair actions can be called.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX signature

```
long Terminate()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemTerminate( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does 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_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 50. Work listactions

An FmcjWorklist or 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.

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 the C++ language, 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 the C-language, 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. In ActiveX, inheritance is not supported so that all functions are explicitly defined on Worklist. Note, however, that common methods are described as PersistentList actions.

The following sections describe the actions which can be applied on a worklist. See “Worklist” on page 335 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 MQ 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 and C++, 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 call this API call. This means that you should set the handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

Usage note
- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkList

ActiveX signature

```
long QueryActivityInstanceNotifs()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorklistQueryActivityInstanceNotifications(
       FmcjWorklistHandle                              hdlList,
       FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++ language signature

```
APIRET QueryActivityInstanceNotifications(
       vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java signature

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications()
throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query work items from a worklist (ActiveX)" on page 791
- For a C-language example see "Query work items from a worklist (C-language)" on page 792
- For a C++ example see "Query work items from a worklist (C++)" on page 794
- For a Java example see "Query work items from a worklist (Java)" on page 796

QueryItems()

This API call retrieves the primary information for all items characterized by the specified worklist from the MQ 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 and C++, 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 call this API call. This means that you should set the handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

Usage note
- See “Action API calls” on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	not applicable
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorklistQueryItems( FmcjWorklistHandle     hdlList,
                        FmcjItemVectorHandle * items )
```

C++ language signature

```
APIRET QueryItems( vector<FmcjItem> & items ) const
```

Java signature

```
public abstract Item[] queryItems() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query work items from a worklist (C-language)" on page 792
- For a C++ example see "Query work items from a worklist (C++)" on page 794
- For a Java example see "Query work items from a worklist (Java)" on page 796

QueryProcessInstanceNotifications()

This API call retrieves the primary information for all process instance notifications characterized by the specified worklist from the MQ 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 and C++, 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 call this API call. This means that you should set the handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

Usage note

- See "Action API calls" on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList

ActiveX signature

```
long QueryProcessInstanceNotifs()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorklistQueryProcessInstanceNotifications(
       FmcjWorklistHandle                          hdlList,
       FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++ language signature

```
APIRET QueryProcessInstanceNotifications(
       vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java signature

```
public abstract
ProcessInstanceNotification[] queryProcessInstanceNotifications()
throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query work items from a worklist (ActiveX)" on page 791

- For a C-language example see “Query work items from a worklist (C-language)” on page 792
- For a C++ example see “Query work items from a worklist (C++)” on page 794
- For a Java example see “Query work items from a worklist (Java)” on page 796

QueryWorkitems()

This API call retrieves the primary information for all work items characterized by the specified worklist from the MQ 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 and C++, 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 call this API call. This means that you should set the handle to 0 in the C-language respectively erase all elements of the vector in the C++ API.

Usage note
- See “Action API calls” on page 150 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList

ActiveX signature

```
long QueryWorkitems()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorklistQueryWorkitems(
       FmcjWorklistHandle         hdlList,
       FmcjWorkitemVectorHandle * workitems )
```

C++ language signature

```
APIRET QueryWorkitems( vector<FmcjWorkitem> & workitems ) const
```

Java signature

```
public abstract
WorkItem[] queryWorkItems() throws FmcException
```

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 does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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 does no longer exist.

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 MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query work items from a worklist (ActiveX)” on page 791
- For a C-language example see “Query work items from a worklist (C-language)” on page 792
- For a C++ example see “Query work items from a worklist (C++)” on page 794
- For a Java example see “Query work items from a worklist (Java)” on page 796

Part 8. Working with ActiveX controls

The following chapters describe the ActiveX controls.

Note: ActiveX signatures are provided in the Object Definition Language (ODL). For example, type *BSTR* is used for strings where the VisualBasic type is actually String.

Chapter 51. The ExecutionService Control

To access an ExecutionService in the ExecutionServiceArray, the ExecutionService Control must be connected to the Workflow Control. The connection is established by the ConnectGUI() method of the ExecutionService Control class. When the connection has been established, the user has full access to the ExecutionService shown within the tree view of the ExecutionService Control window.

Chapter 52. The list controls

To access, for example, the worklists in the WorklistArray, each worklist must be connected to a Worklist Control. This connection is established via the ConnectGUI() method of the Worklist Control class. When the connection has been established, the user has access to the Worklist object in the Worklist Control. The same mechanism is used to connect all other List types to be used within a GUI.

Chapter 53. The Monitor Control

The Monitor Control OCX is the ActiveX component that implements the Process Monitor GUI. To access a monitor, the Monitor Control must be connected to the Workflow Control. The connection is established by the ConnectGUI() method of the Monitor Control class. When the connection has been established, the user has full access to the instance monitor object.

The Monitor Control can be used to display one process model graph. A Refresh() method is provided in order to display changed activity instance states. The OCX is not intended to be used for different process models graphs. You must use separate Monitor OCX instances in order to view different process models.

Chapter 54. Typical scenario of ActiveX Control methods

If you want to look at a typical scenario to get, for example, the name of a specific worklist, the sequence of steps to establish the prerequisites is as follows:

1. The ExecutionService Control accessor method **ExecutionServiceArray** provides access to the local Control object ExecutionServiceArray.
2. The ExecutionServiceArray action method **Add** provides the ExecutionService index and the accessor method **GetAt** provides the transient ExecutionService object.
3. The ExecutionService action method **QueryWorklists** provides transient Worklist objects.
4. The ExecutionService accessor method **WorklistArray** provides access to the local Control object WorklistArray.
5. The WorklistArray accessor method **GetAt** provides access to a specific Worklist object with the WorklistArray.
6. The Worklist accessor method **Name** provides the name of the specific Worklist.

Chapter 55. MQWorkflowCtrl

Methods

The MQWorkflow Control supports the following methods:

ConfigurationID

Returns the configuration ID to be used for profile access.

Signature

```
BSTR ConfigurationID()
```

Return type

BSTR The configuration ID.

Connect

This method initializes MQ Workflow API processing for a particular thread.

Signature

```
long Connect()
```

Return type

long If processing completes successfully, FMC_OK is returned.

ContainerArray

Provides access to the ContainerArray object.

Signature

```
ContainerArray * ContainerArray()
```

Return type

ContainerArray*

A pointer to the ContainerArray object.

CurrentDateAndTime

Creates a new DateAndTime object initialized with the current date and time values.

Signature

```
DateAndTime * CurrentDateAndTime()
```

Return type
DateAndTime*
A pointer to the DateAndTime object.

DateAndTime

Creates a new (uninitialized) DateAndTime object.

Signature

```
DateAndTime * DateAndTime()
```

Return type
DateAndTime*
A pointer to the DateAndTime object.

Disconnect

This method uninitializes MQ Workflow API processing for a particular thread.

Signature

```
long Disconnect()
```

Return type
long If processing completes successfully, FMC_OK is returned.

ExecutionServiceArray

Provides access to the ExecutionServiceArray object.

Signature

```
ExecutionServiceArray * ExecutionServiceArray()
```

Return type
ExecutionServiceArray*
A pointer to the ExecutionServiceArray object.

NewActivityInstance

Creates an empty activity instance object.

Signature

```
ActivityInstance * NewActivityInstance()
```

Return type
ActivityInstance*
A pointer to the activity instance created.

NewActivityInstanceNotification

Creates an empty activity instance notification object.

Signature

```
ActivityInstanceNotification * NewActivityInstanceNotification()
```

Return type
ActivityInstanceNotification*
A pointer to the activity instance notification created.

NewContainer

Creates an empty container object.

Signature

```
Container * NewContainer()
```

Return type
Container* A pointer to the container created.

NewExecutionService

Creates an empty execution service object.

Signature

```
ExecutionService * NewExecutionService()
```

Return type
ExecutionService*
A pointer to the execution service created.

NewInstanceMonitor

Creates an empty instance monitor object.

Signature

```
InstanceMonitor * NewInstanceMonitor()
```

Return type
InstanceMonitor*
A pointer to the instance monitor created.

NewPerson

Creates an empty person object.

Signature

```
Person * NewPerson()
```

Return type
Person* A pointer to the person created.

NewProcessInstance

Creates an empty process instance object.

Signature

```
ProcessInstance * NewProcessInstance()
```

Return type
ProcessInstance*
A pointer to the process instance object created.

NewProcessInstanceList

Creates an empty process instance list object.

Signature

```
ProcessInstanceList * NewProcessInstanceList()
```

Return type
ProcessInstanceList*
A pointer to the process instance list created.

NewProcessInstanceNotification

Creates an empty process instance notification object.

Signature

```
ProcessInstanceNotification * NewProcessInstanceNotification()
```

Return type
ProcessInstanceNotification*
A pointer to the process instance notification object created.

NewProcessTemplate

Creates an empty process template object.

Signature

```
ProcessTemplate * NewProcessTemplate()
```

Return type
ProcessTemplate*
A pointer to the process template object created.

NewProcessTemplateList

Creates an empty process template list object.

Signature

```
ProcessTemplateList * NewProcessTemplateList()
```

Return type
ProcessTemplateList*
A pointer to the process template list created.

NewProgramData

Creates an empty program data object.

Signature

```
ProgramData * NewProgramData()
```

Return type
ProgramData* A pointer to the program data created.

NewProgramTemplate

Creates an empty program template object.

Signature

```
ProgramTemplate * NewProgramTemplate()
```

Return type
ProgramTemplate*
A pointer to the program template created.

NewWorkitem

Creates an empty work item object.

Signature

```
Workitem * NewWorkitem()
```

Return type
Workitem* A pointer to the work item object created.

NewWorklist

Creates an empty worklist object.

Signature

```
Worklist * NewWorklist()
```

Return type
Worklist* A pointer to the worklist created.

ProgramID

Returns the program ID by which an activity implementation is known to the program execution agent.

Signature

```
BSTR ProgramID()
```

Return type
BSTR The program ID.

RemoteUserID

Returns the user ID for whom the original activity implementation has been started by the program execution agent.

Signature

```
BSTR RemoteUserID()
```

Return type

BSTR The user ID.

SetConfigurationID

This method sets the configuration ID to be used for profile access.

Signature

```
long SetConfigurationID( BSTR configID )
```

Parameters

configID Input. The configuration ID to be set; must be a configuration already defined.

Return type

long If processing completes successfully, FMC_OK is returned.

StringArray

Provides access to the StringArray object.

Signature

```
StringArray * StringArray()
```

Return type

StringArray* A pointer to the StringArray object.

UserID

Returns the user ID for whom an activity implementation has been started by the program execution agent.

Signature

```
BSTR UserID()
```

Return type

BSTR	The user ID.

Chapter 56. ContainerCtrl

Properties

The Container Control has the following property, which can be modified:
Visible

Methods

The Container Control supports the following methods:

Container

This method provides access to an MQ Workflow Container object.

Signature

```
Container * Container()
```

Return type
Container* The pointer to the Container object.

ProgramID

Returns the program ID by which an activity implementation is known to the program execution agent.

Signature

```
BSTR ProgramID()
```

Return type
BSTR The program ID.

RemoteUserID

Returns the user ID for whom the original activity implementation has been started by the program execution agent.

Signature

```
BSTR RemoteUserID()
```

Return type

BSTR The user ID.

UserID

Returns the user ID for whom an activity implementation has been started by the program execution agent.

Signature

```
BSTR UserID()
```

Return type
BSTR The user ID.

Events

The Container Control triggers the following events:

Error

Occurs only as the result of an error that takes place when no Visual Basic code is being executed.

Signature

```
void Error( short    number,
            BSTR *   description,
            SCODE    scode,
            BSTR     source,
            BSTR     helpFile,
            long     helpContext,
            boolean *cancel     )
```

Parameters

number	Output. The error number as an integer.
description	Output. The description of the error.
scode	Output. The scode error as a long integer.
source	Output. The source of the error.
helpFile	Output. The name of the help file.
helpContext	Output. The help context identifier.
cancel	Input. If set to *True*, the message is not sent to the next layer.

Chapter 57. Methods supported by all GUI controls

Following methods are supported by the ExecutionService Control, the ProcessTemplateList Control, the ProcessInstanceList Control, the Worklist Control, and the Monitor Control.

AboutBox

Opens the about box of the Control.

Signature

```
void AboutBox()
```

ReadUserSettings

Restores user settings from the registry.

This restores values for the columns to be shown in the report view.

Signature

```
boolean ReadUserSettings( BSTR name )
```

Parameters

name Input. The string denoting the name of the registry key, the values of which are restored.

Return type

boolean If processing completes successfully, *True* is returned.

RemoveGUI

Removes the connection between the ExecutionService Control and the MQWorkflow Control.

Signature

```
long RemoveGUI()
```

Return type

long If processing completes successfully, FMC_OK is returned.

SetHelpFile

Defines the path and filename of the help file to be used in Runtime mode.

Signature

```
void SetHelpFile( BSTR name )
```

Parameters

name Input. The string denoting the fully qualified pathname of the help file.

ShowContextMenu

Enables or disables the context menu.

Signature

```
void ShowContextMenu( boolean toggle )
```

Parameters

toggle Input. If *True* is specified, a right mouse click shows the context menu. The specification of *False* disables the context menu.

WriteUserSettings

Saves the user settings within the registry.

This saves the current columns sizes as well as the columns that are selected to be shown in report view.

Signature

```
boolean WriteUserSettings( BSTR name )
```

Parameters

name Input. The string denoting the name of the registry key where the current values are stored.

Return type
boolean If processing completes successfully, *True* is returned.

Chapter 58. Methods supported by all list controls

Following methods are supported by all list controls, the ProcessTemplateList Control, the ProcessInstanceList Control, and the Worklist Control.

ConnectGUI

Connects the specified process template list, the specified process instance list, or the specified worklist to the GUI.

Signature

```
long ConnectGUI( IDispatch * list, IDispatch * es, IDispatch * wfc )
```

Parameters

es	Input. The pointer to the execution service object.
list	Input. The pointer to the list object.
wfc	Input. The pointer to the MQWorkflowCtrl object.

Return type

long	If processing completes successfully, FMC_OK is returned.

ContextMenuDelete

Calls the context menu item 'Delete'.

Signature

```
long ContextMenuDelete()
```

Return type

long	Return code of *FmcjProcessTemplate::Delete(), or FmcjProcessInstanceListDelete(), or FmcjWorklist::Delete()*.

ContextMenuListProperties

Calls the context menu item 'Properties'.

This method shows a dialog which provides property information of the process template list, process instance list, or worklist.

Signature

```
void ContextMenuListProperties()
```

ContextMenuListSettings

Calls the context menu item 'Processlist settings'.

This method shows a dialog which provides information on name, type, filter, and sort criteria of the process template list, process instance list, or worklist.

Signature

```
void ContextMenuListSettings ()
```

ContextMenuListRefresh

Calls the context menu item 'Refresh ProcessTemplateList', or 'Refresh ProcessInstanceList', or 'Refresh Worklist'.

Signature

```
long ContextMenuListRefresh()
```

Return type

long If processing completes successfully, FMC_OK is returned.

ContextMenuProperties

Calls the context menu item 'Show Properties'.

This method shows the properties of the selected objects within the ProcessTemplateList, the ProcessInstanceList, or the Worklist Control.

Signature

```
void ContextMenuProperties()
```

ContextMenuViewIcon

Displays the ProcessTemplateList, or ProcessInstanceList, or Worklist Control grid in icon mode.

Signature

```
void ContextMenuViewIcon()
```

ContextMenuViewList

Displays the ProcessTemplateList, the ProcessInstanceList, or Worklist Control grid in list mode.

Signature

```
void ContextMenuViewList()
```

ContextMenuViewReport

Displays the ProcessTemplateList, the ProcessInstanceList, or Worklist Control grid in report mode.

Signature

```
void ContextMenuViewReport()
```

ContextMenuViewSmallIcon

Displays the ProcessTemplateList, the ProcessInstanceList, or Worklist Control grid in small icon mode.

Signature

```
void ContextMenuViewSmallIcon()
```

FindFirst

Returns the first object of the process template list, the process instance list, or the worklist.

Signature for process template or process instance lists

```
IDispatch * FindFirst( long * index )
```

Signature for worklists

```
IDispatch * FindFirst( long * objecttype, long * index )
```

Parameters

index Output. A pointer to a long field where the returned index is to be stored. If no selected object is found, 1 is returned.

objecttype Output. An indication whether an actual work item, an activity instance notification, or a process instance notification is returned.

Return type

IDispatch* The object pointer. NULL (0) is returned if the selected object is not found.

FindNext

Returns the next object of the process template list, the process instance list, or the worklist. FindFirst() must have been called before.

Signature for process template or process instance lists

```
IDispatch * FindNext( long * index  )
```

Signature for worklists

```
IDispatch * FindNext( long * objecttype, long * index )
```

Parameters

index Output. A pointer to a long field where the returned index is to be stored. If no selected object is found, 1 is returned.

objecttype Output. An indication whether an actual work item, an activity instance notification, or a process instance notification is returned.

Return type

IDispatch* The object pointer. NULL (0) is returned if the selected object is not found.

Return Value

IDispatch* The ProcessTemplate object pointer. 0 (null) is returned if no selected object is found.

Parameters

index Output. A pointer to a long field where the returned index is stored. If no selected object is found, 1 is returned.

GetItemAt

Returns the object of the process template list, the process instance list, or the worklist at the given index.

Signature

```
IDispatch * GetItemAt( long index )
```

Parameters

index Input. The array index of the object to be returned.

Return type

IDispatch* The object pointer.

GetItemCount

Returns the number of objects in the process template list array, the process instance list array, or the worklist array.

Signature

```
long GetItemCount()
```

Return type

long The number of objects in the array.

Chapter 59. Events triggered by all GUI controls

Following events are triggered by the ExecutionService Control, the ProcessTemplateList Control, the ProcessInstanceList Control, the Worklist Control, and the Monitor Control.

Click

Occurs when the user presses a mouse button over this control.

Signature

```
void Click()
```

DblClick

Occurs when the user presses and releases a mouse button and then presses and releases it again over this control.

Signature

```
void DblClick()
```

KeyPress

Occurs when the user presses and releases an ANSI key.

Signature

```
void KeyPress( short  keyAscii )
```

Parameters

keyAscii Output. An integer that returns a standard numeric ANSI keycode.

Chapter 60. Events triggered by all non-monitor GUI controls

Following events are triggered by the ExecutionService Control, the ProcessTemplateList Control, the ProcessInstanceList Control, and the Worklist Control.

Error

Occurs only as the result of an error that takes place when no Visual Basic code is being executed.

Signature

```
void Error( short    number,
            BSTR *   description,
            SCODE    scode,
            BSTR     source,
            BSTR     helpFile,
            long     helpContext,
            boolean  cancel     )
```

Parameters

number	Output. The error number as an integer.
description	Output. The description of the error.
scode	Output. The scode error as a long integer.
source	Output. The source of the error.
helpFile	Output. The name of the help file.
helpContext	Output. The help context identifier.
cancel	Input. If set to *True*, the message is not sent to the next layer.

KeyDown

Occurs when the user pressed a key while an object has this control as the focus.

Signature

```
void KeyDown( short * keyCode, short shift )
```

Parameters

keyCode Output. A key code, such as vbKeyF1 (the F1 key) or vbKeyHome (the HOME key).

shift Output. An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

KeyUp

Occurs when the user releases a key while an object has this control as the focus.

Signature

```
void KeyUp( short * keyCode, short shift )
```

Parameters

keyCode Output. A key code, such as vbKeyF1 (the F1 key) or vbKeyHome (the HOME key).

shift Output. An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

MouseDown

Occurs when the user presses the mouse button over the ProcessInstanceList Control window.

Signature

```
void MouseDown( short           button ,
                short           shift ,
                OLE_XPOS_PIXELS x ,
                OLE_YPOS_PIXELS y     )
```

Parameters

button Output. Returns an integer that identifies the button that was pressed.

shift Output. An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

x Output. Returns an integer that specifies the current location (abscissa) of the mouse pointer.

y Output. Returns an integer that specifies the current location (ordinate) of the mouse pointer.

MouseMove

Occurs when the user moves the mouse.

Signature

```
void MouseMove( short           button ,
                short           shift ,
                OLE_XPOS_PIXELS x ,
                OLE_YPOS_PIXELS y     )
```

Parameters

button Output. Returns an integer that identifies the button that was pressed.

shift Output. An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

x Output. Returns an integer that specifies the current location (abscissa) of the mouse pointer.

y Output. Returns an integer that specifies the current location (ordinate) of the mouse pointer.

MouseUp

Occurs when the user releases a mouse button over the ProcessInstanceList Control window.

Signature

```
void MouseUp(   short           button ,
                short           shift ,
                OLE_XPOS_PIXELS x ,
                OLE_YPOS_PIXELS y     )
```

Parameters

button Output. Returns an integer that identifies the button that was pressed.

shift Output. An integer that corresponds to the state of the SHIFT, CTRL, and ALT keys at the time of the event.

x Output. Returns an integer that specifies the current location (abscissa) of the mouse pointer.

y Output. Returns an integer that specifies the current location (ordinate) of the mouse pointer.

Chapter 61. Events triggered by all list controls

Following events are triggered by all lists, the ProcessTemplateList Control, the ProcessInstanceList Control, and the Worklist Control.

ViewChanged

Occurs when the list grid view has changed.

Signature

```
void ViewChanged()
```

Chapter 62. ExecutionServiceCtrl

Properties

In design mode, the properties for the ExecutionService Control can be viewed by placing the mouse pointer within the Control's grid area, clicking the right mouse button and selecting **Properties**.

The property dialog contains two tabs: **Fonts** and **Pictures**.

All tabs are available in design mode and run mode.

The **Fonts** tab displays a standard font dialog. The font, its size, style, and the effects used to display the text in the ExecutionService Control can be manipulated here.

The **Pictures** tab provides the ability to associate icons with the various elements of the ExecutionService Control's display.

To view the icon currently associated with a given item, select the item from the **Property Name** drop-down list. The associated icon is displayed in the **Preview** area.

To change the icon for the currently selected item, click on the **Browse** button. A standard browse dialog will be displayed. Any icon chosen through the browser replaces the icon currently associated with the given item. Clicking on the **Clear** button removes the icon currently associated with the given item, leaving the item without an icon.

Following properties can be modified: **Appearance, BorderStyle, Font, IconMQWorkflow, IconOneProcessInstanceList, IconOneProcessTemplateList, IconOneWorklist, IconProcessInstanceLists, IconProcessTemplateLists, IconSystem, IconWorklists**.

Methods

The ExecutionService Control supports the following methods besides the methods supported by all controls - see "Chapter 57. Methods supported by all GUI controls" on page 695:

ConnectGUI

Connects the ExecutionServiceCtrl object with the MQWorkflowCtrl object.

Signature

```
long ConnectGUI( IDispatch * wfc )
```

Parameters

wfc Input. The pointer to the MQWorkflowCtrl object.

Return type

long If processing completes successfully, FMC_OK is returned.

ContextMenuDeleteProcInstList

Calls the context menu item 'Delete Process Instance List'.

Signature

```
long ContextMenuDeleteProcInstList( long index1, long index2 )
```

Parameters

index1 Input. The index within the ExecutionServiceArray.
index2 Input. The index within the ProcessInstanceListArray.

Return type

long Return code of *FmcjProcessInstanceList::Delete()*.

ContextMenuDeleteProcTempList

Calls the context menu item 'Delete Process Template List'.

Signature

```
long ContextMenuDeleteProcTempList( long index1, long index2 )
```

Parameters

index1 Input. The index within the ExecutionServiceArray.
index2 Input. The index within the ProcessTemplateListArray.

Return type

long Return code of *FmcjProcessTemplateList::Delete()*.

ContextMenuDeleteWorklist

Calls the context menu item 'Delete Worklist'.

Signature

```
long ContextMenuDeleteWorklist( long index1, long index2 )
```

Parameters

index1 Input. The index within the ExecutionServiceArray.

index2 Input. The index within the WorklistListArray.

Return type

long Return code of *FmcjWorklistList::Delete()*.

ContextMenuLogoff

Calls the context menu item 'Logoff'.

Signature

```
long ContextMenuLogoff( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

Return type

long Return code of *FmcjExecutionService::Logoff()*.

ContextMenuLogon

Calls the context menu item 'Logon'.

Signature

```
long ContextMenuLogon( long  index,
                       BSTR  userID,
                       BSTR  password )
```

Parameters

index Input. The index within the ExecutionServiceArray.

userID Input. The user identification to logon with.

password Input. The password.

Return type

long Return code of *FmcjExecutionService::Logon()*.

ContextMenuLogonDialog

Calls the context menu item 'Logon'. Calling this method displays a separate logon dialog where the user must specify detailed logon parameters, such as user ID and password.

Signature

```
long ContextMenuLogonDialog()
```

Return type

long Return code of *FmcjExecutionService::Logon()*.

ContextMenuNewProcInstList

Calls the context menu item 'Create New Process Instance List'.

A dialog is shown where the user must specify detailed list creation parameters.

Signature

```
long ContextMenuNewProcInstList( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

Return type

long Return code of *FmcjExecutionService::CreateProcessInstanceList()*.

ContextMenuNewProcTempList

Calls the context menu item 'Create New Process Template List'.

A dialog is shown where the user must specify detailed list creation parameters.

Signature

```
long ContextMenuNewProcTempList( long index  )
```

Parameters

index Input. The index within the ExecutionServiceArray.

Return type

long Return code of *FmcjExecutionService::CreateProcessTemplateList()*.

ContextMenuNewWorklist

Calls the context menu item 'Create New Work List'.

A dialog is shown where the user must specify detailed list creation parameters.

Signature

```
long ContextMenuNewWorklist( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

Return type

long Return code of *FmcjExecutionService::CreateWorklist()*.

ContextMenuProperties

Calls the context menu item 'Properties'.

Signature

```
void ContextMenuProperties()
```

ContextMenuRefresh

Calls the context menu item 'Refresh'.

Signature

```
void ContextMenuRefresh()
```

ContextMenuRefreshProcInstLists

Calls the context menu item 'Refresh Process Instance Lists'.

Signature

```
void ContextMenuRefreshProcInstLists( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuRefreshProcInstances

Calls the context menu item 'Refresh Process Instances'.

Signature

```
void ContextMenuRefreshProcInstances( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuRefreshProcTempLists

Calls the context menu item 'Refresh Process Template Lists'.

Signature

```
void ContextMenuRefreshProcTempLists( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuRefreshProcTemplates

Calls the context menu item 'Refresh Process Templates'.

Signature

```
void ContextMenuRefreshProcTemplates( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuRefreshWorkitems

Calls the context menu item 'Refresh Work Items'.

Signature

```
void ContextMenuRefreshWorkitems( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuRefreshWorklists

Calls the context menu item 'Refresh Worklists'.

Signature

```
void ContextMenuRefreshWorklists( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

ContextMenuUserInformation

Calls the context menu item 'User Information'.

Signature

```
void ContextMenuUserInformation( long index )
```

Parameters

index Input. The index within the ExecutionServiceArray.

Events

The ExecutionService Control triggers the following events besides the events triggered by all non-monitor controls - see "Chapter 59. Events triggered by all GUI controls" on page 705 and "Chapter 60. Events triggered by all non-monitor GUI controls" on page 707:

ItemCollapsed

Occurs when the item has collapsed.

Signature

```
void ItemCollapsed( BSTR name, long type, long index )
```

Parameters

name Output. The item name.

type Output. The type of the item.

index Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.

ItemCollapsing

Occurs when the item is collapsing.

Signature

```
void ItemCollapsing ( BSTR      name,
                      long      type,
                      long      index,
                      boolean * cancel )
```

Parameters

name	Output. The item name.
type	Output. The type of the item.
index	Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.
cancel	Input/Output. A pointer to a boolean variable. If *True* is returned, the item is not collapsed.

ItemExpanded

Occurs when the item has expanded.

Signature

```
void ItemExpanded( BSTR name, long type, long index )
```

Parameters

name	Output. The item name.
type	Output. The type of the item.
index	Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.

ItemExpanding

Occurs when the item is expanding.

Signature

```
void ItemExpanding( BSTR      name,
                    long      type,
                    long      index,
                    boolean * cancel )
```

Parameters

name	Output. The item name.
type	Output. The type of the item.
index	Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.

cancel Input/Output. A pointer to a boolean variable. If *True* is returned, the item is not collapsed.

SelChanged

Occurs when the selection has changed.

Signature

```
void SelChanged( BSTR name, long type, long index )
```

Parameters

name Output. The item name of the newly selected item.

type Output. The type of the newly selected item.

index Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.

SelChanging

Occurs when the item is changing.

Signature

```
void SelChanging( BSTR name, long type, long index, boolean * cancel )
```

Parameters

name Output. The item name of the newly selected item.

type Output. The type of the newly selected item.

index Output. The index of the corresponding ExecutionService within the ExecutionServiceArray.

cancel Input. A pointer to a boolean variable. If set to *True*, the item is not changed.

Chapter 63. ProcessTemplateListCtrl

For GUI processing, a process template list must be connected to a ProcessTemplateList Control. This connection is established via the method **ConnectGUI()** from the ProcessTemplateListCtrl class. When this connection has been established, the user has full access to the ProcessTemplateList object in the ProcessTemplateList Control.

Properties

In design mode, the properties for the Process TemplateListControl can be viewed and adjusted by placing the mouse pointer within the Control's grid area, clicking the right mouse button and selecting **Properties**.

The property dialog contains four tabs: **Fonts, Colors, Column Selection**, and **Column Attribute**.

All of the tabs are available in design mode and runtime mode.

The display of the Control can be configured in a number of ways via this dialog. There are four options for the **View**; namely **Report**, **List**, **Small Icon**, and **Icon**.

The **Fonts** tab displays a standard font dialog. The font, its size, style, and the effects used to display the text in the Control's window can be manipulated in this dialog.

The **Colors** tab provides a mechanism to manipulate the color scheme of the Control's window.

The foreground and background colors of the Control can be set and altered. Select **ForeColor** or **BackColor**, click on one of the sixteen color buttons. Then click on the **Apply** button.

The **Column Selection** tab allows columns or information to be added to or to be removed from the Control's display. To add columns, highlight items in the right pane and click the **Add** button. To remove columns, highlight items in the left pane and click the **Delete** button.

The **Column Attribute** tab provides a mechanism to manipulate certain aspects of how the Control displays its items. They are as follows:

1. The **Icon view** check box is used to indicate which columns of information (the names of which are shown in the list box on the left) are to be included when the Icon display style is in effect. The columns that are selected at any given time are shown below the icon on the right.
2. The **Column width** is used to indicate the number of pixels a column item will occupy. It only pertains to the **Report** format of thedisplay. To adjust the width of a column, highlight the desired item in the list box on the left, enter a numeric value in the **Column width** entry box, and click the **Apply** button.
3. The **Alignment** is used to justify the text of a column either to the left or to the right. It only pertains to the **Report** format of the display.

In Runtime mode, if the mouse pointer is placed on a given item and the right mouse button is clicked, the context menu for that item is displayed. If the **Properties** option is selected, a tabbed dialog appears.

The property dialog contains four tabs: **General, Data and Staff, History**, and **Documentation**. The **General** tab displays the following pieces of information belonging to the process template list.

The **Data and Staff** tab displays information belonging to the administration of the listitem object. They are:

- Input and output container names of the process instance.
- The process administrator of the process instance.
- The user who started the process instance.
- The role and organization criteria for the activities assigned to the users for the process instance.

The **History** tab displays the template information. They are:

- TimeStamps:

  The date and time when the process template was created, started, and the validfrom-time.

The **Documentation** tab displays any annotations added for the listitem object.

Note: In the Client, or any application created with the ProcessTemplateList Control, which utilizes a menu bar, the process item properties dialog (and all other context menu items) can also be accessed via the menu bar's **Process** option.

ProcessTemplateList Settings

The settings for the process template list can be viewed and adjusted by placing the mouse pointer within an unused portion of the Control's window area, clicking the right mouse button, and selecting **ProcessTemplateList settings**.

Following properties can be modified: **Appearance, Arrange, BackColor, BorderStyle, CtrlColumnOrder, CtrlColumnWidth, Font, ForeColor, View**.

Methods

The ProcessTemplateList Control supports the following methods besides the methods supported by all list controls - see "Chapter 57. Methods supported by all GUI controls" on page 695 and "Chapter 58. Methods supported by all list controls" on page 699:

ContextMenuCreateInstance

Calls the context menu item 'Create Instance'.

Signature

```
long ContextMenuCreateInstance()
```

Return type

long Return code of *FmcjProcessTemplate::CreateInstance()*.

RefreshProcessTemplateList

Refreshes the ProcessTemplateList Control.

Signature

```
long RefreshProcessTemplateList()
```

Return type

long If processing completes successfully, FMC_OK is returned.

Events

See "Chapter 59. Events triggered by all GUI controls" on page 705, "Chapter 60. Events triggered by all non-monitor GUI controls" on page 707, and "Chapter 61. Events triggered by all list controls" on page 711.

Chapter 64. ProcessInstanceListCtrl

To access the process instance list, the ProcessInstanceList object must be connected to a ProcessInstanceList Control. This connection is established via the function ConnectGUI() from the ProcessInstanceListCtrl class. When this connection has been established, the user has full access to the ProcessInstanceList object in the ProcessInstanceList Control.

Properties

In design mode, the properties for the ProcessInstanceList Control can be viewed and adjusted by placing the mouse pointer within the Control's grid area, clicking the right mouse button and selecting **Properties**.

The property dialog contains four tabs: **Fonts, Colors, Column Selection** and **Column Attribute**.

All of the tabs are available in design mode and run mode.

The display of the Control can be configured in a number of ways via this dialog. There are four options for the **View**; namely **Report**, **List**, **Small Icon**, and **Icon**.

The **Fonts** tab displays a standard font dialog. The font, its size, style, and the effects used to display the text in the Control's window can be manipulated here.

The **Colors** tab provides a mechanism to manipulate the color scheme of the Control's window.

The foreground and background colors of the Control can be set or altered. Select **ForeColor** or **BackColor**, click on one of the sixteen color buttons, and click on the **Apply** button.

The **Column Selection** tab allows columns or information to be added to or to be removed from the Control's display. To add columns, highlight items in the right pane, and click the **Add** button. To remove columns, highlight items in the left pane, and click the **Delete** button.

The **Column Attribute** tab provides a mechanism to manipulate certain aspects of how the Control displays its items. They are as follows:

1. The **Icon view** check box is used to indicate which columns of information (the names of which are shown in the list box on the left) are to be included when the Icon display style is in effect. The columns that are selected at any given time are shown below the icon on the right.
2. The **Column width** is used to indicate the number of pixels a column item will occupy. It only pertains to the **Report** format of the display. To adjust the width of a column, highlight the desired item in the list box at the left, enter a numeric value in the **Column width** entry box, and click the **Apply** button.
3. The **Alignment** is used to justify the text of a column either to the left or to the right. It only pertains to the **Report** format of the display.

In Runtime mode, if the mouse pointer is placed on a given item and the right mouse button is clicked, the context menu for that item is displayed. If the **Properties** option is selected, a tabbed dialog appears.

The property dialog contains four tabs: **General, Data and Staff, History**, and **Documentation**. The **General** tab displays the following pieces of information pertaining to the ProcessInstance. They are:

- Name
- Description
- Process category
- Status
- Parent process of the process instance
- Top-level process of the process instance
- Whether or not the process instance is being audited
- Whether or not the process instance is exited in case an error occurred
- Whether or not the starter of the process instance will be prompted to enter data for the process instance

The **Data and Staff** tab displays information relating to the administration of the list-item object. They are:

- Input and output Container names of the process instance
- The process administrator of the process instance
- The user who started the process instance
- The role and organization criteria for the users assigned activities for the process instance

The **History** tab displays three pieces of activity information. They are:

- TimeStamps

  The date and time when the process instance was created, started, and last modified.

- Notification

  The date and time that the first notification will be sent (or was sent) for the activity.
- Finished

  The date and time the activity was completed.

The **Documentation** tab displays any annotations added for the list-item object.

Note: In the Runtime Client or any application created with the ProcessInstanceList Control, which utilizes a menu bar, the process item property dialog (and all other context menu items) can also be accessed via the menu bar's **Process** option.

ProcessInstanceList Settings

The settings for the process instance list can be viewed and adjusted by placing the mouse pointer within an unused portion of the Control's window area, clicking the right mouse button and selecting **ProcessInstanceList settings**.

The ProcessInstanceList settings dialog displays three tabs: **General, Filter**, and **Sort**.

In the **General** page the name and the type of the process template list are shown. Here the user can also modify the threshold value and the description of the process template list. The other two pages are intended to display and modify the Filter and the Sort Criteria.

Following properties can be modified: **Appearance, Arrange, BackColor, BorderStyle, CtrlColumnOrder, CtrlColumnWidth, Font, ForeColor, View**.

Methods

The ProcessInstanceList Control supports the following methods besides the methods supported by all list controls - see "Chapter 57. Methods supported by all GUI controls" on page 695 and "Chapter 58. Methods supported by all list controls" on page 699:

ContextMenuRestart

Calls the context menu item 'Restart'.

This method restarts the selected list of process instances within the Control. See "Restart()" on page 548 for a more detailed description.

Signature

```
long ContextMenuRestart()
```

Return type

long Return code of *FmcjProcessInstance::Restart()*.

ContextMenuResume

Calls the context menu item 'Resume'.

This method resumes the selected list of process instances within the Control. See "Resume()" on page 550 for a more detailed description.

Signature

```
long ContextMenuResume()
```

Return type

long Return code of *FmcjProcessInstance::Resume()*.

ContextMenuResumeDeep

Calls the context menu item 'Resume deep'.

This method resumes the selected list of process instances within the Control. See "Resume()" on page 550 for a more detailed description.

Signature

```
long ContextMenuResumeDeep()
```

Return type

long Return code of *FmcjProcessInstance::Resume()*.

ContextMenuStart

Calls the context menu item 'Start'.

This method starts the selected list of process instances within the Control. See "Start()" on page 557 for a more detailed description.

Signature

```
long ContextMenuStart()
```

Return type
long Return code of *FmcjProcessInstance::Start()*.

ContextMenuSuspend

Calls the context menu item 'Suspend'.

This method suspends the selected list of process instances within the Control. See "Suspend()" on page 560 for a more detailed description.

Signature

```
long ContextMenuSuspend()
```

Return type
long Return code of *FmcjProcessInstance::Suspend()*.

ContextMenuSuspendDeep

Calls the context menu item 'Suspend deep'.

This method suspends the selected list of process instances within the Control. See "Suspend()" on page 560 for a more detailed description.

Signature

```
long ContextMenuSuspendDeep()
```

Return type
long Return code of *FmcjProcessInstance::Suspend()*.

ContextMenuTerminate

Calls the context menu item 'Terminate'.

This method terminates the selected list of process instances within the Control. See "Terminate()" on page 563 for a more detailed description.

Signature

```
long ContextMenuTerminate ()
```

Return type
long Return code of *FmcjProcessInstance::Terminate()*.

RefreshProcessInstanceList

Refreshes the ProcessInstanceList Control.

Signature

```
long RefreshProcessInstanceList()
```

Return type

long If processing completes successfully, FMC_OK is returned.

Events

See “Chapter 59. Events triggered by all GUI controls” on page 705, “Chapter 60. Events triggered by all non-monitor GUI controls” on page 707, and “Chapter 61. Events triggered by all list controls” on page 711.

Chapter 65. WorklistCtrl

The following describes the methods related to the WorklistCtrl Control. A worklist comprises a set of work items for a user. The worklist object reflects the worklist, which is stored in the IBM MQWorkflow Runtime server's database.

Properties

In design mode, the properties for the Worklist Control can be viewed and adjusted by placing the mouse pointer within the Control's grid area, clicking the right mouse button and selecting Properties.

The property dialog contains four tabs: **Fonts**, **Colors**, **Column Selection**, and **Column Attribute**.

All of the tabs are available in design mode and runtime mode.

The **Fonts** tab displays a standard font dialog. The font, its size, style, and the effects used to display the text in the Worklist Control's window can be manipulated here.

The **Colors** tab provides a mechanism to manipulate the color scheme of the Control's window.

The foreground and background colors of the Control can be set or altered. Select **ForeColor** or **BackColor**, click on one of the sixteen color buttons, and click on the **Apply** button.

The **Column Selection** tab allows columns or information to be added to or to be removed from the Control's display. To add columns, highlight items in the right pane, and click the **Add** button. To remove columns, highlight in the left pane, and click the **Delete** button.

The **Icon view and Width tab** provides a mechanism to manipulate certain aspects of how the Control displays its items. They are as follows:

1. The **Icon view** check box is used to indicate which columns of information (the names of which are shown in the list box on the left) are to be included when the Icon display style is in effect. The columns that are selected at any given time are shown below the icon on the right. They are **Description** and **Activity Type**.

2. The **Column width** is used to indicate the number of pixels a column item will occupy. It only pertains to the **Report** format of the display. To adjust the width of a column, highlight the desired item in the list box on the left, enter a numeric value in the **Column width** entry box, and click the **Apply** button.
3. The **Alignment** is used to justify the text of a column either to the left or to the right. It only pertains to the **Report** format of the display.

Runtime Mode

If the mouse pointer is placed on a given item and the right mouse button is clicked, the context menu for that item is displayed. If the Properties option is selected, a tabbed dialog appears.

The property dialog contains five tabs: **General**, **Staff**, **Start & exit** , **History**, and **Documentation**. The **General** tab displays the following pieces of information pertaining to the Worklist-item object. They are:

- Activity name
- Activity status
- Activity type
- Program

  The name of the program or process assigned to the activity.
- Received as

  A user category that defines why the activity was placed on the worklist:
  - The user who has been assigned the activity.
  - The substitute who received the activity because a defined user is declared absent.
  - The process administrator who received the activity because the defined user is declared absent and no substitute has been specified for the defined user.

    Note: If no process administrator is defined for a process in Buildtime, the person who starts the process instance is assigned as the process administrator of the process instance.
  - The system administrator who received the activity because the process administrator has been deleted.

  The **Staff** tab displays information belonging to the administration of the Worklist-item object. They are:
- On the worklist of

  User IDs of the users who were assigned the activity or (if the activity has been started) the user ID of the user who started the activity.

- Process administrator

  The process administrator of the process instance to which the activity belongs.
- Priority

  The priority assigned to the activity in Buildtime.
- Started on Server

  The name of the server on which the activity was initiated.

The **Start & exit** tab displays information relating to the start and exit conditions of the work item. They are:

- Start

  The start mode of the activity (manual or automatic).
- Start condition:

  The condition that must be met before the activity is started (automatic start) or is added to a worklist (manual start).
- Exit

  The exit mode of the activity (manual or automatic).
- Exit condition

  The condition that must be met before the activity is finished.

The **History** tab displays the following activity information:

- Received

  The date and time when the activity arrived on the worklist.
- Notification

  The date and time when the first notification is due for the activity.
- Finished

  The date and time when the activity was completed.

The **Documentation** tab displays any annotations added for the list.

Note: In the Client or any application created with the Worklist Control, which also utilizes a menu bar, the work item properties dialog (and all other context menu items) can also be accessed via the menu bar's **Activity** option.

Worklist Settings

The settings for the Worklist Control can be viewed and adjusted by placing the mouse pointer within an unused portion of the Control window area, clicking the right mouse button and selecting **Worklist settings**.

The **General** tab displays various pieces of information pertaining to the worklist and its appearance. The ordering of the first three columns (from left to right) can be set along with the sort order (ascending or descending). The column ordering only applies when the report format is in effect.

After adjusting the settings, select the **OK** button. Items that meet the selection criteria are displayed on the worklist. The filter criteria are saved and persist, even between logons.

Following properties can be modified: **Appearance, Arrange, BackColor, BorderStyle, CtrlColumnOrder, CtrlColumnWidth, Font, ForeColor, View**.

Methods

The Worklist Control supports the following methods besides the methods supported by all list controls - see "Chapter 57. Methods supported by all GUI controls" on page 695 and "Chapter 58. Methods supported by all list controls" on page 699:

ContextMenuFinish

Calls the context menu item 'Finish'.

This method finishes the selected list of work items within the Control. See "Finish()" on page 638 for a more detailed description.

Signature

```
long ContextMenuFinish()
```

Return type

long Return code of *FmcjWorkitem::Finish()*.

ContextMenuForceFinish

Calls the context menu item 'Force finish'.

This method finishes the selected list of Workitems within the Control. See "ForceFinish()" on page 640 for a more detailed description.

Signature

```
long ContextMenuForceFinish()
```

Return type

long Return code of *FmcjWorkitem::ForceFinish()*.

ContextMenuForceRestart

Calls the context menu item 'Force restart'.

This method restarts the selected list of Workitems within the Control. See "ForceRestart()" on page 643 for a more detailed description.

Signature

```
long ContextMenuForceRestart()
```

Return type

long Return code of *FmcjWorkitem::ForceFinish()*.

ContextMenuRestart

Calls the context menu item 'Restart'.

This method restarts the selected list of Workitems within the Control. See "Restart()" on page 650 for a more detailed description.

Signature

```
long ContextMenuRestart()
```

Return type

long Return code of *FmcjWorkitem::Restart()*.

ContextMenuSelectAll

Calls the context menu item 'Select all'.

This method selects all objects within the List Control.

Signature

```
long ContextMenuSelectAll()
```

Return type

long If processing completes successfully, FMC_OK is returned.

ContextMenuStart

Calls the context menu item 'Start'.

This method starts the selected list of work items within the Control. See "Start()" on page 652 for a more detailed description.

Signature

```
long ContextMenuStart()
```

Return type

long Return code of *FmcjWorkitem::Start()*.

ContextMenuStartTool

Calls the context menu item 'Start tool'.

This method starts the selected list of support tools within the Control. See "StartTool()" on page 655 for a more detailed description.

Signature

```
long ContextMenuStartTool()
```

Return type

long Return code of *FmcjWorkitem::StartTool()*.

ContextMenuTransfer

Calls the context menu item 'Transfer'.

This method transfers the selected list of work items within the Control from one user to another one. A separate dialog is shown to specify the *fromUserID* as well as the *toUserID*. A dropdown list provides a list of user IDs from the FDL. The dropdown list is empty in case you are authorized for all users, in which case you must explicitly specify the *toUserID*. The transfer occurs for all selected items within the List Control.

See "Transfer()" on page 505 for a more detailed description.

Signature

```
long ContextMenuTransfer()
```

Return type

long Return code of *FmcjWorkitem::Transfer()*.

PushOption

This method is used to return the current setting of the PUSH option for this worklist.

If *True* is returned, PUSH processing is enabled for the worklist. In this case, pushed items (work items as well as notifications) are used to dynamically update the items in the worklist. The user does not need to invoke worklist refresh processing in order to add new items to the worklist or to see changes (state, time stamps) of items already contained in the worklist.

The MQ Workflow Runtime client invokes this method in order to display the current setting within the Settings dialog of a specific worklist.

Signature

```
boolean PushOption()
```

Return type

boolean If push is enabled for the worklist, *True* is returned, *False* otherwise.

RefreshWorklist

Refreshes the Worklist Control.

Signature

```
long RefreshWorklist()
```

Return type

long If processing completes successfully, FMC_OK is returned.

SetPushOption

This method is used to enable or disable PUSH processing for the current worklist.

The MQ Workflow Runtime client invokes this method when the user changes the value of the PUSH check-box within the Settings dialog of a specific worklist.

Signature

```
void SetPushOption( boolean value )
```

Parameters

value Input. An indicator whether PUSH processing is to be enabled or not.

Events

The Worklist Control supports the following events besides the events triggered by all non-monitor controls - see “Chapter 59. Events triggered by all GUI controls” on page 705, “Chapter 60. Events triggered by all non-monitor GUI controls” on page 707, and “Chapter 61. Events triggered by all list controls” on page 711.

ActivityInstanceNotificationChanged

This OLE event is fired when an existing activity instance notification of the current worklist has changed.

The index parameter denotes the index of the changed activity instance notification in the ActivityInstanceNotifArray of the current worklist.

The WorklistCtrl as exploited by the MQ Workflow Runtime client uses this event to refresh the values in the GUI.

Signature

```
ActivityInstanceNotificationChanged( long index )
```

Parameters

index Input. The index of the activity instance notification in the current worklist.

ProcessInstanceNotificationChanged

This OLE event is fired when an existing process instance notification of the current worklist has changed.

The index parameter denotes the index of the changed process instance notification in the ProcessInstanceNotifArray of the current worklist.

The WorklistCtrl as exploited by the MQ Workflow Runtime client uses this event to refresh the values in the GUI.

Signature

```
ProcessInstanceNotificationChanged( long index )
```

Parameters

index Input. The index of the process instance notification in the current worklist.

WorkitemChanged

This OLE event is fired when an existing work item of the current worklist has changed.

The index parameter denotes the index of the changed work item in the WorkitemArray of the current worklist.

The WorklistCtrl as exploited by the MQ Workflow Runtime client uses this event to refresh the values in the GUI.

Signature

```
WorkitemChanged( long index )
```

Parameters

index Input. The index of the work item in the current worklist.

Starting

Occurs when a work item was started.

Signature

```
void Starting( BSTR name )
```

Parameters

name Output. The string denoting the name of the started work item.

Chapter 66. MonitorCtrl

Properties

In design mode, the properties for the Monitor Control can be viewed and adjusted by placing the mouse pointer within the Control's grid area, clicking the right mouse button, and selecting Properties.

The property dialog contains two tabs: **General** and **Colors**.

All of the tabs are available in design mode and runtime mode.

The **General** tab displays the Zoom factor which can be changed for the current session. A valid value is between 10 and 200. Zoom factor 100 is the default.

The **Colors** tab provides a mechanism to manipulate the color scheme of the Control's window.

It displays the current color settings. For the current session you can change the color used for selected items as well as the color for the background (PaperColor).

Methods

The Monitor Control supports the following methods besides the methods supported by all controls - see "Chapter 57. Methods supported by all GUI controls" on page 695.

ActivityProperties()

This method displays the property pages of the currently selected activity instance.

Signature

```
void ActivityProperties()
```

ConnectGUI

Connects the MonitorCtrl object with the MQWorkflowCtrl object.

Signature

```
void ConnectGUI( IDispatch * wfc )
```

Parameters

wfc Input. The pointer to the MQWorkflowCtrl object.

ControlConnectorProperties

This method displays the property pages of the currently selected ControlConnector.

Signature

```
void ControlConnectorProperties()
```

OpenMonitor

This method displays the process model graph as provided by the monitor parameter. The graph layout is controlled by the layout coordinates provided by the underlying C++ Api layer. The request is ignored when issued more than once.

Signature

```
void OpenMonitor( IDISPATCH * monitor )
```

Parameters

monitor Input. The pointer to the instance monitor object.

Refresh

This method refreshes the current process model graph. It must be used to refresh activity instance states. An automatic refresh is not supported.

Signature

```
long Refresh()
```

Return type

long The return code of *FmcjBlockInstanceMonitor::Refresh()*.

Events

The Monitor Control triggers the following events besides the events triggered by all controls - see "Chapter 59. Events triggered by all GUI controls" on page 705.

AfterRefreshing

This OLE event is fired when refreshing ends. It is preceded by a *BeforeRefreshing* event.

Signature

```
void AfterRefreshing()
```

BeforeRefreshing

This OLE event is fired when the user clicks the *Refresh* menu item. It is preceded by a *DoRefresh* event.

Signature

```
void BeforeRefreshing()
```

BlockActivityClick

This OLE event is fired when the user clicks the right mouse button over an activity instance of kind *Block*.

Signature

```
void BlockActivityClick( IDISPATCH *       activity,
                         OLE_XPOS_PIXELS   x,
                         OLE_XPOS_PIXELS   y,
                         long              button,
                         boolean *         enableDefault )
```

Parameters

activity	Input. A pointer to the activity instance object.
button	Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.
enableDefault	Input. An indicator whether the Monitor Control should perform its default action and display the context menu.
x	Input. The x-coordinate of the mouse when the click occurred.
y	Input. The y-coordinate of the mouse when the click occurred.

BlockActivityDoubleClick

This OLE event is fired when the user double- clicks the left mouse button over an activity instance of kind *Block*.

Signature

```
void BlockActivityDoubleClick( IDISPATCH *       activity,
                               OLE_XPOS_PIXELS   x,
                               OLE_XPOS_PIXELS   y,
                               long              button,
                               boolean *         enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action and open a new monitor window.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

ControlConnectorClick

This OLE event is fired when the user clicks the right mouse button over a control connector instance.

Signature

```
void ControlConnectorClick( IDISPATCH *       connector,
                            OLE_XPOS_PIXELS   x,
                            OLE_XPOS_PIXELS   y,
                            long              button,
                            boolean *         enableDefault )
```

Parameters

connector Input. A pointer to the control connector instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action and display the context menu.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

ControlConnectorDoubleClick

This OLE event is fired when the user double-clicks the left mouse button over a control connector instance.

Signature

```
void ControlConnectorDoubleClick( IDISPATCH *       connector,
                                  OLE_XPOS_PIXELS   x,
                                  OLE_XPOS_PIXELS   y,
                                  long              button,
                                  boolean *         enableDefault )
```

Parameters

connector Input. A pointer to the control connector instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action and display the control connector instance property page.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

DoActivityEnter

This OLE event is fired when the user selects an activity instance and presses the Enter key.

Signature

```
void DoActivityEnter( IDISPATCH * activity,
                      boolean *   enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

enableDefault Input. An indicator whether the Monitor Control should perform its default action which is the same as the mouse double-click; it depends on the activity instance type.

DoControlConnectorEnter

This OLE event is fired when the user selects a control connector instance and presses the Enter key.

Signature

```
void DoControlConnectorEnter( IDISPATCH * connector,
                              boolean *   enableDefault )
```

Parameters

connector Input. A pointer to the control connector instance object.
enableDefault Input. An indicator whether the Monitor Control should perform its default action; currently there is no default action.

DoRefresh

This OLE event is fired when the user selects *Refresh* from the context menu.

Signature

```
void DoRefresh( boolean * enableDefault )
```

Parameters

enableDefault Input. An indicator whether the Monitor Control should perform its default action and issue a complete refresh of all activity instances and control connector instances displayed within the Monitor window.

DoShowContextMenu

This event is fired in case the right mouse button is clicked to show the context menu. The user can cancel the ContextMenu display by setting the enableDefault parameter to *False*.

Signature

```
void DoShowContextMenu( boolean * enableDefault )
```

Parameters

enableDefault Input. An indicator whether the Monitor Control should perform its default action and display the context menu.

Error

Occurs when opening a monitor signals an error.

Signature

```
void Error( short     returnCode,
            BSTR *    messageText,
            boolean * cancelDisplay )
```

Parameters

returnCode Input. The return code.
messageText Input. The formatted message text describing the error.
cancelDisplay Input. If set to *True*, the display is canceled.

MonitorOpen

This OLE event is fired when the user clicks the in-place (or context) menu monitor item *Open Activity*.

Signature

```
void MonitorOpen( IDISPATCH * activity )
```

Parameters

activity Input. A pointer to the activity instance object.

ProcessActivityClick

This OLE event is fired when the user clicks the right mouse button over an activity instance of kind *Process*.

Signature

```
void ProcessActivityClick( IDISPATCH *       activity,
                           OLE_XPOS_PIXELS   x,
                           OLE_XPOS_PIXELS   y,
                           long              button,
                           boolean *         enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action and display the context menu.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

ProcessActivityDoubleClick

This OLE event is fired when the user double-clicks the left mouse button over an activity instance of kind *Process*.

Signature

```
void ProcessActivityDoubleClick( IDISPATCH *       activity,
                                 OLE_XPOS_PIXELS   x,
                                 OLE_XPOS_PIXELS   y,
                                 long              button,
                                 boolean *         enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action; currently there is not default action.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

ProgramActivityClick

This OLE event is fired when the user clicks the right mouse button over an activity instance of kind *Program*. It displays the properties page of the activity instance.

Signature

```
void ProgramActivityClick( IDISPATCH *       activity,
                           OLE_XPOS_PIXELS   x,
                           OLE_XPOS_PIXELS   y,
                           long              button,
                           boolean *         enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

button Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.

enableDefault Input. An indicator whether the Monitor Control should perform its default action and display the context menu.

x Input. The x-coordinate of the mouse when the click occurred.

y Input. The y-coordinate of the mouse when the click occurred.

ProgramActivityDoubleClick

This OLE event is fired when the user double-clicks the left mouse button over an activity instance of kind *Program*. It displays the property pages of the activity instance.

Signature

```
void ProcessActivityDoubleClick( IDISPATCH *       activity,
                                 OLE_XPOS_PIXELS   x,
                                 OLE_XPOS_PIXELS   y,
                                 long              button,
                                 boolean *         enableDefault )
```

Parameters

activity Input. A pointer to the activity instance object.

button	Input. Indicates which button was pressed; 0 denotes the left button, 2 the right button.
enableDefault	Input. An indicator whether the Monitor Control should perform its default action and display the property pages of the selected activity instance.
x	Input. The x-coordinate of the mouse when the click occurred.
y	Input. The y-coordinate of the mouse when the click occurred.

Part 9. Examples and scenarios

Chapter 67. Scenarios

The following scenarios are delivered with MQ Workflow. Scenarios are intended to demonstrate some functionality of the product. They can be executed and looked at. In order to execute a scenario:

1. Import the FDL.
2. Start the MQ Workflow system.
3. Execute the scenario; by default, it has been installed in the \bin subdirectory of your installation directory if you selected to install the samples.

Refer to the readme file in the appropriate directory for possible updates.

- A sample credit request

  For the Windows platforms and ActiveX in \Program Files\MQSeries Workflow\Scenario\Credit:

  – The FDL: fmccred.fdl

  – The activity implementations: fmcn6bna.vbp, fmcn6bni.vbp, fmcn6bnp.vbp, fmcn6bnr.vbp

- A sample life insurance request

  For the Windows platforms and ActiveX in \Program Files\MQSeries Workflow\Scenario\Life:

  – The FDL: fmclife.fdl

Chapter 68. Examples

The following examples are delivered with the MQ Workflow; examples are intended to demonstrate some API usage. They can be compiled and linked and then executed. Some examples also provide a version which can be executed. They can then be found in the \bin subdirectory of your installation directory if you selected to install the samples.

Note: For the most recent set of examples, look at

```
//http://www-4.ibm.com/software/ts/mqseries/txppacs
```

for the MQ Workflow support packs. For example,

- MQ Workflow API Programming Examples
- MQ Workflow Rapid Deployment Wizard

Refer to the readme file in the appropriate directory for possible updates.

- Container handling in an activity implementation
  - For all supported platforms and the C-language in the \smp\c\actimpl subdirectory of the install directory: fmctjcim.c
  - For all supported platforms and the C++-language in the \smp\c++\actimpl subdirectory of the install directory: fmctjpim.cxx
  - For the Windows platforms and ActiveX in \Program Files\MQSeries Workflow\Smp\VB\Actimpl: fmcnshow.vbp

  Note: These programs are able to analyze an unknown container. Especially the fmcnshow program can be used as your initial activity implementation in order to test a new process model.
- The Runtime client

  For the Windows platforms and ActiveX in \Program Files\MQSeries Workflow\Smp\VB\Rtc: fmcn6rtc.vbp
- Hello world

  For all supported platforms and the Java language:
  - In the \smp\java\HelloApplication subdirectory of the installation directory: HelloApplication.java
  - In the \smp\java\HelloApplet subdirectory of the installation directory: HelloApplet.java and HelloApplet.html
  - In the \smp\java\HelloApplet1 subdirectory of the installation directory: HelloApplet1.java and HelloApplet1.html

  - In the \smp\java\HelloServlet subdirectory of the installation directory: HelloServlet.java and HelloServlet.html
- Distributed process example using XML

  For all supported platforms in the \smp\dpxml subdirectory of the installation directory.
- Authentication exit example
  - For all supported platforms and the C-language in the \smp\c\authexit subdirectory of the installation directory.
  - For all supported platforms and the Java language in the \smp\java\authexit subdirectory of the installation directory.

The following chapters additionally show some examples. They are intended to present the stated concept only.

Chapter 69. How to create persistent lists

The following examples show how to create a persistent list, that is, a persistent view on a set of objects. They define a view on process instances. Other possible lists to define are process template lists or worklists.

Create a process instance list (ActiveX)

```
Dim eService As ExecutionService
Dim Err      As String

MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.CreateProcessInstanceList(
              "PIL1",
              TypeOfList.TypeOfList_Private, "ADMIN", False,
              "", True,
              "", True,
              "", True,
              0, True )
If Rc <> 0 Then
  Err = "CreateProcessInstanceList failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

Create a process instance list (C-language)

```
#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;
  }
```

Figure 27. Sample C program to create a process instance list (Part 1 of 4)

```
  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;
  }
```

Figure 27. Sample C program to create a process instance list (Part 2 of 4)

```
 /* create a process instance list */
 rc = FmcjExecutionServiceCreateProcessInstanceList(
                           service,
                           name,
                           Fmc_LT_Private,
                           "USERID",
                           desc,
                           FmcjNoFilter,
                           FmcjNoSortCriteria,
                           &threshold,
                           &instanceList );
```

Figure 27. Sample C program to create a process instance list (Part 3 of 4)

```
  if ( rc != FMC_OK)
    printf( "CreateProcessInstanceList returns: %u%\n",rc );
  else
    printf( "CreateProcessInstanceList okay\n" );

  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 27. Sample C program to create a process instance list (Part 4 of 4)

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           onwer ("USERID");
  unsigned long    threshold= 10;
```

Figure 28. Sample C++ program to create a process instance list (Part 1 of 2)

```
  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 28. Sample C++ program to create a process instance list (Part 2 of 2)

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);
    }
```

Figure 29. Sample Java program to create a process instance list (Part 1 of 8)

```
try
    {
      // An agent bean representing a 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 29. Sample Java program to create a process instance list (Part 2 of 8)

```
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("", "");
```

Figure 29. Sample Java program to create a process instance list (Part 3 of 8)

```
// Log on to the execution service. If the UserID and/or the password is
// invalid, a 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;
```

Figure 29. Sample Java program to create a process instance list (Part 4 of 8)

```
      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");
         }
      }
```

Figure 29. Sample Java program to create a process instance list (Part 5 of 8)

```
     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 29. Sample Java program to create a process instance list (Part 6 of 8)

```
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();
    }
```

Figure 29. Sample Java program to create a process instance list (Part 7 of 8)

```
catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 29. Sample Java program to create a process instance list (Part 8 of 8)

Chapter 70. How to query persistent lists

The following examples show how to retrieve persistent lists from the MQ 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.

Query worklists (ActiveX)

```
Dim eService As ExecutionService
Dim wl       As Worklist
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As Integer

MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.QueryWorklists
If Rc <> 0 Then
  Err = "QueryWorklists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.WorklistArray.GetSize
  For i = 0 To s - 1
    Set wl = eService.WorklistArray.GetAt(i)

    Msg = "Worklist: Name = " + wl.Name
    MsgBox Msg, vbInformation, "Worklist"

  Next i

End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

Query worklists (C-language)

```
#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]= "";
```

Figure 30. Sample C program to query worklists (Part 1 of 10)

```
  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;
  }
```

Figure 30. Sample C program to query worklists (Part 2 of 10)

```
  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &lists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%\n",rc );
  else
     printf( "QueryWorklists() returns okay\n" );
```

Figure 30. Sample C program to query worklists (Part 3 of 10)

```
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 30. Sample C program to query worklists (Part 4 of 10)

```
    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) );
```

Figure 30. Sample C program to query worklists (Part 5 of 10)

```
    FmcjWorklistDescription( worklist, tInfo, 4097 );
    printf("- Description                   : %s\n",tInfo);
    printf("- Description is null ?         : %u\n",
             FmcjWorklistDescriptionIsNull(worklist) );
```

Figure 30. Sample C program to query worklists (Part 6 of 10)

```
    FmcjWorklistFilter( worklist, tInfo, 4097 );
    printf("- Filter                        : %s\n",tInfo);
    printf("- Filter is null ?              : %u\n",
             FmcjWorklistFilterIsNull(worklist) );
```

Figure 30. Sample C program to query worklists (Part 7 of 10)

```
      FmcjWorklistSortCriteria( worklist, tInfo, 4097 );
      printf("- SortCriteria                  : %s\n",tInfo);
      printf("- SortCriteria is null ?        : %u\n",
                FmcjWorklistSortCriteriaIsNull(worklist) );
```

Figure 30. Sample C program to query worklists (Part 8 of 10)

```
      printf("- Threshold                     : %u\n",
                FmcjWorklistThreshold(worklist) );
      printf("- Threshold is null ?           : %u\n",
                FmcjWorklistThresholdIsNull(worklist) );
                                /* deallocate just read object  */
      FmcjWorklistDeallocate(&worklist);
    }
    FmcjWorklistVectorDeallocate(&lists);
  }
```

Figure 30. Sample C program to query worklists (Part 9 of 10)

```
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 30. Sample C program to query worklists (Part 10 of 10)

```
#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;
  }
```

Figure 31. Sample C++ program to query worklists (Part 1 of 10)

```
  // 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;
```

Figure 31. Sample C++ program to query worklists (Part 2 of 10)

```
  if (rc == FMC_OK)
  {
    unsigned int numWList= lists.size();
    cout << "Number of worklists returned : " << numWList << endl;
```

Figure 31. Sample C++ program to query worklists (Part 3 of 10)

```
for( unsigned long i=0; i< numWList; i++ )
{
  worklist= lists[i];
  cout << "Name              : " << worklist.Name()              << endl;
```

Figure 31. Sample C++ program to query worklists (Part 4 of 10)

```
  cout << "Type              : " <<
          ((worklist.Type() == FmcjPersistentList::Private) ? "private" :
           (worklist.Type() == FmcjPersistentList::Public)  ? "public"  :
           "not set" )           << endl;
```

Figure 31. Sample C++ program to query worklists (Part 5 of 10)

```
 cout << "Owner             : " << worklist.OwnerOfList()       << endl;
  cout << "Owner null ?      : " << worklist.OwnerOfListIsNull() << endl;
```

Figure 31. Sample C++ program to query worklists (Part 6 of 10)

```
  cout << "Description       : " << worklist.Description()       << endl;
  cout << "Description null ?: " << worklist.DescriptionIsNull() << endl;
```

Figure 31. Sample C++ program to query worklists (Part 7 of 10)

```
 cout << "Filter            : " << worklist.Filter()            << endl;
  cout << "Filter null ?     : " << worklist.FilterIsNull()      << endl;
  cout << "SortCriteria      : " << worklist.SortCriteria()      << endl;
  cout << "SortCriteria null?: " << worklist.SortCriteriaIsNull()<< endl;
```

Figure 31. Sample C++ program to query worklists (Part 8 of 10)

```
  cout << "Threshold         : " << worklist.Threshold()         << endl;
  cout << "Threshold null ?  : " << worklist.ThresholdIsNull()   << endl;
  cout << endl;     }     cout << endl;   }
```

Figure 31. Sample C++ program to query worklists (Part 9 of 10)

```
  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

Figure 31. Sample C++ program to query worklists (Part 10 of 10)

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);
    }
```

Figure 32. Sample Java program to query worklists (Part 1 of 10)

```
try
{
  // An agent bean representing a 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 32. Sample Java program to query worklists (Part 2 of 10)

```
  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]);
```

Figure 32. Sample Java program to query worklists (Part 3 of 10)

```
 // 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");
```

Figure 32. Sample Java program to query worklists (Part 4 of 10)

```
      // 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
);
```

Figure 32. Sample Java program to query worklists (Part 5 of 10)

```
        // 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 32. Sample Java program to query worklists (Part 6 of 10)

```
          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*/
```

Figure 32. Sample Java program to query worklists (Part 7 of 10)

```
      // 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 32. Sample Java program to query worklists (Part 8 of 10)

```
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();
    }
```

Figure 32. Sample Java program to query worklists (Part 9 of 10)

```
 catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 32. Sample Java program to query worklists (Part 10 of 10)

Chapter 71. 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.

Note: ActiveX supports querying objects only from a predefined list.

Query process instances from a process instance list (ActiveX)

```
Dim eService As ExecutionService
Dim pil      As ProcessInstanceList
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As IntegerDim

MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.QueryProcessInstanceLists
If Rc <> 0 Then
  Err = "QueryProcessInstanceLists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.ProcessInstanceListArray.GetSize

  If s > 0 Then
    Set pil = eService.ProcessInstanceListArray.GetAt(0)
    Rc = pil.QueryProcessInstances
    If Rc <> 0 Then
      Err = "QueryProcessInstances failed, rc = " + Str(Rc)
      MsgBox Err, vbCritical, "Error"
    Else
      Msg = "Number of instances returned: " + Str(pil.GetSize)
      MsgBox Msg, vbInformation, "ProcessInstances"
    End If
  Else
    Err = "No ProcessInstanceList available"
    MsgBox Err, vbCritical, "Error"
  End If
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

Query process instances (C-language)

```
#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" );
```

Figure 33. Sample C program to query process instances (Part 1 of 2)

```
  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);
  }

  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 33. Sample C program to query process instances (Part 2 of 2)

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;
  }
```

Figure 34. Sample C++ program to query process instances (Part 1 of 3)

```
  // 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;
```

Figure 34. Sample C++ program to query process instances (Part 2 of 3)

```
  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 34. Sample C++ program to query process instances (Part 3 of 3)

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);
    }
```

Figure 35. Sample Java program to query process instances (Part 1 of 9)

```
try
    {
      // An agent bean representing a 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 35. Sample Java program to query process instances (Part 2 of 9)

```
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("", "");
```

Figure 35. Sample Java program to query process instances (Part 3 of 9)

```
  // 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");
```

Figure 35. Sample Java program to query process instances (Part 4 of 9)

```
  // 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);
```

Figure 35. Sample Java program to query process instances (Part 5 of 9)

```
        // Iterate over the process instances, printing out their names.
        for (int ndx = 0; ndx < procInstances.length; ndx++)
        {
          System.out.println("   - Name: " + procInstances[ndx].name());
        }
      }
```

Figure 35. Sample Java program to query process instances (Part 6 of 9)

```
   // 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 35. Sample Java program to query process instances (Part 7 of 9)

```
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();
    }
```

Figure 35. Sample Java program to query process instances (Part 8 of 9)

```
    catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 35. Sample Java program to query process instances (Part 9 of 9)

Query work items from a worklist (ActiveX)

```
Dim eService As ExecutionService
Dim wl       As Worklist
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As Integer

MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.QueryWorklists
If Rc <> 0 Then
  Err = "QueryWorklists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.WorkliistArray.GetSize

  If s > 0 Then
    Set wl = eService.WorklistArray.GetAt(0)
    Rc = wl.QueryWorkitems
    If Rc <> 0 Then
      Err = "QueryWorkitems failed, rc = " + Str(Rc)
      MsgBox Err, vbCritical, "Error"
    Else
      Msg = "Number of workitems returned: " + Str(wl.GetSize)
      MsgBox Msg, vbInformation, "Workitems"
    End If
  Else
    Err = "No Worklist available"
    MsgBox Err, vbCritical, "Error"
  End If
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

Query work items from a worklist (C-language)

```
#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" );
```

Figure 36. Sample C program to query work items from a worklist (Part 1 of 2)

```
  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);

    /* 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 36. Sample C program to query work items from a worklist (Part 2 of 2)

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();
```

Figure 37. Sample C++ program to query work items from a worklist (Part 1 of 5)

```
  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }
```

Figure 37. Sample C++ program to query work items from a worklist (Part 2 of 5)

```
  // 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;
    }
```

Figure 37. Sample C++ program to query work items from a worklist (Part 3 of 5)

```
    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 37. Sample C++ program to query work items from a worklist (Part 4 of 5)

```
    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 37. Sample C++ program to query work items from a worklist (Part 5 of 5)

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);
    }
```

Figure 38. Sample Java program to query work items from a worklist (Part 1 of 8)

```
try
{
  // An agent bean representing a 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 38. Sample Java program to query work items from a worklist (Part 2 of 8)

```
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, a FmcException will be thrown.
// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
               AbsenceIndicator.LEAVE );
System.out.println("Logon successful");
```

Figure 38. Sample Java program to query work items from a worklist (Part 3 of 8)

```
// 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());
```

Figure 38. Sample Java program to query work items from a worklist (Part 4 of 8)

```
        // 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*/
```

Figure 38. Sample Java program to query work items from a worklist (Part 5 of 8)

```
      // 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 38. Sample Java program to query work items from a worklist (Part 6 of 8)

```
 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();
    }
```

Figure 38. Sample Java program to query work items from a worklist (Part 7 of 8)

```
   catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 38. Sample Java program to query work items from a worklist (Part 8 of 8)

Chapter 72. 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 executable (C-language)

```
#include <stdio.h>
#include <fmcjccon.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);
```

Figure 39. Sample activity implementation (C-language) (Part 1 of 4)

```
  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));
```

Figure 39. Sample activity implementation (C-language) (Part 2 of 4)

```
/*-- 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));
```

Figure 39. Sample activity implementation (C-language) (Part 3 of 4)

```
/*----- 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 MQ Workflow
}
```

Figure 39. Sample activity implementation (C-language) (Part 4 of 4)

Programming an executable (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;
```

Figure 40. Sample activity implementation (C++) (Part 1 of 4)

```
  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;
```

Figure 40. Sample activity implementation (C++) (Part 2 of 4)

```
/*-- 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;

  ...
```

Figure 40. Sample activity implementation (C++) (Part 3 of 4)

```
/*-- 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 MQ Workflow
}
```

Figure 40. Sample activity implementation (C++) (Part 4 of 4)

```
import com.ibm.workflow.api.*;

// Various needed classes

import java.io.*;
import java.util.*;

public class ActivityImplementation {
  public static PrintWriter out;

  public static void main(String args[])
  {

    try
    { out = new PrintWriter(
                new BufferedWriter(new FileWriter("ActivityImplementation.log")));
    }
    catch (IOException e) {}

    // Maximum nesting depth is 10
    ContainerElement[][] members = new ContainerElement[10][];
    // Maximum element number is 200
    ContainerElement[][] leaves = new ContainerElement[200][];

    String membername;
    String membertype;

    String strvalbuf;
    byte[] binvalbuf = { 0,1,2,3,4,5,6,7,8,9,10 };
    int lvalbuf      = 0;
    double fvalbuf   = 0.0;
    String tInfo;
    int j = 0, k = 0, l = 0;
    int index = 0;
     Calendar ltime = Calendar.getInstance();

  int param = -1;
  int pimreturn = 0;

  String setstr ="";
  int setlong = 0;
  double setdbl = 0;

  ltime.setTime(new Date());

  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  boolean again = false;
  String choice;
```

Figure 41. Sample activity implementation (Java) (Part 1 of 25)

```
/*-------------------------------------------------------------------*/
/* Check command line parameters                                     */
/*-------------------------------------------------------------------*/

  if (args.length!=1)
  {
    System.out.println("\nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>\n"
                       + "-rx: x specifies the return code of the program\n\n"
                       + "Program output is written to ActivityImplementation.log\n");
    System.exit(-1);
  }

  try { // general try clause for whole example

  Agent agent = new Agent();
  agent.setLocator(Agent.LOC_LOCATOR);
  agent.setName("LOCAL");

  ExecutionService service = agent.locate("","");
  System.out.println();

  if (args[0].equals("-r0")) param=0;
  if (args[0].equals("-r1")) param=1;
  if (args[0].equals("-r2")) param=2;
  if (args[0].equals("-r3")) param=3;
  if (args[0].equals("-r4")) param=4;
  if (args[0].equals("-r5")) param=5;

  switch (param)
  {
   case -1:
   System.out.println("\nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>\n"
                      + "-rx: x specifies the return code of the program\n\n"
                      + "Program output is written to ActivityImplementation.log\n");
   System.exit(-1);
   break;
   case 0: pimreturn = 0; break;
   case 1: pimreturn = 1; break;
   case 2: pimreturn = 2; break;
   case 3: pimreturn = 3; break;
   case 4: pimreturn = 4; break;
   case 5: pimreturn = 5; break;
  }

  out.write("\n\n" + "-------------------------------------------------------\n"
        + "API Tutorial - Activity implementation example program.\n"
        + "Called at : " + ltime.getTime().toString() + "\n"
        + "Desired RC: " + pimreturn + "\n\n");
```

Figure 41. Sample activity implementation (Java) (Part 2 of 25)

```
/*********************************************************************/
/* First we cope with the activity's INPUT container:                */
/*  - get the container                                              */
/*  - get the leaves of the container                                */
/*  - display the member values                                      */
/*********************************************************************/

/*----- Get Input Container ------------------------------------------*/

  ReadOnlyContainer inctnr;
  ExecutionAgent eAgent = agent.getExecutionAgent();

  inctnr = eAgent.inContainer();

  out.write("\nReceived input container '" + inctnr.type() + "'\n");
  System.out.println("\nReceived input container '" + inctnr.type() + "'\n");

/*----- Get Leave Count ----------------------------------------------*/

  int ulLeafCount = inctnr.allLeafCount();

  out.write("Input container AllLeafCount() =  " + ulLeafCount + "\n");

/*----- Get the Leaves -----------------------------------------------*/

  leaves[0]=inctnr.allLeaves();

  if (ulLeafCount != leaves[0].length)
  {
    out.write("LeafCount - vector size mismatch: "
          + "ulLeafCount = " + ulLeafCount
          + " size = "       + leaves[0].length + "\n");
  }
```

Figure 41. Sample activity implementation (Java) (Part 3 of 25)

```
/*----- Show the data members ------------------------------------------*/

  out.write("Input container leaves:\n");

  ContainerElement element = leaves[0][0];

  for (j=0; j < ulLeafCount; j++)
  {
    membername = element.fullName();
    membertype = element.type();

    if (membertype.equals("STRING"))
    {
      try
      {
        strvalbuf = element.getString();
        out.write("STRING '" + membername + "'= " + strvalbuf  + "\n");
      }
      catch (FmcException e)
      {
       switch (e.rc)
       {
         case FmcException.FMC_ERROR_MEMBER_NOT_SET:
           out.write("STRING '" + membername  + "'= <not set>" + "\n");
           break;
         default:
           out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
           break;
       }
      }
    } /* end if STRING */
```

Figure 41. Sample activity implementation (Java) (Part 4 of 25)

```
if (membertype.equals("LONG"))
{
  try
  {
    lvalbuf = element.getLong();
    out.write("LONG '" + membername + "'= " + lvalbuf  + "\n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("LONG '" + membername  + "'= <not set>" + "\n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
       break;
   }
  }
} /* end if LONG */
```

Figure 41. Sample activity implementation (Java) (Part 5 of 25)

```
if (membertype.equals("FLOAT"))
{
  try
  {
    fvalbuf = element.getDouble();
    out.write("FLOAT '" + membername + "'= " + fvalbuf  + "\n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("FLOAT '" + membername  + "'= <not set>" + "\n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
       break;
   }
  }
} /* end if FLOAT */
```

Figure 41. Sample activity implementation (Java) (Part 6 of 25)

```
    if (membertype.equals("BINARY"))
    {
      try
      {
        binvalbuf = element.getBuffer();
        out.write("BINARY '" + membername + "'= <not shown>\n");
      }
      catch (FmcException e)
      {
       switch (e.rc)
       {
         case FmcException.FMC_ERROR_MEMBER_NOT_SET:
           out.write("BINARY '" + membername  + "'= <not set>" + "\n");
           break;
         default:
           out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
           break;
       }
      }
     } /* end if BINARY */

    element = leaves[0][j];

  } /* end for */
```

Figure 41. Sample activity implementation (Java) (Part 7 of 25)

```
/*********************************************************************/
/* Now we process the activity's OUTPUT container, but only if the   */
/* program was not started with the -s switch                        */
/*  - get the container                                              */
/*  - navigate through the ContainerElement levels to the leaves     */
/*  - modify the member values                                       */
/*  - send the container to the server                               */
/*********************************************************************/

/*----- Get Output Container ------------------------------------------*/

  ReadWriteContainer outctnr = eAgent.outContainer();

  out.write("Received output container '" + outctnr.type() + "'\n");
  System.out.println("Received output container '" + outctnr.type() + "'");


/*----- Navigate through the ContainerElement structures ---------------*/

  int level            = 0;  // Current nesting level
  int start            = 0;  // Start "stack pointer" of vectors
  int current          = 0;  // End "stack pointer" of vectors
  boolean AllLeavesReached = false;

  // Get the number of 1st level container elements

  int ulMemberCount = outctnr.memberCount();

  if (ulMemberCount == 0)       // No data structure at all, error?
  {
    AllLeavesReached = true;
  }
  else                          // Get the vector of 1st level container elements
  {
    members[level] = outctnr.structMembers();
  }

  while (AllLeavesReached == false)
  {
    out.write("Number of members: " + ulMemberCount + "\n");

    // Check consistency

    if (members[level].length != ulMemberCount)
    {
      out.write("MemberCount - Vector Size mismatch: "
            + "ulMemberCount = " + ulMemberCount
            + " Size = "         + members[level].length + "\n");
    }

    element = members[level][0];

    out.write("IsStruct-IsLeaf-IsArray-Cardinality-Membername\n");
```

Figure 41. Sample activity implementation (Java) (Part 8 of 25)

```
for (j=0; j < ulMemberCount; j++)
    {
      out.write("       " + element.isStruct());

      if (element.isStruct())
      {
        // Put the next nesting level of this data structure "on the stack"
        leaves[current] = element.structMembers();
        current++;
      }

      out.write("      " + element.isLeaf() + "       " + element.isArray()
            + "           " + element.cardinality() + " "
            + element.fullName() + "\n");

      element = members[level][j];

    } /* end for */

    if (start >= current)        // Nothing "on the stack"
    {
      AllLeavesReached = true;
    }
    else
    {
      level++;
      // Get the next vector from the stack and increase stack pointer
      members[level] = leaves[start];
      start++;
      ulMemberCount = members[level].length;
    }

  } /* end while */
```

Figure 41. Sample activity implementation (Java) (Part 9 of 25)

```
/*----- Modify the data members ----------------------------------------*/

  out.write("\n\nSetting the data members...\n");
  System.out.println("\nSetting the data members...\n");

  leaves[0] = outctnr.leaves();

  ulLeafCount = leaves[0].length;

  try
  {

    for (j=0; j < ulLeafCount; j++)
    {
      element = leaves[0][j];

      membername = element.fullName();
      membertype = element.type();

      if (membertype.equals("STRING"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
               // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
```

Figure 41. Sample activity implementation (Java) (Part 10 of 25)

```
    else
    {
      try
      {
        index=
        Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                         membername.indexOf(")")-1));
      }
      catch (NumberFormatException e) {}
    }

    System.out.print("Enter a value for STRING array member '"
         + membername + "[" + index + "]'? [y/n] ");

    setstr = in.readLine();
```

Figure 41. Sample activity implementation (Java) (Part 11 of 25)

```
    if (setstr.equalsIgnoreCase("y"))
    {
      System.out.print("Value: ");
      setstr = in.readLine();

      outctnr.setString2(membername,index,setstr);
      strvalbuf = outctnr.getString2(membername,index);

      out.write("Setting ArrayStringValue '" + membername + "[" + index +
                "]'=" + strvalbuf + "\n");
      System.out.println("Setting ArrayStringValue '" + membername + "[" +
                         index + "]' = " + strvalbuf + "\n");

    }
  }
```

Figure 41. Sample activity implementation (Java) (Part 12 of 25)

```
    else
    {

      System.out.print("Enter a value for STRING member '"
           + membername + "'? [y/n] ");
      setstr = in.readLine();

      if (setstr.equalsIgnoreCase("y"))
      {
        System.out.print("Value: ");
        setstr = in.readLine();

        outctnr.setString(membername,setstr);
        strvalbuf = outctnr.getString(membername);

        out.write("Setting StringValue '" +
                    membername + "'= " + strvalbuf + "\n");
        System.out.println("Setting StringValue '" + membername +
                           "'= " + strvalbuf + "\n");
      }
    }
  } /* end if STRING */
```

Figure 41. Sample activity implementation (Java) (Part 13 of 25)

```
if (membertype.equals("LONG"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
          // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }

          System.out.print("Enter a value for LONG array member '"
               + membername + "[" + index + "]'? [y/n] ");

          setstr = in.readLine();
```

Figure 41. Sample activity implementation (Java) (Part 14 of 25)

```
    if (setstr.equalsIgnoreCase("y"))
   {
     do
     {
       again=false;
       System.out.print("\nValue: ");
       choice = in.readLine();
       try { setlong = Integer.parseInt(choice); }
       catch (NumberFormatException e) { again=true; }
        } while (again);

     outctnr.setLong2(membername,index,setlong);
     lvalbuf = outctnr.getLong2(membername,index);

     out.write("Setting ArrayLongValue '" + membername + "[" + index +
               "]' = " + lvalbuf + "\n");
     System.out.println("Setting ArrayLongValue '" + membername + "[" +
                        index + "]' = " + lvalbuf + "\n");
   }
 }
```

Figure 41. Sample activity implementation (Java) (Part 15 of 25)

```
else
 {

   System.out.print("Enter a value for LONG member '"
        + membername + "'? [y/n] ");

   setstr = in.readLine();

   if (setstr.equalsIgnoreCase("y"))
   {
     do
     {
       again=false;
       System.out.print("\nValue: ");
       choice = in.readLine();
       try { setlong = Integer.parseInt(choice); }
       catch (NumberFormatException e) { again=true; }
        } while (again);

     outctnr.setLong(membername,setlong);
     lvalbuf = outctnr.getLong(membername);

     out.write("Setting LongValue '" +membername+ "' = " +lvalbuf+ "\n");
     System.out.println("Setting LongValue '" +membername+"' = " +
                        lvalbuf + "\n");
   }
 }
} /* end if LONG */
```

Figure 41. Sample activity implementation (Java) (Part 16 of 25)

```
if (membertype.equals("FLOAT"))
     {
       if ( membername.endsWith("]") || membername.endsWith(")") )

       // Array element, take the array function
       {
              // Remember the index and remove the suffix from the name

         if (membername.endsWith("]"))
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                              membername.indexOf("]")-1));
           }
           catch (NumberFormatException e) {}
         }
         else
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                              membername.indexOf(")")-1));
           }
           catch (NumberFormatException e) {}
         }

         System.out.print("Enter a value for FLOAT array member '"
              + membername + "[" + index + "]'? [y/n] ");

         setstr = in.readLine();
```

Figure 41. Sample activity implementation (Java) (Part 17 of 25)

```
  if (setstr.equalsIgnoreCase("y"))
  {
    do
    {
      again=false;
      System.out.print("\nValue: ");
      choice = in.readLine();
      try { setdbl = (Double.valueOf(choice)).doubleValue(); }
      catch (NumberFormatException e) { again=true; }
       } while (again);

    outctnr.setDouble2(membername,index,setdbl);
    fvalbuf = outctnr.getDouble2(membername,index);

    out.write("Setting ArrayFloatValue '" + membername + "[" + index +
              "]' = " + fvalbuf + "\n");
    System.out.println("Setting ArrayFloatValue '" + membername + "[" +
                       index + "]' = " + fvalbuf + "\n");
  }
}
```

Figure 41. Sample activity implementation (Java) (Part 18 of 25)

```
else
   {
     System.out.print("Enter a value for FLOAT member '"
          + membername + "'? [y/n] ");

     setstr = in.readLine();

     if (setstr.equalsIgnoreCase("y"))
     {
       do
       {
         again=false;
         System.out.print("\nValue: ");
         choice = in.readLine();
         try { setdbl = (Double.valueOf(choice)).doubleValue(); }
         catch (NumberFormatException e) { again=true; }
          } while (again);

       outctnr.setDouble(membername,setdbl);
       fvalbuf = outctnr.getDouble(membername);

       out.write("Setting FloatValue '" +membername+ "' = " +fvalbuf+ "\n");
       System.out.println("Setting FloatValue '" + membername + "' = " +
                          fvalbuf + "\n");
     }
   }
 } /* end if FLOAT */
```

Figure 41. Sample activity implementation (Java) (Part 19 of 25)

```
if (membertype.equals("BINARY"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
               // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }

          outctnr.setBuffer2(membername,index,binvalbuf);
          binvalbuf = outctnr.getBuffer2(membername,index);

          out.write("Setting ArrayBinaryValue '" + membername + "[" + index +
                    "]' = <not shown>\n");
          System.out.println("Setting ArrayBinaryValue '" + membername + "[" +
                             index + "]' = <not shown>\n");
        }
```

Figure 41. Sample activity implementation (Java) (Part 20 of 25)

```
        else
        {
          outctnr.setBuffer(membername,binvalbuf);
          binvalbuf = outctnr.getBuffer(membername);

          out.write(
          "Setting BinaryValue '" + membername + "' = <not shown>\n");
          System.out.println(
          "Setting BinaryValue '" + membername + "' = <not shown>\n");
        }
      } /* end if BINARY */

    } /* end for */
  }/* End try*/
  catch (FmcException e)
  {
  }/* End catch*/
```

Figure 41. Sample activity implementation (Java) (Part 21 of 25)

```
/*--------------------------------------------------------------------*/
/* Send output container to PEA                                       */
/*--------------------------------------------------------------------*/
  System.out.println("\nSetting output container.");
  out.write("\nSetting output container.\n");
  eAgent.setOutContainer(outctnr);
```

Figure 41. Sample activity implementation (Java) (Part 22 of 25)

```
/*--------------------------------------------------------------------*/
/* Logoff and deinit the API environment.                             */
/*--------------------------------------------------------------------*/

  System.out.println("\nLogging off.");
  out.write("\nLogging off.\n");
  service.logoff();
  out.flush();
  out.close();
  System.exit(pimreturn);

  } // end of general try clause
```

Figure 41. Sample activity implementation (Java) (Part 23 of 25)

```
  catch(FmcException e)
  {
    int c;

    // 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 ( c = 0; c < e.parameters.length ; c++)
    {
      System.out.println("    " + e.parameters[c] );
    }
    System.out.println("  StackTrace : ");
    e.printStackTrace();

    out.write("FmcException occured");
    out.write("\n  RC         : " + e.rc);
    out.write("\n  Origin     : " + e.origin);
    out.write("\n  MessageText: " + e.messageText);
    out.write("\n  Exception  : " + e.getMessage());
    out.write("\n  Parameters : ");
    for ( c = 0; c < e.parameters.length ; c++)
    {
      out.write("    " + e.parameters[c] );
    }
    out.write("\n  StackTrace : ");
    e.printStackTrace(out);
    out.flush();
    out.close();

  }
```

Figure 41. Sample activity implementation (Java) (Part 24 of 25)

```
  catch(Exception e)
  {
    // Catch and report any exception that occurred.
    e.printStackTrace();
    e.printStackTrace(out);
    out.flush();
    out.close();
  }

} // end of void main

} // end of ActivityImplementation
```

Figure 41. Sample activity implementation (Java) (Part 25 of 25)

Part 10. Appendixes

Appendix A. FlowMark Version 2 compatibility mode

Note: The FlowMark Version 2 interface will no longer be supported with the next release or version.

The MQ Workflow APIs support a FlowMark Version 2.3 API compatibility mode that allows you to run FlowMark Version 2.3 programs. It is, however, recommended that you replace the Version 2 API calls with the MQ Workflow Version 3 API calls in all your applications.

The following languages and compilers are supported in compatibility mode:

- The C-language API
  - For AIX and the IBM C++ Professionell for AIX Version 5.0
  - For Windows NT or Windows 98 and IBM VisualAge for C++ 3.5 or Microsoft Visual C++ 5.0
- The C++ language API
  - For AIX and the IBM C++ Professionell for AIX Version 5.0
  - For HP-UX and the HP aC++ Compiler S700 Version A.01.15.01
  - For Windows NT or Windows 98 and IBM VisualAge for C++ 3.5 or Microsoft Visual C++ 5.0
- The VisualBasic API
  - For Windows NT or Windows 98 and Microsoft VisualBasic 5.0

Repeating the compile-and-link step should be sufficient to make your Version 2.3 programs run.

MQ Workflow Version 3 contains new header, library and dynamic link libraries for this purpose. The header files have the FlowMark Version 2.3 names so that you do not have to change your source code. The library and dynamic link libraries have new names. You can, however, choose to (DLL) rename these files to their Version 2 names; otherwise, you have to adapt your link step to the new names. The following table provides an overview on the compatibility API and the files to include and link with:

Table 4. FlowMark Version 2 Compatibility APIs on AIX

Language	**AIX**		
	header	**lib**	**DLL**
C	exmajapc.h	fmcjdapc.lib	libfmcjdapc.a
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	libfmcjdcom.a libfmcjdcbr.a libfmcjdrun.a libfmcjdcon.a

Table 5. FlowMark Version 2 Compatibility APIs on HP-UX

Language	HP-UX		
	header	**lib**	**DLL**
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	libfmcjdcom.sl libfmcjdcbr.sl libfmcjdrun.sl libfmcjdcon.sl

Table 6. FlowMark Version 2 Compatibility APIs on Windows NT/98

Language	Windows NT/98		
	header	**lib**	**DLL**
C	exmwjapc.h	fmcjdapc.lib	fmcjdapc.dll
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	fmcjdcom.dll fmcjdcbr.dll fmcjdrun.dll fmcjdcon.dll
VisualBasic	exmbjapv.bas		fmcjdapv.dll

Because MQ Workflow Version 3 has an extended functionality and flexibility, there are some deviations related to the API. You must be aware of these differences, which can influence your Version 2 program behaving differently. This applies mainly to return codes and authorization definitions.

Deviations from FlowMark Version 2

- Version 3 provides for more detailed states than Version 2 and, therefore, these states are mapped on a best-can-do basis:

  When a work item is to be started and the program execution agent is not running or when the program to be started is not found, the work item goes into state *InError*. The InError state is exposed as a Version 2 *Running* state so that a Version 2 Restart() or Terminate() can be issued. Because of the actual InError state, a Version 3 ForceRestart() or ForceFinish() is called which, however, requires the caller to have process administration authority.

  When a work item is checked out, the work item goes into state *CheckedOut*. The CheckedOut state is exposed as a Version 2 *Running* state so that a Version 2 Restart() or Terminate() can be issued. Because of the actual CheckedOut state, a Version 3 ForceRestart() or ForceFinish() is called which, however, requires the caller to have process administration authority.

  When a manual exit work item has executed its activity implementation, it is set to state *Executed*. This is mapped to the Version 2 *Ready* state to show that the program has executed once and can be finished (called ManualExit() in Version 2).

- Version 3 supports an authorization concept that is more restricted than the Version 2 concept:

  Process administration authority is needed for work item ForceFinish() and ForceRestart(). This is also needed for the Version 2 C-language ChangeActivityState() function when you request finish or restart.

  Authorization changes become active at once. This is because authorization is checked by the server only. The UserSettings() method returns the settings of the user at the time when called, that is, it fetches the current user authorizations from the server every time it is called. In Version 2, the user settings were retrieved once when logging on.
- Version 3 return codes are mapped to Version 2 return codes on a best-can-do basis:

  ERROR_TIMEOUT is returned when the client does not receive an answer within the specified time. This can also mean that the server is not running.
- The scope or database and server specifications are case-sensitive. They are not folded to uppercase. The scope or database corresponds to the system group specification in Version 3; the server to the system specification.
- The password is case-sensitive.
- Logon() allows for a specification of the absence setting. If it is not set (as in the compatibility mode), the absence behavior - whether the absence information is to be reset or not - is taken from the person record.
- If the MQ Workflow server allows for a *unified logon*, an empty password and user ID are accepted (see also "Logon()" on page 409).
- Logon() allows for a specification of the session mode. If it is not set (as in the compatibility mode), the session mode is taken from the user or configuration profile. If it is not found there, session mode *present* is used. Only one present session can exist per user.

  If an application that is using the present session mode ends abnormally during the application test phase, a session record is still held on the server. You then have to wait with a new logon attempt until that session expires or to log on with the *present here* session mode.

  Present here forces any other present session for the user logoff. Such, present here especially forces any pending present session logoff. If you rely on the *already logged on* return code, do not set any session mode or set the *present* mode in the profile. *Default* allows for multiple parallel sessions per user.

  You can use the fmczchk utility to set or erase profile values; the key is V2_SESSION_MODE; the values are DEFAULT, PRESENT, or PRESENTHERE. For example, to set a default session mode in configuration FMC, issue:

  ```
  fmczchk -y FMC -c inst:m,V2_SESSION_MODE,DEFAULT
  ```

- IsDeleteFinishedItems() always returns false since this setting has been moved from the user to the process. The attempt to change this setting returns FMC_OK but nothing is changed.
- The PersonsAuthorizedFor() method does not return any persons when the logged-on user is authorized for all persons.
- SetPersonsToStandInFor() requires staff authorization.
- When the logged-on user is an administrator for all categories, then IsAdminForCategory(x) returns true even if the category does not exist.
- Process templates are versioned in MQ Workflow. This means that a process template can be no longer valid, which is defined as *invalid*.
- Creation of a process instance returns primary values only.
- Process instance names are generated differently. The process template name is no longer padded by _n but by a $ sign followed by an object ID representation so that names are unique.
- Process instance input container and output container names are secondary attributes, that is, the process instance must be refreshed before they can be read.
- The IsTerminatedOnError() method on process templates or process instances always returns false.
- MQ Workflow introduces the concept of autonomy of subprocesses. Only non-autonomous subprocesses with respect to control autonomy are suspended or resumed when the deep option is specified. Suspend() of a ready process instance is not supported.
- Creation of a worklist does not check whether the owner of the items contained is a registered user or whether you are authorized to see the items of that user. The worklist is created anyhow. The item owner is part of the filter in Version 3 so that the owner is only applied when the worklist content, that is, the items, are queried. If you specified an unknown item owner or if you are not authorized to see the items of the specified owner, you will not see any items at all.
- The last modification time of a work item is changed even if only the description of the work item changes.
- To check out a work item is possible only if the corresponding MQ Workflow setting defines that checking out is allowed. If not, FMC_ERROR_CHECKOUT_NOT_POSSIBLE is returned.
- Deletion of a ready work item is allowed as long as it is not the last work associated with the activity instance. You can always delete ready work items in a terminated or terminating process instance.
- Version 3 allows for a priority setting between 0 and 999 (9 in FlowMark Version 2). The compatibility API allows for a filter criterion with a MAX_PRIORITY specification of 999 so that objects with a priority greater than 9 can be searched for.

- When a notification duration is not set, then ***no*** notification occurs. If the person to notifiy is not supplied or is unknown, the process administrator is notified.
- Finish() on a notification is supported by an implementation returning FMC_OK. This means that notifications must be deleted explicitly.
- Input and output containers are only sent to the program execution agent when they are accessed by the activity implementation or support tool. This behavior can be set in Version 3.
- The program execution agent provides the program identification (called session ID in Version 2) only to *trusted* programs. This property can be set in Version 3.
- Passthrough() cannot be called from a support tool.
- The maximum size of a container passed between the client and the server can be 32KB.
- A container can have leaves of binary data types.
- ExmcChangeActivityState() performs the finish action without first checking for a requested ready or running state. It currently requires that the activity name is unique within a process instance, which means it does not support unique activity names within blocks. It returns EXMPJ_WRONG_ACTIVITY_STATE also if the process is not running.
- Bundles are not yet supported.

FlowMark Version 2 C-language programs

Running an existing application program

MQ Workflow Version 3 is delivered so that FlowMark Version 2 application programs can run unchanged. See "Deviations from FlowMark Version 2" on page 828 for possible increased authorization requirements. A compile and link should be sufficient to make your Version 2 programs run.

If you want to run an existing FlowMark Version 2 C-language application program in an MQ Workflow Version 3 environment:

- Make sure that MQ Workflow Version 3 paths are searched so that the new header file for Version 3 is used:
  - *exmwjapc.h* for Windows NT or Windows 95
  - *exmajapc.h* for AIX
- Change your application build step to link with the MQ Workflow Version 3 API library *fmcjdapc.lib* instead of the FlowMark Version 2 library
- Compile and link your application

FlowMark Version 2 Visual Basic programs

Running an existing application program

MQ Workflow Version 3 is delivered so that FlowMark Version 2 application programs can run unchanged. See "Deviations from FlowMark Version 2" on page 828 for possible increased authorization requirements.

If you want to run an existing FlowMark Version 2 Visual Basic application program in the MQ Workflow Version 3 environment, make sure that:

- MQ Workflow Version 3 paths are searched so that the new Version 3 provided declarations *exmbjapv.bas* are used
- The MQ Workflow Version 3 dynamic link library fmcjdapv.dll is in a directory in your PATH statement

FlowMark Version 2 C++ programs

Running an existing application program

MQ Workflow Version 3 is delivered so that FlowMark Version 2 application programs can run unchanged. See "Deviations from FlowMark Version 2" on page 828 for possible increased authorization requirements. A re-compile and re-link should be sufficient to make your Version 2 programs run.

If you want to run an existing FlowMark Version 2 C++ application program in an MQ Workflow Version 3 environment:

- Make sure that MQ Workflow Version 3 paths are searched so that the new Version 3 provided *exmpjapi.hxx* header file is used

  Note: Since the FlowMark Version 2 exmpjapi.hxx header file is self-sufficient, you should have included no other FlowMark header files in your application. If this is not the case, delete all other inclusions.
- Change your application build step to link with the MQ Workflow Version 3 API libraries *fmcjdcom.lib*, and *fmcjdcbr.lib, fmcjdrun.lib* and/or *fmcjdcon.lib* instead of the FlowMark Version 2 library, see "Chapter 16. Compiling and linking" on page 163
- Compile and link your application

Using MQ Workflow Version 3 methods

If you want to extend an existing FlowMark Version 2 C++ application program in order to use the new methods provided with the MQ Workflow Version 3 API, then you should migrate your application first. This is because there are some extensions and deviations from FlowMark Version 2.

Migration is done according to the following list:

1. **General steps to be done**

   Note: You should follow the sequence of steps illustrated below since some of the global change steps base on each other.

   Header file inclusion:

```
#include <bool.h>                    // true, false  (dependent inclusion)
#include <fmcjstr.hxx>               // string       (dependent inclusion)
#include <vector.h>                  // vector       (dependent inclusion)
#include <fmcjprun.hxx>              // C++ runtime client interface
   or
#include <fmcjpcon.hxx>              // C++ container interface
```

   a. Include the MQ Workflow Version 3 C++ Runtime API header file *fmcjprun.hxx* or *fmcjpcon.hxx* instead of the FlowMark Version 2 header file exmpjapi.hxx.

      Note: Because the FlowMark Version 2 exmpjapi.hxx header file is self-sufficient, you should have included no other FlowMark header files in your application. If not so, delete all other inclusions.

   b. Conditionally include bool.h before fmcjprun.hxx or fmcjpcon.hxx.

      If your compiler does not support any bool definition, include this MQ Workflow delivered definition of bool. Otherwise, use the bool definition of your compiler.

      Note: bool.h must be included before your string definition file.

   c. Conditionally include fmcjstr.hxx before fmcjprun.hxx or fmcjpcon.hxx.

      If your compiler does not support any string class, include this MQ Workflow delivered definition of a string class. Otherwise, include your compiler string definition file.

   d. Conditionally include vector.h before fmcjprun.hxx.

      If your compiler does not support any vector, include this MQ Workflow delivered definition of a vector. Otherwise, include your compiler vector definition file.

   Names of return/error codes:

   e. Change all EXM_API_OK occurrences to FMC_OK. Change all EXM_API_ERROR occurrences to FMC_ERROR.

      After this step, return/error codes are named correctly.

   Names of classes:

   f. Change all Exm occurences to Fmcj. Change the resulting FmcjServer references to FmcjExecutionService. Change the resulting FmcjWorkitemNotification references to

FmcjActivityInstanceNotification. Change the resulting FmcjItemBase references to FmcjItem. Change the resulting FmcjUser references to FmcjPerson.

After this step, classes are named correctly.

Names of methods:

g. Change all GetXxx() method names to Xxx() **except** the GetElement() method which does not change its name. Change all ChangeXxx() method names to SetXxx().

This means that accessor methods are consistently named according to the data member name; mutator methods have the prefix "Set".

Change all EndCondition() method calls to ExitCondition().
Change all ExecutionSessionID() method calls to ProgramID().
Change all PersistentHandle() method calls to PersistentOid().
Change all ReadPersistentObject() method calls to PersistentObject().
Change all Organization() method calls to OrganizationName().
Change all Roles() method calls to NamesOfRoles().
Change all RolesToCoordinate() method calls to NamesOfRolesToCoordinate().
Change all QueryWorkitemNotifications() method calls to QueryActivityInstanceNotifications().

This is because of Version 3 C++- and C-language API cleanup and compatibility. The PersistentHandle() method name has been changed to avoid confusion with C-API handles.

After this step, all supported methods are named correctly.

2. **Mandatory specific steps**

These are steps which you **must** carry out if you use the named class and method.

- **FmcjActivityInstanceNotification**

The IsEscalated() method has been changed to StateOfNotification() returning the exact state of escalation as an enumeration.

The information formerly queried via the IsFirstEscalatation(), and IsSecondEscalation() methods can be accessed by using the Kind() method. The Kind() method returns the exact type of an item as an enumeration.

The IsProcessType() and IsProgramType() methods have been changed to ActivityKind() returning the kind of the associated activity instance as an enumeration.

The NotificationTime() method has been changed to FirstNotificationTime() and SecondNotificationTime() so that both times can be queried.

The ManualExit() and ManualStart() accessor methods return false (the boolean default) as long as the object is not complete.

The Finish() method is no longer needed, that is, it has to be removed.

- **FmcjContainer**

  ActivityInfo() and ProcessInfo() methods need to be removed; it is no longer necessary to call these methods before accessing their data members. Specifying their fully qualified names when querying a container is sufficient.

  Activity implementations do not need to pass their program identifications to the program execution agent when they are dealing with their input or output containers. This means, that either the program ID parameter has to be removed from the InContainer(), or OutContainer() calls, or the appropriate RemoteInContainer(), or RemoteOutContainer() methods have to be called.

- **FmcjExecutionService** respectively **FmcjService**

  Constructor FmcjExecutionService(systemGroup) - ExmServer(scope) - is no longer supported since you always connect to either a specific system or your home system.

  The Name() method has to be replaced by SystemName().

  The Scope() method has to be replaced by SystemGroupName().

  The user identification is case-sensitive. It is no longer folded to uppercase when logging on.

  The CreateWorklist() method needs to specify the additional parameters for persistent lists, namely worklist owner, type, description, sort criteria, and threshold. The filter attribute has become a string and the owner of the workitems is part of the filter criterion.

  The QueryProcessTemplates() and QueryProcessInstances() methods newly allow for specifying sort criteria, and thresholds. The filter attribute has become a string.

  The UserSettings() method has become an action method returning an APIRET value. In FlowMark Version 2, user information was provided as the result of a successful Logon() request. In MQ Workflow Version 3, user settings have to be queried explicitly.

  Passthrough() does no longer need to pass a program identification. This means that either the program ID parameter has to be removed or the RemotePassthrough() method has to be called.

- **FmcjFilter**

  To provide for increased flexibility and extentability, the FmcjFilter class has been removed. A string containing the filter expression has to be provided instead of the FmcjFilter object.

- **FmcjItem**

The ItemType enumeration ″WorkitemNotification″ has become more specific and is split into FirstActivityInstanceNotification and SecondInstanceActivityNotification. Such, any check on an item whether it is a notification has to be replaced with an appropriate or-statement to check whether it is a first or second notification.

Methods ManualExit(), ManualStart(), ExitCondition(), StartCondition(), Priority(), and Staff() are not applicable for process instance notifications. Thus, they have been moved from the FmcjItem class to the FmcjWorkitem and FmcjActivityInstanceNotification classes. This means that you can only call them on objects of the respective kind.

ChangeDescription() has been renamed to SetDescription(). Since the description is now an optional parameter, a pointer to a string needs to be passed instead of a reference to a description.

- **FmcjPerson**

  The functionality to specify whether finished items are to be deleted has been moved from the logged-on user to the process models. Thus, any IsDeleteFinishedItems() and SetDeleteFinishedItems() calls need to be removed.

- **FmcjProcessInstance**

  The IsAudited() method has been changed to AuditMode() returning the exact type of auditing as an enumeration.

- **FmcjProcessInstanceNotification**

  The Expired() method has been removed since a process instance notification is only raised when the process instance is expired.

  The Finish() method is no longer needed, that is, it has to be removed.

- **FmcjProcessTemplate**

  The IsAudited() method has been changed to AuditMode() returning the exact type of auditing as an enumeration.

  The CreateInstance() and CreateAndStartInstance() methods have additional, still reserved parameters; at least 0 pointers need to be provided.

- **FmcjWorkitem**

  The IsEscalated() method has been changed to StateOfNotification() returning the exact state of escalation as an enumeration.

  The IsProcessType() and IsProgramType() methods have been changed to ActivityKind() returning the kind of the work item - inherited from the activity - as an enumeration.

  The NotificationTime() method has been changed to FirstNotificationTime() and SecondNotificationTime() so that both times can be queried.

  The ManualExit() and ManualStart() accessor methods return false (the boolean default) as long as the object is not complete.

The ManualExit() action method has been changed to Finish() to better fit to the resulting state.

The CheckIn() method does no longer require a work item output container. You must pass a pointer to your output container instead of the container itself.

The CheckOut() method potentially returns all information about the activity implementation known to MQ Workflow. You need to request the common data only and pick up your input container; from there in order to achieve the FlowMark Version 2 behavior.

- **FmcjWorklist**

  The filter is returned as a string instead of a Filter object.

  The IsDefault() method has been removed. There is only a worklist if you or someone else created one.

  The owner of the work items has become part of the filter specification; there may be multiple owners.

  The QueryWorkitems() method no longer allows for the specification of an ad-hoc filter. A persistent filter can be specified when a worklist definition is created. If you need to filter, either create the worklist with the appropriate filter or use the FmcjExecutionServive::QueryWorkitems() method which allows for specifying an ad-hoc filter.

- **CppStartApi** and **CppFinishApi** are no longer needed and need to be removed.

3. **Optional specific steps**

   These are steps which you can choose to execute or not.
   - FmcjItem::StartTime() If you used this method, it actually returned the creation time of the item. If you want to keep this semantics, change the method name to CreationTime(). StartTime returns the starting time.
   - Refresh If you used the refresh method to update relevant object values after an action, this is not always necessary. The object is automatically refreshed with the changed values as the result of calling an action method. For example, calling FmcjWorkitem::Start() updates the workitem's state.
   - FmcjPerson::CategoriesAuthorizedFor() This method only returns the categories for which you are authorized with basic rights. If you want to keep the FlowMark Version 2 behavior, then you have to add the CategoriesAuthorizedForAsAdmin().

Appendix B. 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.

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.

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 Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100
70569 Stuttgart
Germany

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 information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp.1993, 1999. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

- AIX
- CICS
- C Set++
- FlowMark
- IBM
- IMS
- MQSeries
- OS/2
- OS/390

- VisualAge

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, other countries, or both.

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

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

Glossary

This glossary defines important terms and abbreviations used in this publication. If you do not find the term you are looking for, refer to the index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

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.

application programming interface. An interface provided by the MQ Workflow workflow manager that enables programs to request services from the MQ Workflow workflow manager. The services are provided synchronously.

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

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.

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.

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.

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*.

fork activity. An activity that is the source of multiple control connectors.

form. In Lotus Notes, a form controls how you enter information into Lotus Notes and how that information is displayed and printed.

formula. In Lotus Notes, a mathematical expression that is used, for example, to select documents from a database or to calculate values for display.

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*.

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.

message queuing. A communication technique that uses asynchronous messages for communication between software components.

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.

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 imported and 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 imported and 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. The MQ Workflow component that manages the implementations of program activities, such as .EXE and .DLL files.

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.

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.

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

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.

U

user ID. An alphanumeric string that uniquely identifies an MQ Workflow user.

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 assigned to a user and retrieved from a workflow management system.

worklist view. List of work items and notifications 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 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 MQ Workflow.
- *IBM MQSeries Workflow: Concepts and Architecture*, GH12-6285, explains the basic concepts of MQ Workflow. It also describes the architecture of MQ Workflow and how the components fit together.
- *IBM MQSeries Workflow: Getting Started with Buildtime*, SH12-6286, describes how to use Buildtime of MQ Workflow.
- *IBM MQSeries Workflow: Getting Started with Runtime*, SH12-6287, describes how to get started with the MQ Workflow 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 MQ Workflow.
- *IBM MQSeries Workflow: Administration Guide*, SH12-6289, explains how to administer an MQ Workflow system.

Related publications

- *IBM MQSeries: Application Programming Guide*, SC33-0807.
- *IBM MQSeries: System Administration*, SC33-1873.
- *Frank Leymann, Dieter Roller, Production Workflow: Concepts and Techniques* (New Jersey: Prentice Hall PTR, 1999)
- *Frank Leymann, Dieter Roller, "Workflow-based Applications", IBM Systems Journal 36, no. 1 (1997): 102–123*, - you can also refer to the Internet: http://www.almaden.ibm.com/journal/sj/361/leymann.html
- *Workflow Handbook 1997* published in association with WfMC. Edited by Peter Lawrence.
- *Extensible Markup Language (XML) 1.0*, W3C Recommendation February 1998: http://www.w3.org/TR/REC-xml

Index

A

accessor API calls
  API calls 109
  bool 109
  char 137, 144
  date/time 110
  default values 106
  definition 106
  enumeration 111
  error handling 11
  integer 142
  IsNull 142
  lifetime of values 107
  long 136, 142, 145
  multi-valued 138
  object 145
  object valued 139, 140
  return codes 107
  string 137, 144
  vector 35
action API calls
  definition 150
  error handling 10
activation time 256
activity implementation
  activity instance 152
  API calls 151
  container 161, 181, 191
  error handling 10
  general information 152
  input container 369, 373
  Java High Performance Bridge 192
  output container 371, 376, 378, 380
  passthrough 416
  program identification 152
  pseudo code 161, 181, 191
  remote passthrough 472
  return code 162, 181, 191
  user identification 152
  XML 219
activity instance
  accessor API calls 256
  action API calls 259
  activation time 256
  activity implementation 152
  array methods 260
  assignment 256
activity instance *(continued)*
  basic API calls 255
  category 256
  comparison 255, 256
  completeness 255
  constructor 255
  copy 255
  creation 255
  deallocation 255
  definition 341
  description 256
  destructor 255
  documentation 256
  duplication 255
  empty 255
  end time 256
  enumeration, escalation 114
  enumeration, state 115
  enumeration, type 118
  error reason 257, 291
  exit condition 257
  exit mode 257
  expiration time 257
  finish, force 341
  first notification 257
  icon 257
  implementation 257
  input container 347
  input container, name 257
  kind 255
  last modification time 257
  last state change time 257
  monitor 80
  monitor, process instance 349
  name 257, 258
  notification 423
  notification state 259
  notification time 257, 258
  notified persons 257, 258
  object ID 258
  output container 352
  output container, name 258
  overview 255
  persons, notified 257, 258
  priority 258
  process administrator 258
  process instance name 258
  process instance state 258
  program, implementing 257
activity instance *(continued)*
  refresh 354
  restart, force 344
  retrieval 363, 624
  second notification 258
  staff 258
  start condition 259
  start mode 258
  start time 259
  starter 259
  state 259
  subprocess instance, retrieval 356
  support tools 259
  symbol layout 259
  system 258
  system group 258
  terminate 359
  vector API calls 263
activity instance notification
  accessor API calls 261, 296
  action API calls 262
  activity instance, kind 261
  activity instance, object ID 262
  activity instance, retrieval 363
  array methods 263
  assignment 261
  basic API calls 260
  category 297
  comparison 261
  constructor 260
  copy 260
  creation 260
  creation time 297
  deallocation 260
  definition 363
  delete 489
  description 297
  description, set 500
  destructor 260
  documentation 297
  duplication 260
  end time 297
  error reason 261
  exit condition 261
  exit mode 262
  expiration time 261
  first notification 261
  icon 297

activity instance notification *(continued)*
implementation 262
input container, name 297
kind 260
last modification time 297
monitor, process instance 492
name 298
name, set 503
notification state 262
notification time 261, 262
object ID 298
object identifier 363
output container, name 298
overview 260
owner 298
priority 262
process administrator 298
process instance 495
process instance, name 298
process instance, state 298
process instance ID 298
program, implementing 262
received reason 298
received time 298
refresh 498
second notification 262
staff 262
start condition 262
start mode 262
start time 298
start tool 366
state 262
support tools 262
system 298
system group 298
transfer 505
update 145, 298
agent
accessor API calls 264
API calls 264
basic API calls 264
bound 265
configuration ID 264
construction 264
context 265
listener 264
listener,remove 265
locator policy 265
name 265
overview 264
program execution agent 264
version 266
allocation
conversion 98
allocation *(continued)*
copy 99
declaration 94
explicit 167, 205
implicit 167, 205
APIENTRY 164
application
activity implementation 9, 161, 247
activity implementation, ActiveX 181
activity implementation, Java 191
client 9, 159, 247
client, ActiveX 179
client, Java 189
support tool 9, 161, 191, 247
support tool, ActiveX 181
array
ActiveX 40
activity instance notification 263
activity instances 260
container 270
container element 274
control connector instance 274
exceptions 40
execution serviceactivity instance notification 286
Java 43
point 306
process instance list 312
process instance notification 314
process template list 319
query result 34
string 329
work item 335
worklistactivity instance notification 336
assign reason 112
assignment 97
asynchronous protocol 19
audit setting 113
authentication
exit 27
authorization
definitions 83
explicit 83
implicit 83
process administrator 83
system administrator 83
XML message 216
authorization exit
activate 30
Autenticate() 28
authenticate() 28
authorization exit *(continued)*
DeInit() 28
error handling 31
Init() 28
interface 28
Logon() 409
overview 27
RTAuthenticationExitTypeServer 30

B

basic API calls
definition 93
error handling 11
return codes 93
bool definition 159

C

calling convention 164
category
activity instance 256
item 297
person, administrator 303
person, authorized for 302
process instance 308
process template 316
check in 629
check out 632
client concentrator 186
code page 164
comparison 97
compatibility
C++ language 827, 832
C-language 827, 831
FlowMark 827
REXX language 827
Visual Basic language 827, 832
compile
bool, string, vector 159
calling convention 164
compilers supported 165
FMC_APIENTRY 164
headers 163
library files 163
platforms supported 165
complete
data view 100
function 100, 107
concepts
memory management 10, 167
object access 9
object management 205
result object 10
session 9
constructor
activity instance 255
copy 99

constructor *(continued)*
declaration 94
container
accessor API calls 266
activity implementation 151, 161, 181, 191
activity implementation API calls 269
analyze structure 54
array 45
array index 46
array methods 270
basic API calls 266
basic data types 45
binary 267
CCSID 269
container element 46
conversion 98
data member 45
data structure 45
definition 45
double 268
element 268
element overview 270
element vector 274
example 46
exception 77
fixed data members 48
float 267
fully qualified name 45
input, activity instance 347
input, process template 595
input, work item 646
input container 369, 373
leaf 46, 54
leaf, number 267, 268
leaves 269
long 267
member 269
member, number 269
name in dot notation 45
output, activity instance 352
output, work item 648
output container 371, 376, 378, 380
overview 266
predefined data members 47
process instance, output container 543
read-only 369
read/write 369
return codes 77
stream, inbound 268
stream, outbound 267
string 267
container *(continued)*
structural member 46, 55
support tool 151, 161, 181, 191
type 56, 269
value 46, 64, 72
value, set 269
container element
access 63
array 58, 61, 271
array methods 274
assignment 271
binary 271
cardinality 272
comparison 271
constructor 271
copy 271
deallocation 271
definition 46
destructor 271
double 272
duplication 271
element, nested 272
empty 271
exception 77
float 271
leaf 46, 58, 59
leaf, number 273
leaves 273
long 271
member 273
member, number 273
name 57, 272, 273
return codes 77
string 272
structural member 58, 60
type 46, 57, 273
value 69, 274
control connector instance
accessor API calls 275
array methods 274
assignment 275
basic API calls 275
bend point 275
comparison 275
constructor 275
copy 275
deallocation 275
destructor 275
duplication 275
empty 275
enumeration, state 119
enumeration, type 120
kind 275
monitor 80
name 276
control connector instance *(continued)*
overview 275
source 276
state 276
target 276
transition condition 276
vector 276
conversion
container 98
read-only container 324, 325
read/write container 325, 326
copy
constructor 99
container 98
conversion 98
function 99

D

data access
models 19
pull 19
push 19
view 100, 107
date/time
accessor API calls 277
assignment 277
basic API call 277
comparison 277
constructor 277
current 277
day 277
destructor 277
empty 277
hour 277
minute 277
month 277
overview 276
second 277
string format 277
string representation 277
valid range 278
year 277
deallocation
declaration 99
function 35, 99
vector 35
debug
activity implementation 16
authorization 16
client application 15
dynamic link library 16
enablement 16
executable 16
prerequisites 15

debug *(continued)*
support tool 16
test database 15
description
activity instance 256
activity instance notification 297
error 280
item 297, 500
persistent list 514
person 302
process instance 308, 552
process instance list 300, 384
process instance notification 297
process template 317
process template list 300, 392
program data 321
program template 323
work item 297
worklist 300, 399
destructor
declaration 99
development kit
requirement 7
DLL options
accessor API calls 278
assignment 278
basic API calls 278
constructor 278
copy 278
creation 278
deallocation 278
destructor 278
duplication 278
empty 278
entry point 278
execution, fenced 278
file name 278
keep loaded 278
overview 278
path name 278
documentation
activity instance 256
activity instance notification 297
item 297
process instance 308
process instance notification 297
process template 317
work item 297

E

empty
function 101, 106
object 101
end time
activity instance 256
end time *(continued)*
item 297
process instance 308
work item 297
enumeration
assign reason 112
audit setting 113
escalation, activity instance 114
escalation, process instance 132
EXE options style 123
execution mode 122
execution user 123
kind, execution data 121
program retrieval, work item 135
state, activity instance 115
state, connector 119
state, item 128
state, process instance 133
time period, external service 124
type, activity instance 118
type, connector 120
type, implementation data 127
type, item 131
type, persistent list 132
equal
comparison 97
function 97
error
accessor API calls 291
ActiveX exceptions 14
basic API calls 291
handling 10
Java exceptions 11
overview 291
reason 291
reason, activity instance 257
reason, activity instance notification 261
reason, work item 333
result object 169
return codes 11
XML 227
exception, Java 291
exceptions
ActiveX GUI controls 14
Java 11
EXE options
accessor API calls 287
basic API calls 287
overview 287
style 123
execution data 21
accessor API calls 280
activity instance notification 280
execution data 21 *(continued)*
assignment 280
basic API calls 280
constructor 280
container 281
copy 280
creation 280
deallocation 280
destructor 280
duplication 280
empty 280
error description 280
kind 121, 280
object ID 281
overview 280
process instance 281
process instance notification 281
user context 281
work item 281
execution mode 122
execution service
accessor API calls 282, 328, 329
action API calls 284
activity implementation API calls 286
activity instance creation 282
activity instance notification creation 282
allocation 281
allocation, for system 281
allocation, for system group 281
array methods 286
assignment 282
basic API calls 281
compariosn 282
constructor 281
container creation, read-only 284
container creation, read/write 284
copy 281
creation 281
deallocation 282
definition 383
destructor 282
duplication 281
instance monitor creation 282
log off 407
log on 409
logged-on user 329
logon status 329
overview 247, 281
passthrough 416
password, set 615
PEA startup 420

execution service *(continued)*
person creation 282
process instance creation 283
process instance list 384
process instance list creation 283
process instance notification creation 283
process template creation 283
process template list 391
process template list creation 283
program data creation 283
program execution management API calls 286
program template creation 284
query, activity instance notification 423
query, item 431
query, process instance 446
query, process instance list 437
query, process instance notification 440
query, process template 454
query, process template list 451
query, work item 459
query, worklist 466
remote passthrough 472
session, begin 409
session, end 407
session, passthrough 416
session, remote passthrough 472
session creation 284
session ID 284
settings, logged on user 617
system 329
system group 329
timeout 329
timeout, set 329
user ID 329
work item creation 283
worklist 398
worklist creation 283
execution user 123
exit condition
activity instance 257
activity instance notification 261
work item 333
exit mode
activity instance 257
activity instance notification 262
work item 333
expiration
activity instance 257
activity instance notification 261
suspension, process instance 310
expiration *(continued)*
work item 333
External service options
accessor API calls 289
basic API calls 288
overview 288
time period 124

F

filter
activity instance notification 423
definition 33
item 431
persistent list 509, 517
process instance 446
process instance list 384, 385
process instance notification 440
process template 454
process template list 392
work item 459
worklist 398, 399
finish
activity instance, force 341
work item 638
work item, force 640
FlowMark Version 2 827
FMC_APIENTRY 164
FmcjActivityInstance
API calls 255
ForceFinish() 341
ForceRestart() 344
InContainer() 347
ObtainProcessMonitor() 349
OutContainer() 352
Refresh() 354
SubProcessInstance() 356
Terminate() 359
FmcjActivityInstanceNotification
ActivityInstance() 363
API calls 260
Delete() 489
ObtainProcessMonitor() 492
ProcessInstance() 495
Refresh() 498
SetDescription() 500
SetName() 503
StartTool() 366
Transfer() 505
Update() 145
FmcjContainer
API calls 266
container element 369
definition 369
InContainer() 369
leaves 369
FmcjContainer *(continued)*
OutContainer() 371
RemoteInContainer() 373
RemoteOutContainer() 376
SetOutContainer() 378
SetRemoteOutContainer() 380
FmcjContainerElement
accessor API calls 271
API calls 270
basic API calls 270
FmcjControlConnectorInstance
API calls 275
FmcjDateAndTime
API calls 276
FmcjDllOptions
API calls 278
FmcjError
API calls 291
FmcjExecutionData
API calls 280
FmcjExecutionService
API calls 281
CreateProcessInstanceList() 384
CreateProcessTemplateList() 391
CreateWorklist() 398
definition 383
Logoff() 407
Logon() 409
Passthrough() 416
PEAStartUp() 420
Query ActivityInstanceNotifications() 423
QueryItems() 431
QueryProcessInstanceLists() 437
QueryProcessInstanceNotifications() 440
QueryProcessInstances() 446
QueryProcessTemplateLists() 451
QueryProcessTemplates() 454
QueryWorkitems() 459
QueryWorklists() 466
Receive() 469
Refresh() 613
RemotePassthrough() 472
SetPassword() 615
SetPersonAbsent() 475
TerminateReceive() 477
UserSettings() 617
FmcjExeOptions
API calls 287
FmcjExternalOptions
API calls 288
FmcjGlobal
API calls 292
FmcjImplementationData
API calls 293

FmcjInstanceMonitor
API calls 294
ObtainBlockMonitor() 481
ObtainInstanceMonitor() 481
ObtainProcessMonitor() 481
Refresh() 484
FmcjItem
API calls 296
Delete() 489
ObtainProcessMonitor() 492
ProcessInstance() 495
Refresh() 498
SetDesription() 500
SetName() 503
Update() 145
FmcjMessage
API calls 299
FmcjPEA
API calls 279
FmcjPersistentList
API calls 300
Delete() 509
Refresh() 512
SetDescription() 514
SetFilter() 517
SetSortCriteria() 520
SetThreshold() 522
FmcjPerson
API calls 301
Refresh() 527
SetAbsence() 529
SetSubstitute() 531
FmcjPoint
API calls 306
FmcjProcessInstance
API calls 307
Delete() 535
InContainer() 538
ObtainProcessMonitor() 540
OutContainer() 543
Refresh() 545
Restart() 548
Resume() 550
SetDescription() 552
SetName() 555
Start() 557
Suspend() 560
Terminate() 563
Transfer() 505
FmcjProcessInstanceList
API calls 312
Delete() 509
QueryProcessInstances() 567
Refresh() 512
SetDescription() 514
FmcjProcessInstanceList *(continued)*
SetFilter() 517
SetSortCriteria() 520
SetThreshold() 522
FmcjProcessInstanceNotification
API calls 313
Delete() 489
ObtainProcessMonitor() 492
ProcessInstance() 495
Refresh() 498
SetDescription() 500
SetName() 503
Transfer() 505
Update() 145
FmcjProcessTemplate
API calls 315
CreateAndStartInstance() 571
CreateInstance() 579
Delete() 582
ExecuteProcessInstance() 585
InitialInContainer() 595
ProgramTemplate() 597
Refresh() 600
FmcjProcessTemplateList
API calls 318
Delete() 509
QueryProcessTemplates() 603
Refresh() 512
SetDescription() 514
SetFilter() 517
SetSortCriteria() 520
SetThreshold() 522
FmcjProgramData
API calls 320
FmcjProgramTemplate
API calls 322
Execute() 607
FmcjReadOnlyContainer
API calls 324
FmcjReadOnlyContainerHolder
API calls 325
FmcjReadWriteContainer
API calls 325
FmcjResult
API calls 327
FmcjService
API calls 328
definition 613
PEAShutDown() 418
Refresh() 613
SetPassword() 615
UserSettings() 617
FmcjStringVector
vector 330
FmcjSymbolLayout
API calls 331
FmcjWorkitem
ActivityInstance() 624
API calls 332
CancelCheckOut() 627
CheckIn() 629
CheckOut() 632
Delete() 489
Finish() 638
ForceFinish() 640
ForceRestart() 643
InContainer() 646
ObtainProcessMonitor() 492
OutContainer() 648
ProcessInstance() 495
Refresh() 498
Restart() 650
SetDescription() 500
SetName() 503
Start() 652, 655
Terminate() 657
Transfer() 505
Update() 145
FmcjWorklist
API calls 335
Delete() 509
QueryActivityInstance Notifications() 661
QueryItems() 665
QueryProcessInstance Notifications() 668
QueryWorkitems() 671
Refresh() 512
SetDescription() 514
SetFilter() 517
SetSortCriteria() 520
SetThreshold() 522
fully qualified name 45
function
accessor 106
action 150
activity implementation 151
basic 93
categories 93
client/server call 150
program execution management 155
vector accessor 35

G

global services
accessor API calls 292
basic API calls 292
overview 292

H

handle
  object 9
High Performance Bridge 192

I

icon
  activity instance 257
  activity instance notification 297
  graphical information 331
  item 297
  process instance 308
  process instance notification 297
  process template 317
  program data 321
  program template 323
  work item 297
implementation data
  accessor API calls 293
  activity instance 257
  activity instance notification 262
  basic API calls 293
  basis, implementation data 125
  DLL 278
  EXE 287
  external service 288
  overview 293
  platform 125
  program 320
  program template 322
  type 127
  work item 333
input container
  activity implementation 161, 181, 191
  activity instance 257, 347
  activity instance notification 297
  item 297
  needed 309, 317
  process instance 308, 538
  process instance notification 297
  process template 317, 595
  program data 321
  program template 323
  support tool 161, 181, 191
  work item 297, 646
instance monitor
  accessor API calls 295
  action API calls 295
  activity instance 80, 295
  assignment 295
  basic API calls 295
  comparison 295
  constructor 295
instance monitor *(continued)*
  control connector instance 80, 295
  copy 295
  creation 295
  deallocation 295
  definition 481
  destructor 295
  duplication 295
  monitor, block activity 481
  monitor, process activity 481
  object ID 295
  obtain 79, 81
  overview 79, 294
  ownership 81
  process instance 349, 492, 540
  refresh 484
item
  accessor API calls 296
  action API calls 299
  assignment 296
  basic API calls 296
  category 297
  comparison 296
  completeness 296
  constructor 296
  copy 296
  creation 296
  creation time 297
  deallocation 296
  definition 489
  delete 489
  description 297
  description, set 500
  destructor 296
  documentation 297
  duplication 296
  empty 296
  end time 297
  filter 431, 459
  icon 297
  input container, name 297
  kind 296
  last modification time 297
  monitor, process instance 492
  mutator API calls 298
  name 298, 503
  object ID 298
  object identifier 489
  output container, name 298
  overview 296
  owner 298
  process administrator 298
  process instance, name 298
  process instance, retrieval 495
item *(continued)*
  process instance, state 298
  process instance ID 298
  properties 500
  query 431
  received reason 298
  received time 298
  refresh 498
  sort criteria 434, 463
  start time 298
  state 128, 621
  system 298
  system group 298
  threshold 434, 463
  transfer 505
  type 131
  update 298
  vector 299
  worklist 398

J

Java High Performance Bridge 192

K

kind
  function 101

L

last modification time
  activity instance 257
  activity instance notification 297
  item 297
  process instance 309
  process instance notification 297
  process template 317
  work item 297
last state change time
  activity instance 257
  process instance 309
locale 164
log off 407
logon
  absence setting 411
  default 410
  present 410
  session, execution server 409
  session mode 410

M

memory
  management 10, 167
  ownership 10
  thread 168, 205
message
  accessor API calls 299
  overview 299

message *(continued)*
text 300
message interface 207
method
accessor 106
action 150
activity implementation 151
basic 93
categories 93
client/server call 150
program execution management 155
migration
C++ programs 832
C-language programs 831
compatibility mode 831, 832
steps 832
Visual Basic programs 832
modules 3
MQSeries message descriptor 209

N

name
activity instance 257, 258
activity instance notification 298
agent 265, 266
agent, set 266
container 269
container element 272, 273
control connector instance 276
data structure 269, 273
DLL 278
entry point 278
EXE 288
external service 290
implementation 257, 262, 333
input container 297, 308
input container, activity instance 257
item 298, 503
name 607
organization 309, 317
output container 298, 309, 317
output container, activity instance 258
parent, process instance 310
persistent list 509
person, first name 302
person, last name 304
person, middle name 304
person, organization 304
person, organizations 304
person, roles 304
process instance 258, 298, 309, 311, 555, 571, 579, 586
name *(continued)*
process instance list 300, 384, 567
process instance notification 298
process template 310, 317
process template list 300, 391, 392, 603
program 257, 262, 333
role 310, 318
symbol position 331
syntax 503, 555, 571, 579, 586
system 258, 298, 305, 311, 329
system group 258, 298, 329
top level process instance 311
work item 298
working directory 288
worklist 300, 398, 399, 661
notification
activity instance 257, 258
activity instance notification 261, 262
activity instance notification, query 423, 661
filter 423, 440
item, query 431
persons, activity instance 257
process instance 309
process instance notification 314
process instance notification, query 440, 668
sort criteria 427, 442, 443
threshold 427, 443
work item 333
worklist, create 398

O

object
access 9
management 205
memory management 10
optional property 106
persistent 167, 205
primary property 107
secondary property 107
transient 167, 205
object identifier
activity instance 258, 262, 279, 333
activity instance notification 298, 363
control connector source 276
control connector target 276
execution data 281
instance monitor 295
item 489
object identifier *(continued)*
process instance 298, 310, 535
process instance list 300
process instance notification 298
process template 317, 571
process template list 300, 310
work item 298, 621
worklist 300
output container
activity implementation 161, 181, 191
activity instance 352
process instance 543
work item 648
owner
instance monitor 81
persistent list 509
process instance list 384, 567
process template list 391, 603
transfer, item 505
worklist 398, 661

P

passthrough 416, 472
password, set 615
persistent list
accessor API calls 300
action API calls 301
basic API calls 300
definition 33, 509
delete 509
description 300, 384, 392, 399
description, set 514
empty 300
filter 300, 384, 385, 392, 398, 399, 509
filter, set 517
name 300, 384, 391, 392, 398, 399, 509
object ID 300
overview 249, 300
owner 300, 384, 391, 398, 509
private 301
process instance 384
process template list 391
public 301
query 567, 603
query, process instance list 437
query, worklist 661, 665, 668, 671
refresh 512
sort criteria 300, 384, 387, 392, 394, 399, 403, 509
sort criteria, set 520
threshold 300, 384, 392, 398, 509

persistent list *(continued)*
threshold, set 522
type 132, 301, 384, 391, 398, 509
worklist 398

person
absence 302, 475, 529
absence, resetting 304
accessor API calls 301
action API calls 305
administrator 303
assignment 301
authorization 302, 303, 304, 305
administrator 303
categories 302, 303
category administrator 303
definition 303
for me 305
operation 303
persons 303, 305
process definition 303
staff definition 303
topology definition 303
basic API calls 301
categories 302
category administor 303
comparison 301
completeness 301
constructor 301
copy 301
creation 301
deallocation 301
definition 527
description 302
destructor 301
duplication 301
empty 301
first name 302
last name 304
level 304
manager 303, 304
middle name 304
notification 257, 258
oranization 304
organization 304
overview 301
password, set 615
person ID 304
phone 305
refresh 527
role 304
role coordination 304
settings, logged on user 617
substitute 305, 531
substitutee 305
system 305

person *(continued)*
user ID 305
point
accessor API calls 306
array methods 306
assignment 306
basic API calls 306
comparison 306
constructor 306
copy 306
creation 306
deallocation 306
destructor 306
duplication 306
empty 306
overview 306
vector 307
x-coordinate 306
y-coordinate 306
predefined data members 47
_ACTIVITY 48
_ACTIVITY_INFO 50
CoordinatorOfRole 51
definition 50
Duration 53
Duration2 53
LowerLevel 52
MembersOfRoles 50
Organization 51
OrganizationType 52
People 52
PersonToNotify 53
Priority 50
UpperLevel 52
_PROCESS 48
_PROCESS_INFO 49
definition 49
Duration 50
Organization 49
ProcessAdministrator 49
Role 49
_PROCESS_MODEL 48
_RC 48
activity information 47, 50
fixed 47, 48
process information 47, 49
primary view
definition 107
IsComplete() 100
priority
activity instance 258
activity instance notification 262
execute program 324
work item 333
process administrator 83

process administrator 83 *(continued)*
activity instance 258
activity instance notification 298
process instance 310
process instance notification 298
process template 318
work item 298

process instance
accessor API calls 307
action API calls 311
assignment 307
audit mode 308
basic API calls 307
category 308
comparison 307
completeness 307
constructor 307
copy 307
create 571, 579
creation 307
creation time 308
creator 308
deallocation 307
definition 535
delete 535
description 308, 552
destructor 307
documentation 308
duplication 307
empty 307
end time 308
escalation 132
execute 585
filter 446
icon 308
input container 538
input container, name 308
input container, needed 309
last modification time 309
last state change time 309
monitor 540
name 309, 535, 555, 571, 579, 586
notification 440
notification state 310
notification time 309
notified person 309
object ID 310
object identifier 535
organization 309
output container 543
output container, name 309
overview 307
parent name 310

process instance *(continued)*
persistent list, create 384
process administrator 310
process template ID 310
process template name 310
query 446
refresh 545
restart 548
resume 550
role 310
sort criteria 448
start 557, 571
start time 310
starter 310
state 133, 310, 535
suspend 560
suspension expiration time 310
suspension time 311
system 311
terminate 563
threshold 448
top level name 311
vector 315
process instance list
accessor API calls 300
action API calls 301, 312
array methods 312
assignment 312
basic API calls 312
comparison 312
constructor 312
copy 312
creation 312, 384
deallocation 312
delete 509
description 300, 384
description, set 514
destructor 312
duplication 312
filter 300, 384, 385
filter, set 517
name 300, 384, 567
object ID 300
overview 312
owner 300, 384, 567
private 301
public 301
query 437, 567
refresh 512
sort criteria 300, 384, 387
sort criteria, set 520
threshold 300, 384
threshold, set 522
type 132, 301, 384, 567
vector 313
process instance notification
accessor API calls 296, 314
action API calls 314
array methods 314
assignment 314
basic API calls 313
comparison 314
constructor 313
copy 313
creation 313
deallocation 313
delete 489
description, set 500
destructor 313
duplication 313
kind 313
monitor,process instance 492
name, set 503
notification time 314
overview 313
process instance 495
refresh 498
transfer 505
update 145
vector 315
process template
accessor API calls 316
action API calls 318
assignment 316
audit mode 316
basic API calls 316
category 316
comparison 316
completeness 316
constructor 316
copy 316
create process instance 571, 579
creation 316
creation time 317
deallocation 316
definition 571
delete 582
description 317
destructor 316
documentation 317
duplication 316
empty 316
excution user 123
execute process instance 585
execution mode 122
filter 454
icon 317
input container 595
input container, name 317
input container, needed 317
process template *(continued)*
last modification time 317
name 317, 571
object ID 317
object identifier 571
organization 317
output container, name 317
overview 315
persistent list, create 391
process administrator 318
program template 597, 607
query 454
refresh 600
role 318
sort criteria 456
start process instance 571
threshold 456
valid-from date 571
valid from time 318
vector 320
process template list
accessor API calls 300
action API calls 301, 319
array methods 319
assignment 319
basic API calls 319
comparison 319
constructor 319
copy 319
creation 319, 391
deallocation 319
delete 509
description 300, 392
description, set 514
destructor 319
duplication 319
filter 300, 392
filter, set 517
name 300, 391, 392, 603
object ID 300
overview 318
owner 300, 391, 603
private 301
public 301
query 451, 603
refresh 512
sort criteria 300, 392, 394
sort criteria, set 520
threshold 300, 392
threshold, set 522
type 132, 301, 391, 603
vector 319
profile
defaults 383
user 383

profile *(continued)*
workstation 383
program data
accessor API calls 321
assignment 321
basic API calls 320
comparison 321
constructor 320
copy 320
creation 320
deallocation 320
description 321
destructor 320
duplication 320
empty 321
execution mode 321
execution user 321
icon 321
implementation 321
input container 321
output container 321
overview 320
stream format 321
trusted 322
unattended mode 321
program execution agent
activity implementation API calls 279
activity instance, object ID 279
overview 279
program ID 279
shutdown 418
start 420
user ID 279
program execution management API calls
API calls 155
error handling 10
program execution agent 151
program execution server
definition 220
program identification 152
program template
assignment 322
comparison 322
constructor 322
copy 322
creation 322
deallocation 322
definition 607
description 323
destructor 322
duplication 322
empty 322
execute 607
program template *(continued)*
execution mode 323
execution user 323
icon 323
implementation 323
input container 323
input container, needed 323
output container 323
output container, needed 323
overview 322
stream format 322
structures from activity 323
trusted 323
unattended mode 323
valid from time 323
programming
activity implementation 9, 247
client 9, 247
prerequisites 7
support tool 9, 247
property
optional 106
primary 107
secondary 107
protocol
asynchronous 19
supported 19
synchronous 19
unsolicited 19, 410
pull data 19
push
data, receive 469
enable 20
kind of information 20
receive 21
session mode 410
terminate receive 477
push data 19

Q

query
activity instance notification 423
array of objects 34
data 33
item 431
process instance 446
process instance list 437
process instance list, process instances 567
process instance notification 440
process template list 451
process template list, process templates 603
vector of objects 34
work item 459
worklist 466, 661
query *(continued)*
worklist, items 665
worklist, process instance notification 668
worklist, work item 671

R

read-only container
accessor API calls 266
activity implementation, input container 369, 373
activity instance, input container 347
activity instance, output container 352
assignment 324
basic API calls 324
comparison 324
constructor 324, 325
conversion 324, 325
copy 324
creation 324, 325
deallocation 324
definition 369
destructor 324
duplication 324
holder 325
overview 324
process instance, output container 543
work item, input container 646
read-only container holder
accessor API calls 325
basic API calls 325
overview 325
read/write container
accessor API calls 266, 326
activity implementation, output container 371, 376, 378, 380
assignment 326
basic API calls 325
binary, set 326
comparison 326
constructor 326
conversion 325, 326
copy 326
creation 326
deallocation 326
definition 369
destructor 326
float, set 326, 327
long, set 326, 327
mutator API calls 326
overview 325

read/write container *(continued)*
process instance, input container 538
process template, input container 595
string, set 326, 327
value, set 327
work item, output container 648
receive data 469
remote
terminate, subprocess 563
restart
activity instance, force 344
work item 650
work item, force 643
result object
access 328
accessor API calls 328
basic API calls 327
definition 169
destructor 327
error information 10
information contained 169
message text 328
origin 328
overview 327
parameter 328
retrieval 328
return code 328
thread 169
return code
access API calls 107
action API calls 10
activity implementation 162, 181, 191
basic API calls 93
error handling 10
list of 11

S

secondary view
definition 107
IsComplete() 100
service
accessor API calls 328
action API calls 329
execution service 383
logged-on user 329
logon status 329
overview 328
password, set 615
settings, logged on user 617
system 329
system group 329
timeout 329
service *(continued)*
timeout, set 329
user ID 329
session
absence setting 411
begin 383, 409, 416, 472
default 410
end 383, 407
establish 25
establish, execution server 383
log off 383, 407
log on 383, 409
mode 410
overview 25
passthrough 416
present 410
remote passthrough 472
requirement 9
unified logon 410
sort criteria
activity instance notification 427
definition 34
item 434, 463
persistent list 509, 520
process instance 448
process instance list 384, 387
process instance notification 442, 443
process template 456
process template list 392, 394
work item 463
worklist 399, 403
staff
activity instance 258
activity instance notification 262
authorization 303
work item 333
start
process instance 557, 571
support tool 366
work item 652, 655
start condition
activity instance 259
activity instance notification 262
work item 334
start mode
activity instance 258
activity instance notification 262
work item 333
start time
activity instance 259
process instance 310
work item 298
starter
activity instance 259
starter *(continued)*
process instance 310
state
activity instance 259
activity instance notification 262
control connector instance 276
item 621
last change time 309
process instance 258, 298, 310, 535
work item 334, 621
stateless
application 87
AsStream 88
FromStream 88
identity-based objects 88
Persistent*Object* 88
PersistentOID 88
server 87
SessionID 88
value-based objects 88
string
array methods 329
vector 330
string definition 159
subprocess
resume 550
suspend 560
terminate 563
support tool
activity instance 259
activity instance notification 262
input container 151, 161, 181, 191
program identification 152
pseudo code 161, 181, 191
work item 334
suspension
process instance 560
symbol layout
accessor API calls 331
activity instance 259
assignement 331
basic API calls 331
comparison 331
constructor 331
copy 331
creation 331
deallocation 331
destructor 331
duplication 331
empty 331
icon position 331, 332
name position 331, 332
overview 331

synchronous protocol 19
syntax diagrams
how to read xiii
syntax rules
description, item 500
description, persistent list 514
description, process instance 552
name, item 503
name, process instance 555, 571, 579, 586
XML DTD 237
system
execution server 383
execution service 329
person 305
process instance 258, 298, 311
system administrator 83
system group
execution server 383
execution service 329
name 298
process instance 258, 311

T

thread 168, 205
considerations 196
Java CORBA Agent 185
number of 186
safeness 168
thread pool 186
worker thread 186
threshold
activity instance notifications 427
definition 34
items 434, 463
multi-thread libraries 164
persistent list 509, 522
process instance list 384
process instance notifications 443
process instances 448
process template list 392
process templates 456
worklist 398
transient object 9
type
persistent list 509
private, persistent list 509
private, process instance list 567
private, process template list 603
private, worklist 661
process instance list 384, 567
process template list 391, 603
type *(continued)*
public, persistent list 509
public, process instance list 567
public, process template list 603
public, worklist 661
worklist 398, 661

U

unified logon 410
unsolicited information 19
user
activity implementation 152
default values, profile 383
password, set 615
settings 617

V

vector
accessor function 35
activity instance notifications 263
activity instances 263
container elements 274
control connector instances 276
deallocate 35
definiiton 159
first element 36
implementation data 294
items 299
next element 36
overview 330
points 307
process instance lists 313
process instance notifications 315
process instances 315
process template lists 319
process templates 320
query result 34
return codes 35
size 37
work items 335
worklist 337
view
data view 107
IsComplete() 100
primary 107
secondary 107

W

work item
accessor API calls 296, 332
action API calls 334
activity instance, retrieval 624
activity instance ID 333
activity instance kind 333
work item *(continued)*
array methods 335
assignment 332
basic API calls 332
cancel checkout 627
category 297
check in 629
check out 632
comparison 332
constructor 332
copy 332
creation 332
creation time 297
deallocation 332
definition 621
delete 489
description 297
description, set 500
destructor 332
documentation 297
duplication 332
end time 297
error reason 291, 333
exit condition 333
exit mode 333
expiration time 333
finish 638
finish, force 640
first notification 333
icon 297
implementation 333
input container 646
input container, name 297
kind 332
last modification time 297
monitor, process instance 492
name 298
name, set 503
notification state 334
notification time 333
object ID 298
object identifier 621
output container 648
output container, name 298
overview 332
owner 298
persistent list, create 398
priority 333
process administrator 298
process instance 495
process instance, name 298
process instance, state 298
process instance ID 298
program retrieval 135
query 431, 459

work item *(continued)*
query, worklist 671
received reason 298
received time 298
refresh 498
restart 650
restart, force 643
second notification 333
staff 333
start 652, 655
start condition 334
start mode 333
start time 298
state 334, 621
support tool 334
system 298
system group 298
terminate 657
transfer 505
update 145, 298
vector 335
workflow model 3
worklist
accessor API calls 300, 336
action API calls 301, 336
array methods 336
assignment 335
basic API calls 335
beep option 336
comparison 335
constructor 335
copy 335
creation 335, 398
deallocation 335
definition 661
delete 509
description 300, 399
description, set 514
destructor 335
duplication 335
filter 300, 398, 399
filter, set 517
name 300, 398, 399, 661
object ID 300
overview 335
owner 300, 398, 661
private 301
public 301
query 466, 665, 668, 671
query, activity instance notification 661
refresh 512
sort criteria 300, 399, 403
sort criteria, set 520
threshold 300, 398
worklist *(continued)*
threshold, set 522
type 132, 301, 398, 661
vector 337
workstation profile
default values 383

X

XML
activity implementation 219
authentication 216
authorization 216
code page support 214
container 212
correlation 211
DTD 237
error handling 227
example, container 211
example, create/start process instance 216
example, execute process instance 213
message content 210
message format 209, 237
message interface 207
user context data 211
user-defined program execution server 220
XML message header 211

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

IBM MQSeries Workflow
Programming Guide
Version 3.3

Publication No. SH12-6291-06

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

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. _____

Readers' Comments — We'd Like to Hear from You
SH12-6291-06

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany

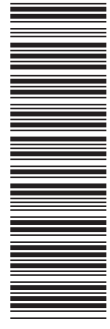
Fold and Tape **Please do not staple** Fold and Tape

SH12-6291-06

Cut or Fold
Along Line

Part Number: CT89AIE
Program Number: 5697-FM3

Printed in Denmark by IBM Danmark A/S



SH12-6291-06

Spine information:

IBM MQSeries Workflow Programming Guide Version 3.3