IBM MQSeries Workflow

Programming Guide

Version 3.2

SH12-6291-03

IBM MQSeries Workflow

Programming Guide

Version 3.2

SH12-6291-03

Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 795.

Fourth Edition (June 1999)

This edition applies to version 3, release 2 of IBM MQSeries Workflow (product number 5697-FM3) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SH12-6291-02.

Contents

About this book

This book describes how to use the IBM MQSeries Workflow Client Application Programming Interfaces, hereafter called MQ Workflow APIs. The first part of the book describes the concepts underlying the APIs while the remainder of the book provides you with an API reference manual. 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. Programmers must have experience with the respective operating system they are using.

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 805.

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 functions/methods according to their behavior and describes basic and accessor methods in a generic way.
- "Part 2. The C and C++ APIs" on page 131 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 147 provides for an overview on the ActiveX Controls.
- "Part 4. The JAVA API" on page 157 provides for an overview on the Java API.

"Part 5. Using the MQ Workflow APIs" on page 177 provides for an overview on the functionality supported by the MQ Workflow Runtime. All functions/methods 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 6. Programming interfaces" on page 259 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 functions/methods are described on a per-object basis. See "Chapter 11. Function/method types" on page 79 for the description of the basic and accessor methods.
- "Part 7. Working with ActiveX Controls" on page 559 describes the methods and events supported by the ActiveX Controls.

"Part 8. Examples and scenarios" on page 637 provides some examples showing how to use the APIs.

"Part 9. Using the Lotus Notes API" on page 681 describes the API provided for the integration of MQ Workflow and Lotus Notes.

"Appendix A. How to read the syntax diagrams" on page 781 states the parts of a syntax diagram.

"Appendix B. FlowMark Version 2 compatibility mode" on page 783 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.

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

Summary of Changes

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 function/method Refresh().
- The process template exposes a new action function/method 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 function/method Restart().
- The work item and the activity instance notification expose new action functions/methods StartTool(). The work item exposes a new function/method CancelCheckOut().
- Restrictions are removed from the work item Restart(), ForceFinish(), and ForceRestart() functions/methods; 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 function/method 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 functions/methods Refresh(), SetAbsence(), and SetSubstitute().
- The process template exposes a new action function/method Delete().
- The IsTerminatedOnError() accessor function/method 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 function/method 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 785 for more information.

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.

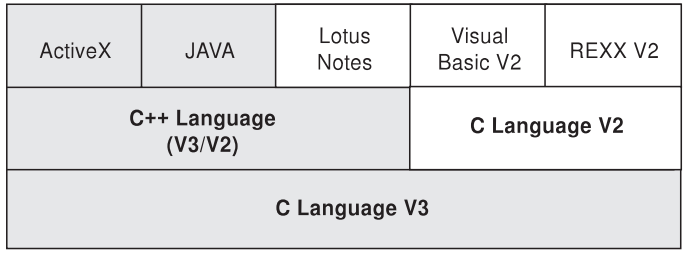
Return codes, provided by the assigned program, can then be used to evaluate exit and transition conditions.

Chapter 2. Programming interfaces

The MQ Workflow workflow manager provides application program interface (API) support and 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.

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.

The following MQ Workflow programming aids 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 Lotus Notes API



The basic interface for requesting Runtime services from MQ Workflow is a C-language API. Access can be gained to its functions from all languages that support C calls - see “Chapter 14. Compiling and linking” on page 137 for more information. On top of the C-language API, a C++ language API is provided. 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. The same is true for the Lotus Notes API which, however, operates in the FlowMark Version 2 compatibility mode. Besides the Version 3 APIs, FlowMark Version 2 C-language, C++, VisualBasic, and REXX APIs are supported.

The MQ Workflow APIs provide functions/methods:

- 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; the Lotus Notes API uses the check-out/check-in mechanism

Chapter 3. Prerequisites for programming

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

- The MQ Workflow Development Kit is 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 131, “Part 3. ActiveX Controls” on page 147, and “Part 4. The JAVA API” on page 157 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.

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 functions in the C-language API respectively methods in the C++, ActiveX, or Java language APIs, hereafter called functions/methods, are available for each action on an object or for accessing each property of an

object. This approach allows function/method 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 functions/methods. See chapter "Chapter 16. The result object" on page 143 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.

Handling errors

All action, activity implementation, or program execution management functions/methods 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 return code is one of a set of predefined codes (see "List of return codes"). The exact return codes or exceptions for each of those functions/methods are listed with the description of each call. You should design your programs to handle all return codes or exceptions that can arise.

In addition to the return code, a so-called *result object* can be accessed in C and C++ which describes the result of the call in more detail - see "Chapter 16. The result object" on page 143.

Basic and accessor functions/methods do not return any value or return the value queried as their return value. Since they are querying transient objects and are able to return default values, an error does normally not occur. It can, however, happen during application development that a wrong handle or a buffer too small to hold a character value is specified. To look for such erroneous situations, the *result object* can be queried (besides checking the trace).

List of return codes

The following list shows the numeric values of the return codes or exceptions that are issued by the MQ Workflow APIs; it is strongly adviced to use the symbolic names instead of the integer values:

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
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
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

Table 1. List of return codes (continued)

Numeric value	Symbolic value
136	FMC_ERROR_SIZE_EXCEEDED
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
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
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

Table 1. List of return codes (continued)

Numeric value	Symbolic value
1023	FMC_ERROR_MIGRATION_OF_RUNNING_PROGRAM
1024	FMC_ERROR_MIGRATION_OF_CHECKEDOUT_SUSPENDED
1025	FMC_ERROR_MIGRATION_NO_SUBPROCESS
2000	FMC_ERROR_INVALID_QUEUE_SCOPE

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 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" on page 14).
- 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; currently only the execution of a process instance is supported in asynchronous mode.

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 functions/methods operate this way; your application (thread) is blocked until the response arrives or until your timeout set on the execution service object exceeds.

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() function/method 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.

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.

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 functions/methods 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 functions/methods" on page 89 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 for receiving data in order to prevent that the whole application is blocked.

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() function/method 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 data arriving is received; note that the correlation identification is set as the result of a successful receive.

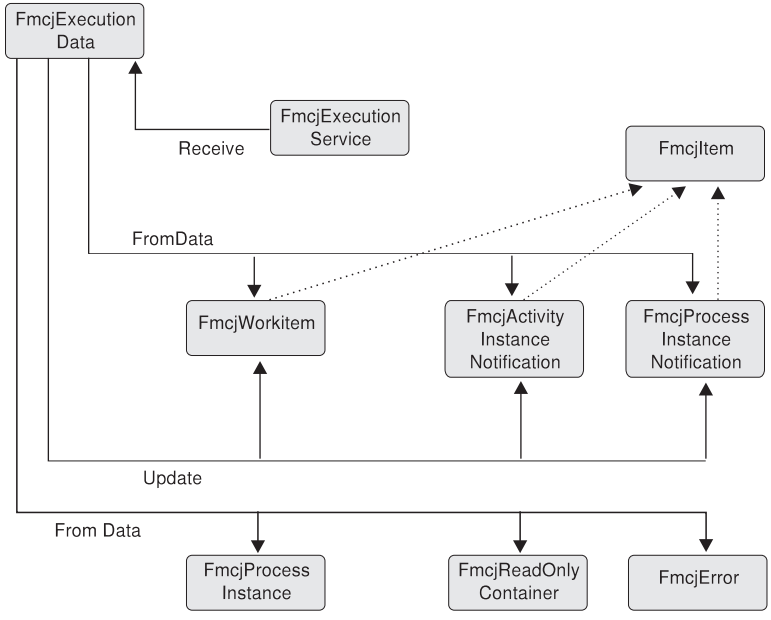


Figure 1. Handling data sent by an MQ Workflow server. Legend: --➤ Inheritance (C++); ⟶ provides for access

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 occurred, the error description can be looked at.

Note: ActiveX uses the event mechanism in order to inform an application that data has been changed. See “Events” on page 624.

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 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.

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() functions/methods 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. 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.

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 functions/methods 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 functions/methods 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.[1] If the vector handle is not initialized to 0, it **must** point to a vector of objects of the appropriate kind so that newly queried objects can be appended. In other words, any nonzero handle is used by 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

1. 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 function/method 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 * handle)
```

Parameters

handle 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.

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]="";



  FmcjGlobalConnect();

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


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


  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);
```

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

   FmcjGlobalDisconnect();
   return FMC_OK;
}
```

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]="";


  FmcjGlobalConnect();

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


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

```
  printf("Instances in the vector:\n");
  while (0 != (hdlInstance=FmcjProcessInstanceVectorNextElement(hdlVector)))
  {
    printf("- name: %s\n",
           FmcjProcessInstanceName(hdlInstance,tInfo,
                                   FMC_PROCESS_INSTANCE_NAME_LENGTH));
    FmcjProcessInstanceDeallocate(&hdlInstance) );
  }
   FmcjProcessInstanceVectorDeallocate(&hdlVector) );



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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

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()
```

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

New*Object*

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 New Object( long index )
```

Parameters

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

*Object*Remove

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 Object Remove( 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 8. 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 32KB 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, however, 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 parenthesis ().

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

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

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

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

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

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

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

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

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

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

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

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

Also note that the fully qualified names are not prefixed with the name of the data structure PERSON. That data structure denotes the type of the container. There is only one exception to the rule, when the container itself is specified to be an array, for example, an array of PERSONs. Then, to set the name of a specific person, the fully qualified name is specified as

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

For detailed examples see "Part 8. Examples and scenarios" on page 637.

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 functions/methods.

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 function/method. An exception is the _RC data member which, however, should not be set - see below.

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 functions/methods. Their values *can be read* using API container functions/methods. Fixed data member _RC *cannot be read* and should only be set when your compiler does not support a program exit code.

_ACTIVITY
: This data member contains the name of the considered activity instance. The value of this data member is automatically set when the activity instance respectively 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 respectively 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 respectively an associated work item is started.

  Data type: STRING

_RC This data member contains the return code of the activity implementation. Typically it is used to evaluate exit and transition conditions. It cannot be read and is set automatically (overwritten) to the exit code of the activity implementation when that program ends.

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 from.

- 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 regarded 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 from.

- 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).

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 9 can be specified. If the value specified is invalid or the data member is not set, a default of 0 (zero) is used.

Data type: LONG

_ACTIVITY_INFO.MembersOfRoles

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

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

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

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

Data type: STRING

_ACTIVITY_INFO.CoordinatorOfRole

The role or roles a person must coordinate to receive a work item for

the activity instance. Multiple roles to coordinate may be specified and are then to be separated by a semicolon (;).

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

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

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

Data type: STRING

`_ACTIVITY_INFO.Organization`

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

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

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

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

Data type: STRING

`_ACTIVITY_INFO.OrganizationType`

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

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

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 level persons must at least 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 level persons should not exceed to receive work items of the activity instance. A value between 0 and 9 can be set. The default value is 9.

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

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

Data type: LONG

_ACTIVITY_INFO.People

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

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

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

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

Data type: STRING

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

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

Data type: LONG

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

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

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

Data type: LONG

_ACTIVITY_INFO.Duration2
Used to specify the maximum number of seconds allowed to act on an activity instance notification.

If the notification is not acted on before the specified number of hours 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 functions/methods in order to determine the structure of an unknown container and/or its leaves. Applied on a container, they return a collection of container elements. Once the collection of container elements is available, similar functions/methods can be recursively applied in order to step down through a nested structure.

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.

Determining the leaves

Following functions/methods 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 respectively the number of all leaves, user-defined and predefined.

Determining the structural members

Following functions/methods 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

Following functions/methods 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

Following functions/methods 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.

Determining the structural properties of a container element

Following functions/methods 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

Following functions/methods allow to determine the number of leaves of a container element or to retrieve the leaves themselves.

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

Following functions/methods 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

Following functions/methods 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, then that name can be used in order to directly access the container element without iterating and searching through the whole container structure.

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

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

C-language signature

```
APIRET FMC_APIENTRY FmcjContainerGetElement(
                    FmcjContainerHandle          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 to be queried.

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

Return type

long/APIRET The return code of calling this function/method - see return codes.

Accessing a value of a container

Following functions/methods 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() functions/methods 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 function/method - see return codes.

Accessing a value of a container element

Following functions/methods 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() functions/methods 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 function/method - see return codes.

Setting a value of a container

Following functions/methods allow to set the value of a container leaf.

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

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(
                    FmcjContainerHandle     handle,
                    char const *            qualifiedName,
                    unsigned long           index,
                    FmcjBinary const *      value,
                    unsigned long           dataLength )

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

C-language signatures

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

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

C-language signatures

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

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

C-language signatures

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

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

C++ language signatures

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

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

C++ 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 const &    value        ) const

APIRET Value( 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 function/method - see return codes.

Return codes/FmcException

Following return codes can be returned or can be described by the result object respectively following exceptions can be thrown, the number in parentheses shows their integer value:

FMC_OK(0) The function/method 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 9. 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.

Obtaining a process instance monitor

Once a process instance[2] has been accessed, a **process instance monitor**[3] can be obtained. The transient process instance monitor object then represents all information about activity instances directly contained in the described process instance as well as all information on control connector instances connecting those activity instances.

2. or activity instance or a (work) item

3. ActiveX does not distinguish between process instance monitors and block instance monitors; they are both called *instance monitor*.

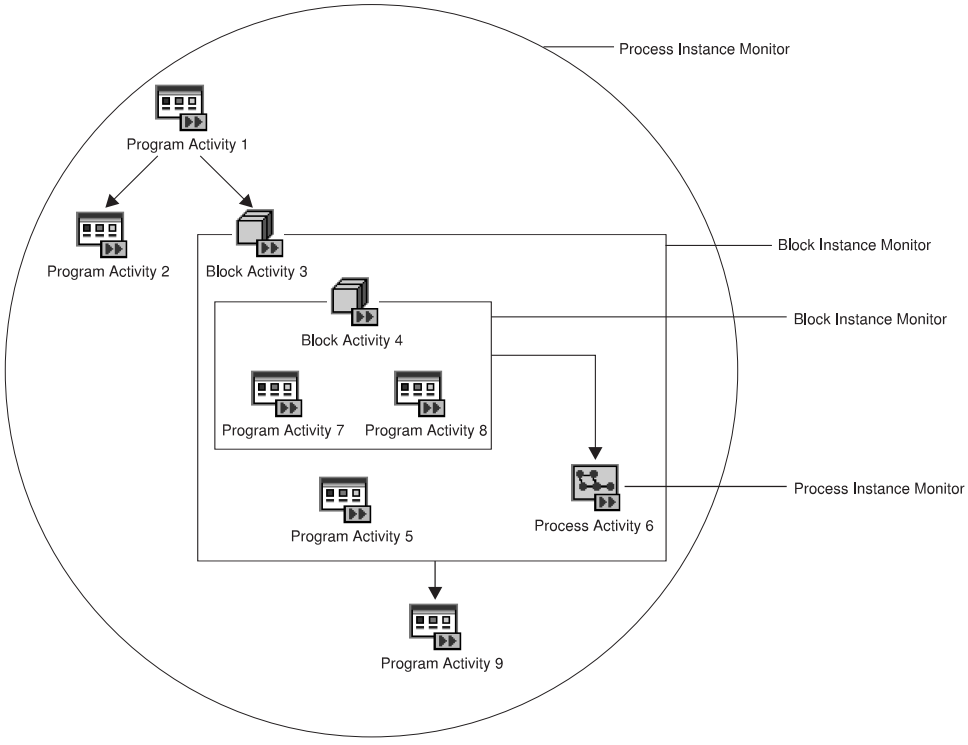


Figure 2. Process instance monitors and block instance monitors

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

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

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

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

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

Note: The deep option is currently ignored.

Ownership of monitors

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

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

Chapter 10. 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 40 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: 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 11. Function/method types

MQ Workflow functions/methods can be divided into several categories which characterize the kind and behavior of the request to be executed.

basic	to manage transient objects
accessor	to read properties of transient objects
action	to read or manipulate persistent objects
activity implementation	to deal with containers from within an activity implementation or support tool
program execution management	to handle program execution agents

Basic and accessor functions/methods are described in more detail but generally below. This is because all these functions/methods 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. The functions/methods of the other categories are described separately in "Part 6. Programming interfaces" on page 259. Those are the functions/methods which require client/server communication or communication with the program execution agent.

Basic functions/methods

Basic functions/methods 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 15. Memory management" on page 141.

Basic functions/methods 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 functions and the MQ Workflow result object can return the following codes, the number in parentheses shows their integer value:

FMC_OK(0) The function/method 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_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

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

Allocation

Following functions/methods 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

```
ProcessTemplate *               NewProcessTemplate()
ProcessInstance *               NewProcessInstance()
Workitem *                      NewWorkitem()
ActivityInstanceNotification *  NewActivityInstanceNotification()
ProcessInstanceNotification *   NewProcessInstanceNotification()
```

C-language signatures

```
APIRET FMC_APIENTRY
FmcjExecutionServiceAllocate( FmcjExecutionServiceHandle * handle )

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

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 )
```

Java signature

```
Agent()
```

Parameters

handle
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 system where the execution server runs.

systemGroup
Input. The system group where the execution server resides.

year/month/day
Input. The date part of the date/time.

hour/minute/second
Input. The time part of the date/time.

Return type
APIRET
The return code set by the allocation.
Object*
The newly constructed object.

Assignment

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

C++ language signature

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

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

Comparison/equality

Following functions/methods 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.

Copy

Following functions/methods 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 functions/methods 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 functions/methods" on page 89).

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 93.

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;
```

```
FmcjGlobalConnect();

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



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


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


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


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

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



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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

C++ Example: using basic methods

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


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



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

```
  if ( rc == FMC_OK && wList.size() >= 2 )
  {
    workitem1= wList[0];               // assign first element
    if ( workitem1.IsComplete() )
      cout << "Surprise - more than primary data available" << endl;
    else
      cout << "Primary data of first workitem available" << endl;



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


                                       // copy workitem
    FmcjWorkitem workitem3(workitem1);
   if ( workitem1 == workitem3 )
      cout << "Workitems represent same persistent object" << endl;
    else
      cout << "Surprise - workitems are not equal" << endl;
   }                                    // destructors called automatically


  // logoff
  rc = service.Logoff();

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

Accessor functions/methods

Accessor functions/methods are provided so that properties of transient objects can be read or changed. If the transient object represents a persistent one, then the values that are returned reflect the state of the persistent object when it was retrieved and used to set the transient object or when it was created or updated. Retrieval is done from an MQ Workflow server using the appropriate create, query, or refresh functions/methods. 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 *not complete*, or when the accessed property is *optional* and not set.

Default values are: an empty string or buffer for character-valued properties, 0 (zero) for integer-valued properties, false for boolean-valued properties, a

timestamp with all members set to 0 (zero) for time-valued properties, "NotSet" for enumeration-valued properties, and an empty vector for multi-valued properties.

A transient object just constructed in C++, 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 function/method can be executed on an empty object.

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 returned when objects are queried. Secondary properties, and a refresh of the primary properties, are only returned on an explicit Refresh() request; on a per-object basis. You can use the *IsComplete()* function/method to determine whether both primary and secondary object values have been read from the server.

Note: The ActiveX API automatically refreshes an incomplete object when a secondary property is accessed.

Besides being primary or secondary, properties of a persistent object can be optional. This means that they can carry a value or not. When a default value is returned to you, you can use the *IsNull()* function/method 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.

Note that being Null is a concept orthogonal to being completely read. As long as the object is not complete, IsNull() will return true for a secondary, optional property because nothing is known yet about the actual value and whether it has been set or not. For example, the documentation is a secondary and optional property of an object. When the object has been queried, then only the primary properties have been retrieved from the server. The Documentation() function/method returns an empty string or buffer. To determine whether a documentation has been set at all, you can use the DocumentationIsNull() function/method. The result will be "true" independent from the actual documentation setting as long as IsComplete() returns false. The documentation is assumed to be not set as long as the secondary data has not been retrieved.

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

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

Return codes

Accessor functions/methods provide the value asked for as their return value. Default values are returned when an error occurred during the execution of the accessor function/method. 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 function/method 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_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_BUFFER(800)
The buffer provided is too small to hold the largest possible value. See file fmcmxcon.h for required lengths.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

Accessor functions/methods 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.

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.

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.

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

  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

- AuditSetting/ FmcjProcessTemplateAuditSetting/ FmcjProcessTemplate::AuditSetting/ com.ibm.workflow.api.ProcessTemplatePackage.AuditSetting

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

- AIEscalation/ FmcjActivityInstanceEscalation/ FmcjActivityInstance::Escalation/ com.ibm.workflow.api.ActivityInstancePackage.Escalation

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

  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

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

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::suspendting

Java(15) ExecutionState.SUSPENDING

- ActivityInstanceType/ FmcjActivityInstanceType/ FmcjActivityInstance::Type/ com.ibm.workflow.api.ActivityInstancePackage.Type

  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

- ConnectorState/ FmcjControlConnectorInstanceStateValue/ FmcjControlConnectorInstance::state/ com.ibm.workflow.api.ControlConnectorInstancePackage.EvaluationState

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

- ConnectorType/ FmcjControlConnectorInstanceType/ FmcjControlConnectorInstance::Type/ com.ibm.workflow.api.ControlConnectorInstancePackage.Type

  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

- ExeOptionsStyle/ FmcjExeOptionsStyle/ FmcjExeOptions::Style/ com.ibm.workflow.api.ProgramDataPackage.Style

  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

- ExternalOptionsTimePeriod/ FmcjExternalOptionsTimePeriod/ FmcjExternalOptions::TimePeriod/ com.ibm.workflow.api.ProgramDataPackage.TimePeriod

  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

- FmcjExecutionDataKindEnum/ FmcjExecutionData::KindEnum

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

Error(1) Indicates that execution of an asynchronous call returns an error.

ActiveX	not applicable
C-language	Fmc_DART_Error
C++	FmcjExecutionData::Error
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_ProcessInstanceNotification
C++	FmcjExecutionData::ProcessInstanceNotification
Java	not supported

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

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

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 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 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 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

- ImplementationDataProgramType/ FmcjImplementationDataType/ FmcjImplementationData::Type/ com.ibm.workflow.api.ProgramDataPackage.Type

  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

- Kind/ FmcjItemType/ FmcjItem::ItemType/ com.ibm.workflow.api.ItemPackage.ItemType

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_NOTIFICAT

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_NOTIFI

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

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

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

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

  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

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

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

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

  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

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

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

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.

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
```

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 abstract int priority() throws FmcException;

Accessing a value of type string

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

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, 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 functions/methods to read a single value (refer to "C-language vectors" on page 27).

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
```

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

system
Input. The system where the execution server runs.
systemGroup
Input. The system group where the execution server runs.

Return type
ExecutionAgent
The program execution agent which provides for the context of an activity implementation.
ExecutionService
The execution service which provides for the interface to the execution server.
Object/Handle/FmcjObject
The property value.

Declaration examples

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);
C++	FmcjReadOnlyContainer* InContainer() const;
Java	public abstract ReadOnlyContainer inContainer() throws FmcException;

Determining whether an optional property is set

This function/method 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.

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 function/method sets the specified property to the specified value.

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
```

Parameters

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

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 abstract void SetTimeout(int newValue) throws FmcException;

An example is the FmcjService::SetTimeout function/method 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 functions/methods requiring communication between the client and the server. It can be set (changed) as often as wanted. It is to be provided as microseconds. 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 an object valued property

This function/method 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 function/method 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 function/method 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;


 FmcjGlobalConnect();


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

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



 /* assumption: workitem has been queried from the server             */
                                        /* access a value of type bool  */
  if ( FmcjWorkitemCategoryIsNull( workitem ) )
    printf( "Category is not set\n" );
  else                                  /* access a value of type char  */
  {                                     /* use a buffer which fits      */
    FmcjWorkitemCategory( workitem, category, FMC_CATEGORY_NAME_LENGTH+1 );
    printf( "Category   : %s\n", category );
  }


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


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


                                       /* access an enumerated value   */
  enumValue= FmcjWorkitemReceivedAs( workitem );
  if ( enumValue == Fmc_IR_Normal )
    printf( "Received as: %s\n","qualified user" );
  ...
                                        /* access a multi-valued field  */
   sList= FmcjWorkitemSupportTools( workitem );
   printf( "Support tools: " );
   for( i=0; i< FmcjStringVectorSize(sList); i++ )
   {                                    /* use a large buffer           */
     printf("%s ", FmcjStringVectorNextElement(sList, generalBuffer, 200) );
   }


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

C++ example: accessing values

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
  FmcjGlobal::Connect();
   // logon
  FmcjExecutionService service;   APIRET rc = service.Logon("USERID", "password");
                                        // set the timeout for requests
  service.SetTimeout( 60000 );


  // assumption: workitem has been queried from the server
                                         // access a value of type bool
  if ( workitem.CategoryIsNull() )
    cout << "Category is not set" << endl;
  else                                  // access a value of type char
  {                                     // use a buffer which fits
    cout << "Category   : " << workitem.Category()<< endl;
  }


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


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


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


  vector<string> tools; int j;          // access a multi-valued field
  workitem.SupportTools( tools );
  cout << "Support tools: " ;
  while ( j < tools.size() )
    cout << tools[j++] << " ";
   // logoff
  rc = service.Logoff();
   FmcjGlobal::Disconnect();
  return FMC_OK;
}                                       // destructors called automatically
```

Action functions/methods

Action functions/methods are client-server calls, involving communication with an MQ Workflow server. As such, they require to be logged on.

Action functions/methods 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 functions/methods are either synchronous requests waiting for the server's reply, asynchronous requests expecting the server's reply at some other point in time, or functions/methods receiving information from an MQ Workflow server.

All action function/methods are described separately in "Part 6. Programming interfaces" on page 259. You can find examples in "Part 8. Examples and scenarios" on page 637.

Activity implementation functions/methods

An activity or support tool can be implemented by a program which uses the MQ Workflow API. In this case, the activity implementation functions/methods provide access to the input and output containers of the activity instance respectively 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() function/method 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.

Besides being an Executable, the activity implementation or support tool program can also be a dynamic link library (DLL) or shared library. 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 67. An activity implementation" on page 677 for activity implementation examples.

Program execution management functions/methods

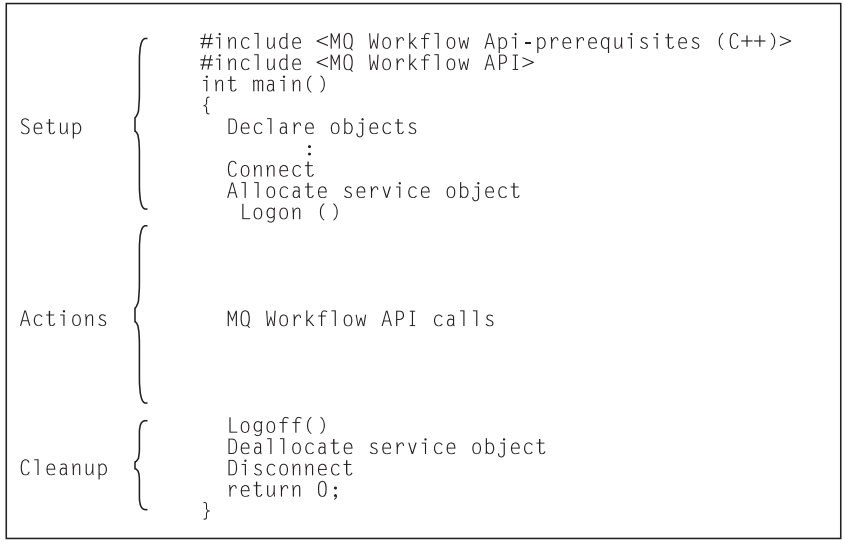
Program execution management functions/methods provide for the management of MQ Workflow program execution agents. They allow for a user-associated program execution agent to be started and to be stopped (shutdown).

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 12. An MQ Workflow client application

An MQ Workflow C or C++ 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 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() function/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. 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 functions/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 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 13. 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

Actions         {   read values
                    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() function/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. 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. [4]

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 18. An MQ Workflow client application" on page 153) and request services from an MQ Workflow server, normally the server from where its execution had been triggered. The Passthrough() function/method is then used instead of the Logon() function/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.

4. 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 14. 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. The shared libraries are linked to /usr/lib.
- 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 OS/2, in the \fmcos2\api directory on your selected drive.
- 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 Windows NT, in the \fmcwinnt\api directory on your selected drive.
- For Windows 95, in the \fmcwin95\api directory on your selected drive.
- For Windows 98, in the \fmcwin98\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: *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 139 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 by the Hewlett-Packard Company. Documentation of this library is provided on the MQ Workflow CD-ROM in a file named STLDOC.PS. It is installed in the stl subdirectory of the API.

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 (.lib files) 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 should use the multi-thread 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 C Set++(R) Version 3.1.4
- For HP-UX, HP aC++ Compiler S700 Version A.01.15.01
- For OS/2, IBM VisualAge(R) for C++ 3.0
- For Solaris and the C-language, Sun WorkShop Compiler Version 4.2

  For Solaris and C++, Kuck&Associates Inc. KAI C++ Version 3.3
- For the Windows platforms, IBM VisualAge for C++ 3.5 and Microsoft Visual C++ 5.0

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	compiler type	MQ Workflow
HP-UX	compiler type	compiler include
OS/2	MQ Workflow	MQ Workflow
Solaris	compiler type	MQ Workflow
Windows	compiler type	MQ Workflow

Sample compile statements

Sample compile statements are:

- For AIX:

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

  Note: If you include bool.h shipped with MQ Workflow, you must additionally specify the compile option EXM_AIX.
- For HP-UX:

  ```
  aCC _D_THREAD_SAFE -DRWSTD_MULTI_THREAD _D_REENTRANT
      -o <object file> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
  ```
- For OS/2 and IBM VisualAge for C++ 3.0:

  ```
  icc /GM+ <optional parameters> <source file>
  ```
- For Solaris and the C-language:

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

- For Solaris and C++:

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

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

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

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

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

Chapter 15. 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-API) or the FmcjExecutionService 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 16. 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 function/method 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;
}
```

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;
}
```

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.

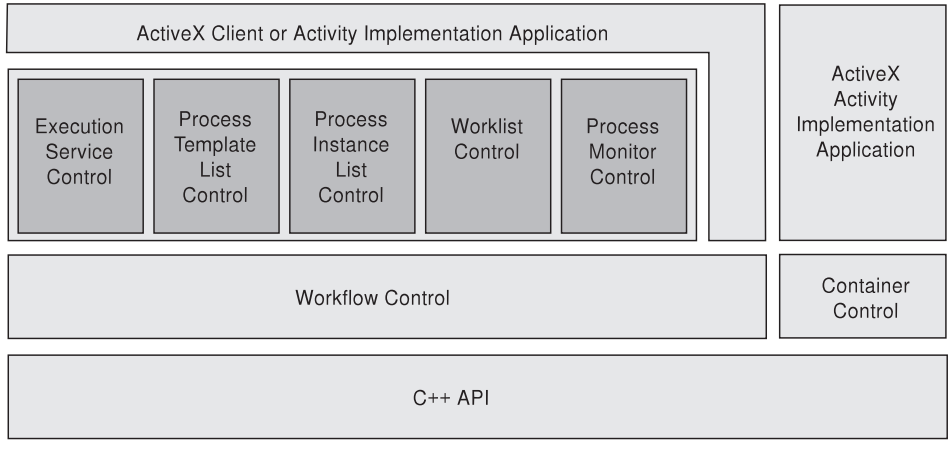
Part 3. ActiveX Controls

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

Chapter 17. 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 8. Examples and scenarios” on page 637.

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 32 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.

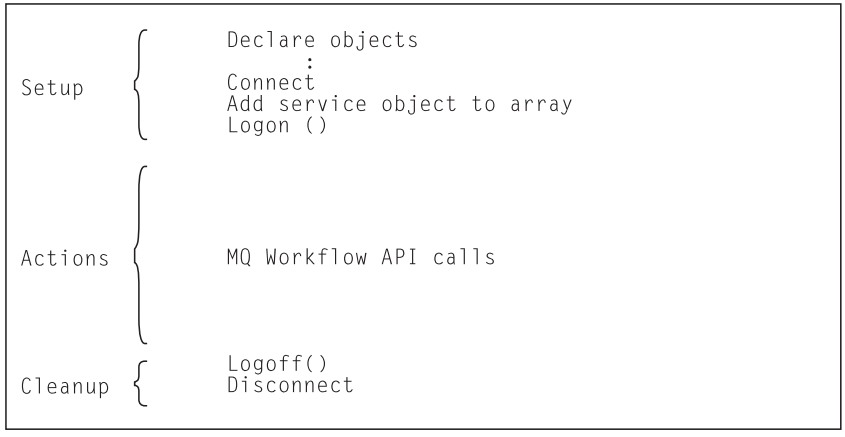
The WorkitemArray object, for example, can create and maintain any number of Workitem objects. There are methods for adding, retrieving, and deleting array elements as well as determining the number of entries in the array.

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 18. 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.

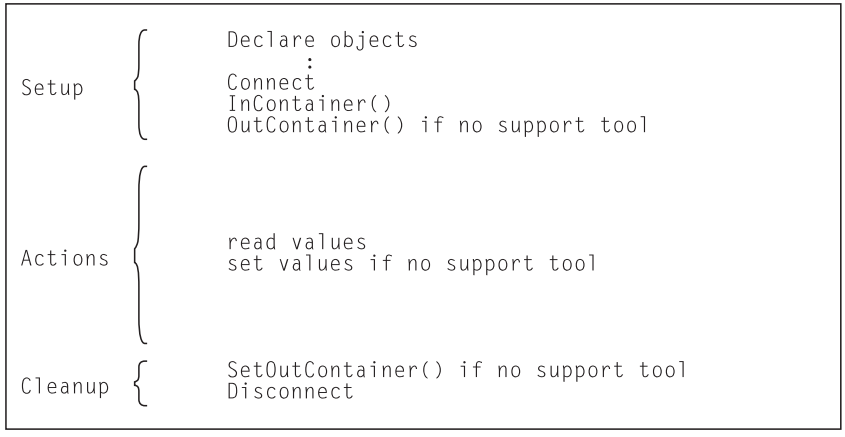
After a successful logon, you can issue action or program execution management methods in order to query or manage MQ Workflow objects you are authorized for.

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 19. 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.

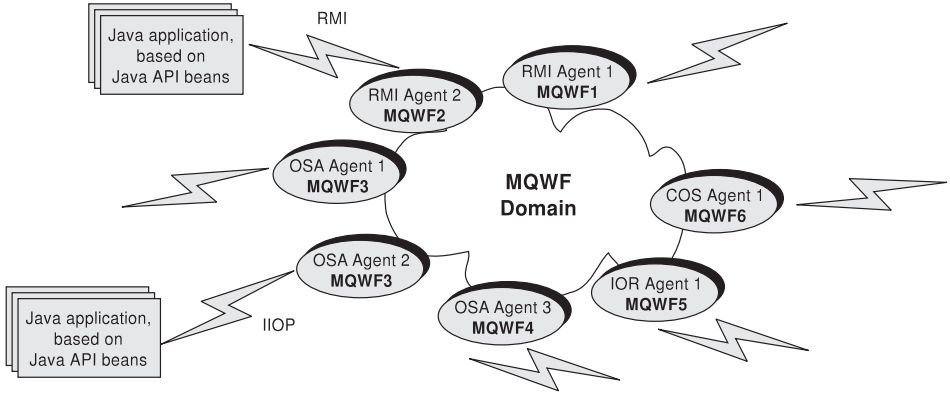
Your activity implementation can as well behave like a client application (see "Chapter 12. An MQ Workflow client application" on page 133) and request services from an MQ Workflow server, normally the server from where its execution had been triggered. The Passthrough() method is then used instead of 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:

- An agent that connects an MQSeries Workflow domain to the Java world.
- A set of API Beans that provide MQSeries Workflow API functionality to Java based applications.

Chapter 20. The Java CORBA Agent



In order to support *thin clients*, a Java agent approach has been chosen. The Java CORBA Agent serves as a proxy for the MQ Workflow domain.

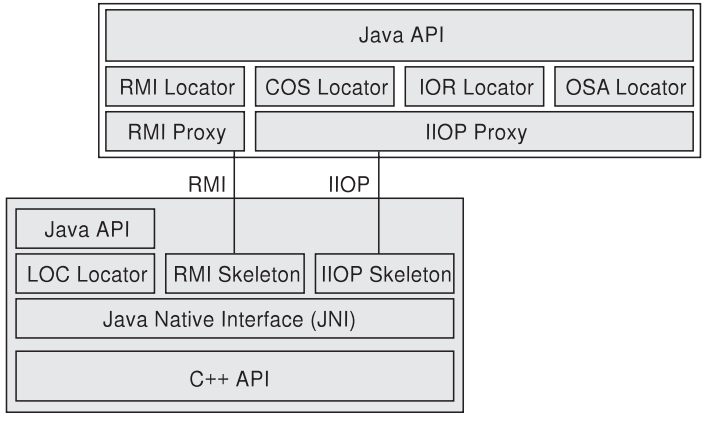
The Java CORBA Agent is implemented in Java and wraps the MQ Workflow C++ API into a form that is accessible from the Java environment. On one side, the Java CORBA Agent thus wraps the native product APIs and on the other side publishes a Java form of the APIs on the network.

Chapter 21. The communication layer

The Java CORBA Agent is running on an MQ Workflow machine and Java clients are running somewhere on the network. MQ Workflow supports a CORBA, RMI, and Local environment so that clients can access the agent.

- CORBA is the Object Management Group (OMG) standard for distributed computing. It is very easy to publish existing objects on a network using ORBs. The currently supported ORB is Inprise's VisiBroker Java 3.3.
- Java Remote Method Invocation (RMI) is an approach that is completely Java based and does not require additional software. RMI is included in most Java environments.
- Local bindings offer a special mechanism which imbeds the Java CORBA Agent. They bypass the communication layer and use procedure calls. If client applications use local bindings, then they have to consider the trade-off that they become MQSeries clients. It follows that local bindings are best suited for agent side applications, for example, servlets and Java-based non-GUI activity implementations.

Chapter 22. The locator methods



There is a multitude of methods available how the clients can find their agent. The different methods that are supported by MQ Workflow are listed below:

- OSA naming: a VisiBroker specific naming facility (Smart Agent) that is very easy to use. It only requires one OSAgent running on the subnetwork that keeps track of all the objects and their name in the network. As smart agents synchronize their information via UDP, the only thing that has to be known is the name of the object the client program is looking for.
- IOR naming: Via InterOrbReferences a vendor-independent naming service for CORBA applications exists. The stringified idendity of a specific object (its IOR) is published in a file on a Web server. This file can be accessed from clients via a published URL to obtain the actual reference of an object.
- COS naming: Corba Naming Service is the native CORBA directory service. Objects can use COS to publish their identity to the CORBA system.
- RMI registry: The RMI registry comes with every Java development kit. It can be run as a stand-alone program where object implementations register or it can be embedded into the application. Embedding has the big advantage that no separate program is necessary to provide naming functions. To locate objects via the RMI registry, the host which runs the RMI registry has to be known.
- LOC naming: This approach can be used to connect the Java API to an MQ Workflow C++ API that is located on the same physical machine. This approach can be useful if a client should be written on a platform that offers the APIs but does not offer a native client, for example, on AIX. It can

also be used to access the MQ Workflow APIs from a Web server through servlet technology without the additional communication overhead because local bindings use procedure calls.

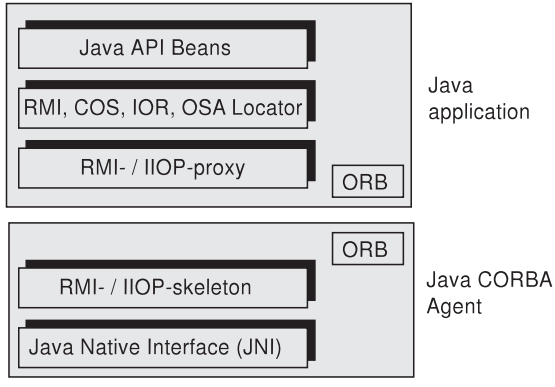
Chapter 23. The Java API Beans

Both, the client side communication layer and the API Beans layer are implemented in Java. This makes it possible to run applications developed with the MQSeries Workflow Java API on any machine that provides a Java Virtual Machine.

The API Beans provide functionality equivalent to the other MQ Workflow APIs. Due to the introduction of an agent, its name, context, and locator policy have, however, to be specified.

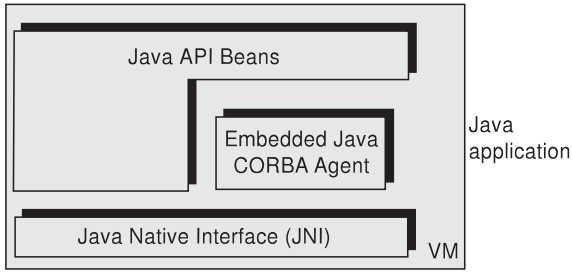
Following are some usage scenarios of the Java API.

Java in the intranet



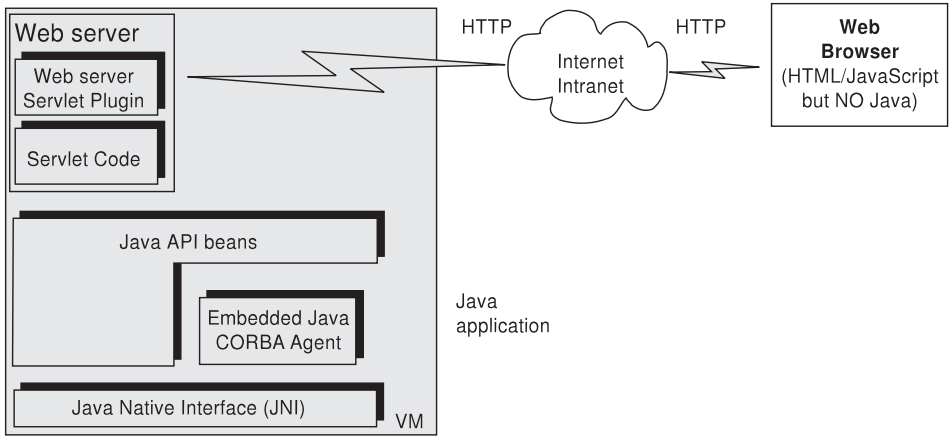
In this case, a non-LOC locator policy must be used and an external agent must be specified. The API Beans and, for a non-RMI protocol, the VisiBroker ORB (COS,IOR,OSA) must have been installed.

Java as a programming language



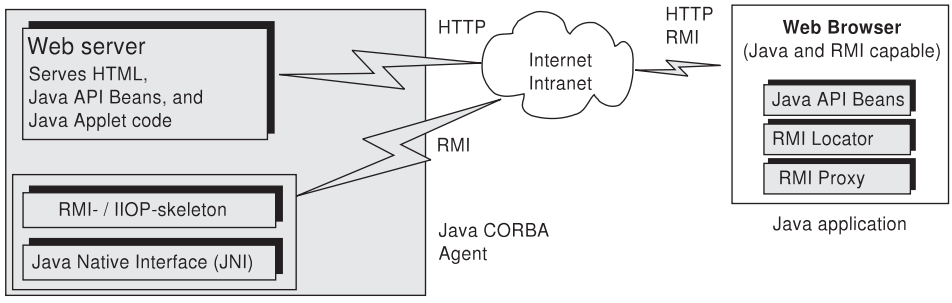
In this case, the LOC_LOCATOR policy of the Java CORBA Agent is used. The Java API Beans must have been installed.

Java in the Internet (Servlet)



In this case, a LOC_LOCATOR policy must be used. The API Beans must have been installed.

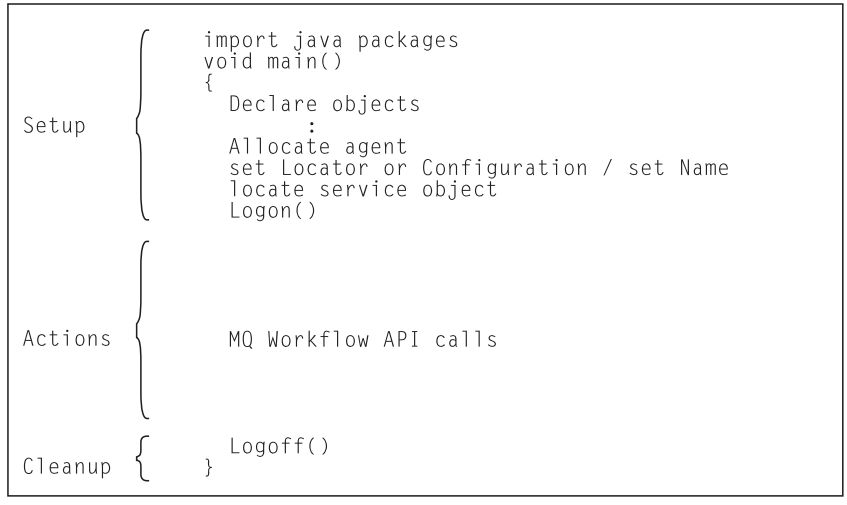
Java in the Internet (Applet-RMI)



In this case, an RMI_LOCATOR policy must be used and an RMI agent must be specified. An RMI Agent and the Java API Beans must have been installed. The applet must be specified in the context of the agent object.

Chapter 24. 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.

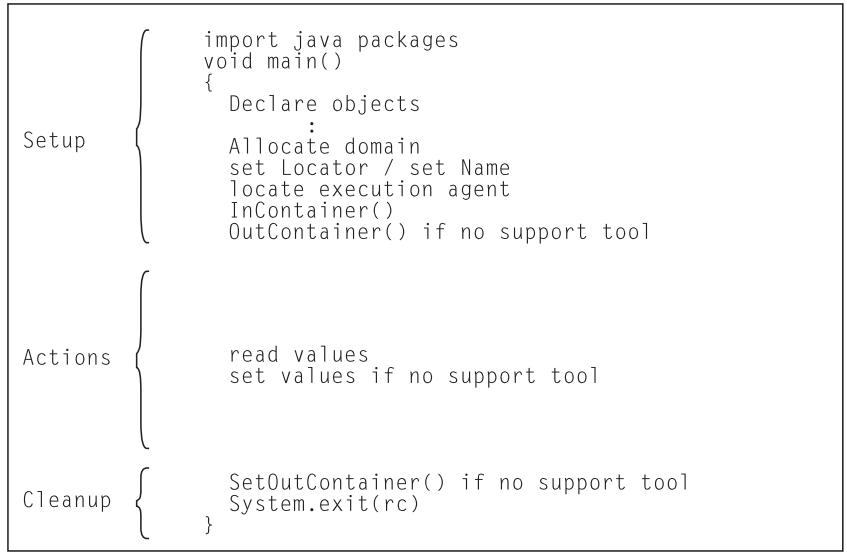
You then need to access a service object which represents the server you are going to ask services from. You do this by locating it via an appropriate agent. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your 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 25. An MQ Workflow activity implementation or support tool

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 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 12. An MQ Workflow client application" on page 133) 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.

Chapter 26. Compiling

All programs developed for use with the MQ Workflow Java API Beans 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\java320 subdirectory of the installation directory.

JDK 1.1.x (x=6 or higher) can be used to compile and run your applications accessing the MQ Workflow Java API. A sample compile statement is:

```
javac -O <java file>.java
```

-O is an optional parameter denoting an optimized build. The CLASSPATH must point to fmcojapi.jar.

Depending on the used locator policies, access can be gained to the Java CORBA Agent:

LOC, RMI
The CLASSPATH environment variable must point to fmcojagt.jar. For example,

```
CLASSPATH=.;d:\fmcwinnt\bin\java320\fmcojagt.jar;
```

OSA, IOR
The CLASSPATH environment variable must point to fmcojagt.jar and to the Inprise VisiBroker Java 3.2 (plus PatchPack 3) or higher. For example,

```
CLASSPATH=.;d:\fmcwinnt\bin\java320\fmcojagt.jar;
           d:\inprise\vbroker\lib\vbjapp.jar;
           d:\inprise\vbroker\lib\vbjorb.jar;
           d:\inprise\vbroker\lib\vbjtools.jar;
```

COS The CLASSPATH environment variable must point to fmcojagt.jar and to the Inprise VisiBroker Java 3.2 (plus PatchPack 3) or higher. Additionally the CORBA naming service vbjcosnm.jar must be specified. For example,

```
CLASSPATH=.;d:\fmcwinnt\bin\java320\fmcojagt.jar;
           d:\inprise\vbroker\lib\vbjapp.jar;
           d:\inprise\vbroker\lib\vbjorb.jar;
           d:\inprise\vbroker\lib\vbjtools.jar;
           d:\inprise\vbroker\lib\vbjcosnm.jar
```

Chapter 27. 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 object 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. Examples are the Agent or the ExecutionService object, which allows services to be requested from an execution server.

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

Although the 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, 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.

Garbage Collection when using Java API Beans

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 Beans, the user is provided with a build-in garbage collection mechanism, the MQ Workflow Java API Beans 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 Beans Agent a set of parameters controlling the reaper have to be set. These control parameters are:

- The reaper cycle time value, defined in milliseconds, is valid for both the client's reaper and the server's reaper. 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. Using the MQ Workflow APIs

Chapter 28. 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.

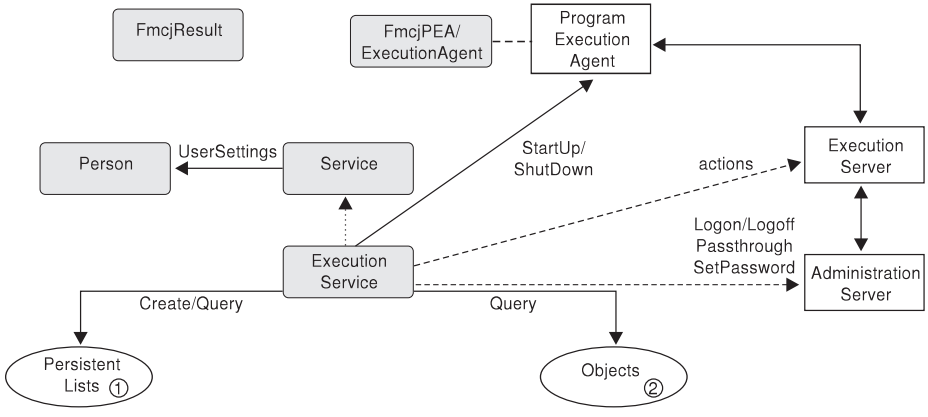


Figure 3. Setting up client/server communication. Legend: -➔ Inheritance (C++); ⟶ provides for access; — ⟶ sends messages to

As a first step, an FmcjExecutionService or ExecutionService object must be obtained (constructed/allocated/located). An FmcjExecutionService or ExecutionService object represents a session between a user and an MQ

Workflow execution server. It essentially provides the basic functions/methods 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 function/method 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.

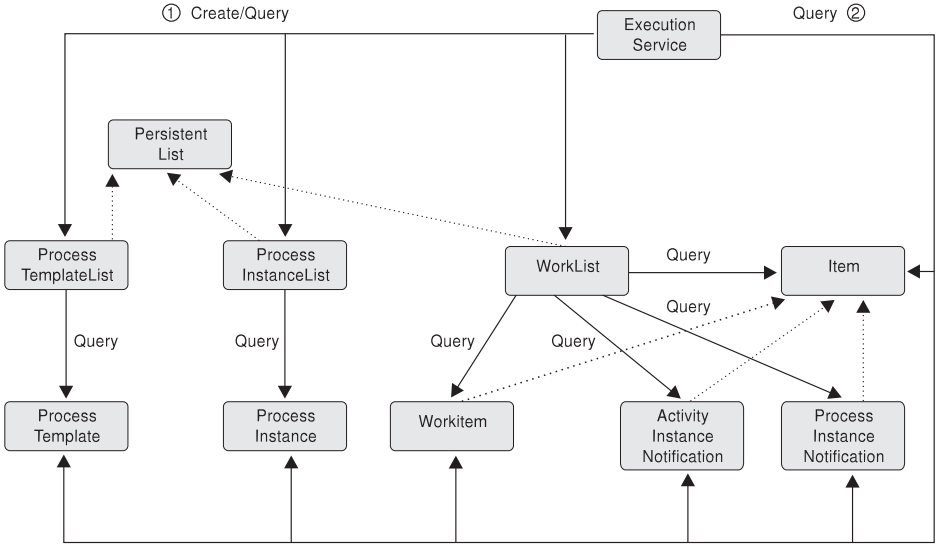


Figure 4. Querying objects. Legend: --➔ Inheritance (C++); ⟶ provides for access

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.

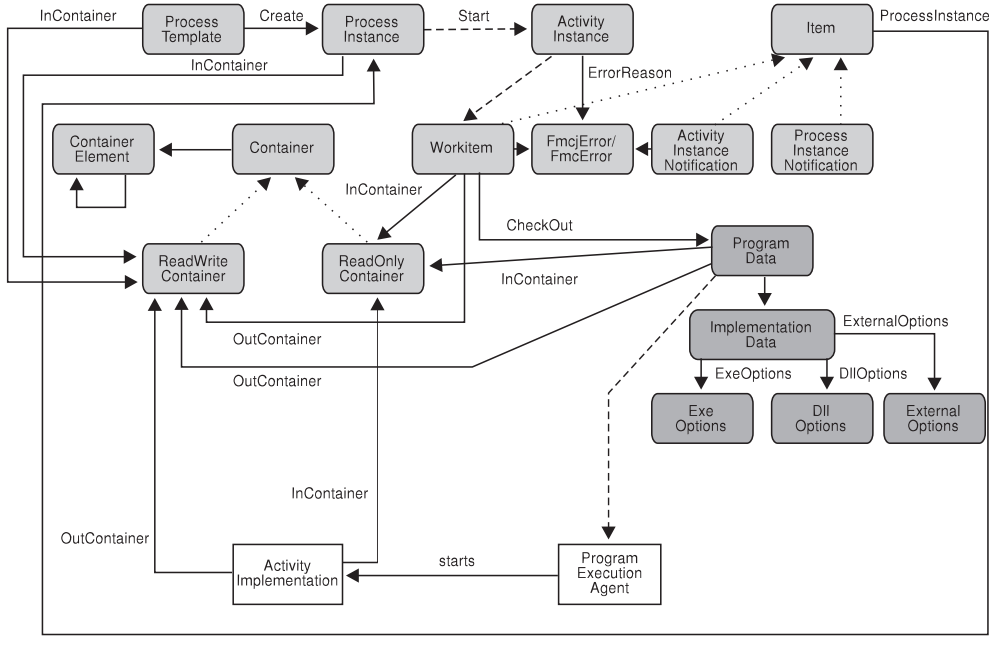


Figure 5. Dealing with process instances and (work) items. Legend: -➝ Inheritance (C++); ⟶ provides for access; — ⟶ data is passed to or results in

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 functions/methods for reading and wrting values. See “Chapter 8. Handling containers” on page 37 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 functions/methods 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.

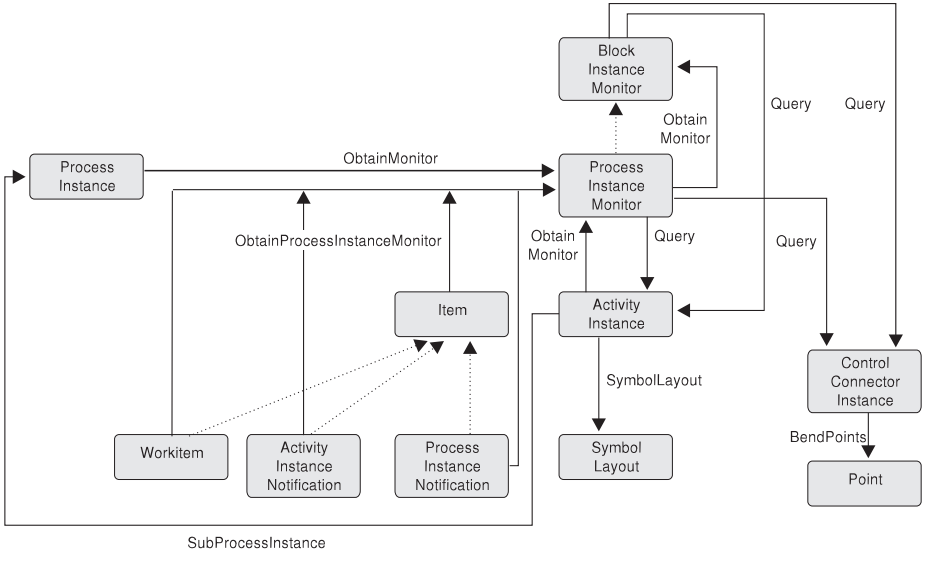


Figure 6. Monitoring a process instance. Legend: --➔ Inheritance (C++); ⟶ provides for access

When a process instance or item, that is, a work item, an activity instance notification, or a process instance notification, has been retrieved, you can obtain the associated process instance monitor. The process instance monitor then allows for analyzing the states of activity instances and control connector instances. The path taken through the process instance can thus be

determined. In case you want to present this information graphically, the activity instance symbol layout and the control connector instance positions and bend points offer support.

Once a process instance monitor has been obtained, you can iterate into the process model by obtaining block instance monitors for activities of type Block or process instance monitors for activities of type Process, that is, for subprocess instances. See "Chapter 9. Monitoring a process instance" on page 71 for more information.

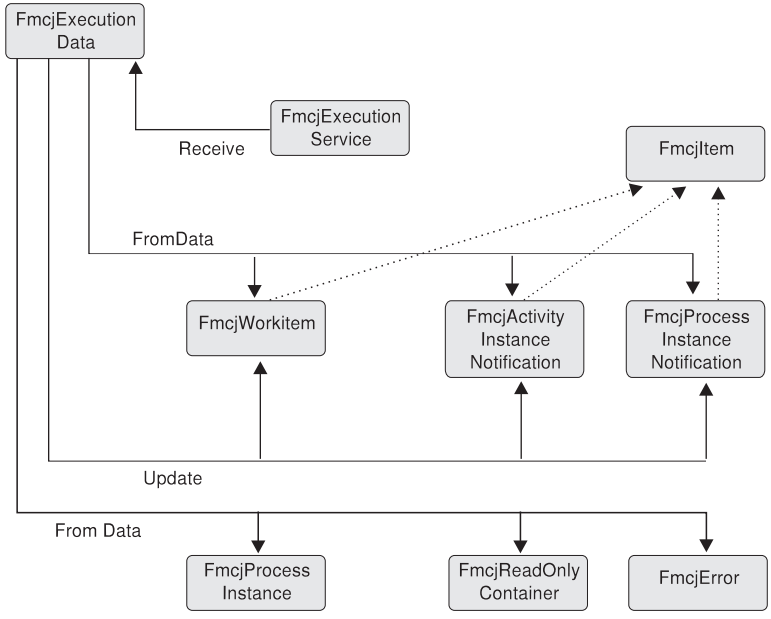


Figure 7. Handling data sent by an MQ Workflow server. Legend: --➔ Inheritance (C++); ⟶ provides for access

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 17 for more information.

API classes/objects

An alphabetical list of classes respectively a list of function prefixes in the C-language follows. All functions/methods following this list are valid calls on the respective objects. The stated names are valid ActiveX or Java classes.

To become valid C++ classes, a prefix of Fmcj has to be added. To become valid C-language function calls, the class name has to be prefixed by Fmcj and extended by the actual function name. For example, if your ActiveX class is Workitem, then your C++ class is named FmcjWorkitem and all your functions in the C-language start with FmcjWorkitem; FmcjWorkitemStart is a supported 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.
BlockInstanceMonitor	The monitor for an activity instance of kind Block; see InstanceMonitor in ActiveX.
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	The ActiveX representation of date and time values. FmcjCDateTime is the C-language equivalent structure. The C++ class is called FmcjDateTime. Java uses the Calendar object.

Class/Object	Description
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.
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 functions/methods which are global API functions/methods in C and C++.
ImplementationData	The program implementation definitions.
InstanceMonitor	The monitor for a process instance or an activity instance in ActiveX.
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.

Class/Object	Description
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.
ProcessInstanceListArray	The ActiveX result of a query for process instance lists.
ProcessInstanceListVector	The C-language result of a query for process instance lists.
ProcessInstanceMonitor	The monitor for a process instance; see InstanceMonitor in ActiveX.
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.
ReadOnlyContainer	A data container that can only be read.
ReadWriteContainer	A data container that can be read and written to.
Result	The detailed result of a request; only C-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.

Functions/methods per object

Activity instance

An activity instance represents an instance of a process template activity. It is part of a process instance.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an activity instance object.	80
Copy()	Allocates and initializes the storage for an activity instance object by copying.	83
Deallocate()	Deallocates the storage for an activity instance object.	84
destructor()	Destructs an activity instance object.	84
Equal()	Compares two activity instances.	82
IsComplete()	Indicates whether the complete activity instance information is available.	84
IsEmpty()	Indicates whether no activity instance information is available.	85
Kind()	States the kind of the activity instance, whether it is a program, a process, or a block.	86
operator=()	Assigns an activity instance to this one.	82
operator==()	Compares two activity instances.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance. Note that the activity instances returned by the (process or block) instance monitor contain both primary and secondary values.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method 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.	92
ActivationTimeIsNull()	P/B	Indicates whether an activation time is set.	121
Category()	P/C	Returns the process category of the activity instance.	116
CategoryIsNull()	P/B	Indicates whether a category is set.	121
Description()	P/C	Returns the description of the activity instance.	116
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the activity instance.	116
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the ending time of the activity instance.	92
EndTimeIsNull()	S/B	Indicates whether an end time is set.	121
ErrorReason()	S/O	Returns an error object describing the reason why the activity instance is in state InError.	119
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	121

Accessor methods	Set/ Type	Description	Page
ExitCondition()	S/C	Returns the exit condition of the activity instance.	116
FirstNotificationTime()	S/D	Returns the time the first notification for the activity instance is to occur or has occurred.	92
FirstNotificationTimeIsNull()	S/B	Indicates whether a first notification time is set.	121
FirstNotifiedPersons()	S/M	Returns the persons who received a first notification for the activity instance.	118
FullName()	P/C	Returns the fully qualified name of the activity instance (dot notation).	116
Icon()	P/C	Returns the icon associated with the activity instance.	116
Implementation()	P/C	Returns the name of the implementing program of the activity instance.	116
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	121
InContainerName()	S/C	Returns the name of the input container of the activity instance.	116
LastModificationTime()	P/D	Returns the last time a primary attribute of the activity instance was changed.	92
LastStateChangeTime()	P/D	Returns the last time the state of the activity instance changed.	92
ManualExitMode()	S/B	Returns whether the exit mode of the activity instance is manual.	91
ManualStartMode()	S/B	Returns whether the start mode of the activity instance is manual.	91
Name()	P/C	Returns the name of the activity instance.	116
OutContainerName()	S/C	Returns the name of the output container of the activity instance.	116
PersistentOid()	P/C	Returns a representation of the object identification of the activity instance.	116
Priority()	P/I	Returns the priority of the activity instance.	115

Accessor methods	Set/ Type	Description	Page
PriorityIsNull()	P/B	Indicates whether a priority is set.	121
ProcessAdmin()	S/C	Returns the process administrator of the activity instance.	116
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	121
ProcessInstanceName()	P/C	Returns the name of the process instance the activity instance is part of.	116
ProcessInstanceState()	P/E	Returns the state of the process instance the activity instance is part of.	93
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	116
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the activity instance is part of.	116
SecondNotificationTime()	S/D	Returns the time the second notification for the activity instance is to occur or has occurred.	92
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	121
SecondNotifiedPersons()	S/M	Returns the persons who received a second notification for the activity instance.	118
Staff()	S/M	Returns all persons a work item for the activity instance has been assigned to.	118
StartCondition()	S/C	Returns the start condition of the activity instance.	116
Starter()	P/C	Returns the starter of the activity instance.	116
StarterIsNull()	P/B	Indicates whether a starter is set.	121
StartTime()	P/D	Returns the start time of the activity instance.	92
StartTimeIsNull()	P/B	Indicates whether a start time is set.	121
State	P/E	Returns the state of the activity instance.	93

Accessor methods	Set/ Type	Description	Page
StateOfNotification()	S/E	Returns the notification state of the activity instance.	93
SupportTools()	P/M	Returns the support tools associated with the activity instance.	118
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	121
SymbolLayout()	S/O	Returns the symbol layout of the activity instance.	119

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
ObtainInstanceMonitor()	Retrieves the process instance monitor for the process instance the activity instance is part of in ActiveX.	395
ObtainProcessInstanceMonitor()	Retrieves the process instance monitor for the process instance the activity instance is part of.	261
SubProcessInstance()	Retrieves the process instance implementing the activity instance of type Process.	264

Activity instance array

An activity instance array represents the result of a query for activity instances in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the activity instance array.	34

Activity instance notification

An activity instance notification represents a notification for an activity instance. **All functions/methods of FmcjItem are also applicable to activity instance notifications.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an activity instance notification object.	80
Copy()	Allocates and initializes the storage for an activity instance notification object by copying.	83
Deallocate()	Deallocates the storage for an activity instance notification object.	84
destructor()	Destructs an activity instance notification object.	84
Kind()	In the C++ language, states that the object is an activity instance notification.	86
operator=()	Assigns an activity instance notification to this one.	82
operator==()	Compares two activity instance notifications.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instance notifications are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance notification.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method 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.	93
ErrorReason()	S/O	Returns an error object describing the reason why the associated activity instance is in state InError.	119
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	121
ExitCondition()	S/C	Returns the exit condition of the associated activity instance.	116

Accessor methods	Set/ Type	Description	Page
Expired()	P/B	Returns whether the associated activity instance has been started and is expired now.	91
FirstNotificationTime()	S/D	Returns the first notification time of the activity instance, that is, the time when this notification has been created.	92
Implementation()	P/C	Returns the implementing program or process name of the associated activity instance.	116
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	121
ManualExitMode()	S/B	Returns whether the exit mode of the associated activity instance is manual.	91
ManualStartMode()	S/B	Returns whether the start mode of the associated activity instance is manual.	91
Priority()	P/I	Returns the priority of the associated activity instance.	115
SecondNotificationTime()	S/D	Returns the second notification time of the associated activity instance.	92
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	121
Staff()	S/M	Returns all persons owning a work item for the associated activity instance.	118
StartCondition()	S/C	Returns the start condition of the associated activity instance.	116
StartOverdue()	P/B	Returns whether the start of the associated activity instance is overdue.	91
State	P/E	Returns the state of the associated activity instance.	93
StateOfNotification()	S/E	Returns the notification state of the associated activity instance.	93
SupportTools()	P/M	Returns the support tools associated with the activity instance.	118

Accessor methods	Set/ Type	Description	Page
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	121

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
PersistentObject()	Retrieves the specified activity instance notification.	267
StartTool()	Starts the specified support tool.	270
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	395

Activity instance notification array

An activity instance notification array represents the result of a query for activity instance notifications in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Activity instance notification vector

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 27 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.

Vector methods	Description
Size()	Returns the number of elements in the activity instance notification vector.

Activity instance vector

An activity instance vector represents the result of a query for activity instances in the C-language.

Refer to "C-language vectors" on page 27 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates an activity instance vector object.
FirstElement()	Returns the first element of the activity instance vector.
NextElement()	Returns the next element of the activity instance vector.
Size()	Returns the number of elements in the activity instance vector.

Agent

An agent object represents an MQ Workflow instance in Java.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an agent object. Initially an agent has no context, locator policy, or name.	80

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	123

Accessor methods	Type	Description	Page
addVetoableChangeListener()	O	Adds the specified listener to the set of listeners to be notified of vetoable property changes.	123
getConfigurationID()	C	Returns the configuration to be used for profile accesses.	116
getExecutionAgent()	O	Returns a program execution agent to the calling activity implementation provided that the LOC_LOCATOR policy was used. Otherwise, null is returned.	119
getLocator()	I	Returns the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR.	115
getName()	C	Returns the name of the Java Agent. If the agent is not bound, an empty string is returned.	116
isBound()	B	Indicates whether the agent bean is bound to a Java CORBA agent.	91
locate()	O	Locates the execution service in the provided system group and system.	119
removePropertyChangeListener()	O	Removes the specified listener from the set of listeners.	123
removeVetoableChangeListener()	O	Removes the specified listener from the set of listeners.	123
setConfigurationID()	C	Sets the configuration ID to be used for profile access. A locator policy of LOC_LOCATOR is automatically assumed.	116
setContext()	O	Sets the context of the agent. An applet must set the context by issuing a *agent.setContext(this,null);*	119
setLocator()	I	Sets the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR. This call must precede the Agent.setName(). If LOC_LOCATOR is set, the default configuration ID for profile access is automatically used.	122
setName()	C	Sets the name of the Java Agent.	116

Accessor methods	Type	Description	Page
toString()	C	Returns the name of the agent.	116

Block instance monitor

A block instance monitor object represents a monitor of an activity instance of type *Block*. **All functions/methods of a block instance monitor are also applicable to process instance monitors.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
destructor()	Destructs a block instance monitor object, that is, the transient representation in the C++ interface. The internal block instance monitor object is, however, not deallocated since it is part of the process instance monitor. It is deallocated when the process instance monitor is destructed/deallocated.	84

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. All properties are primary because a block instance monitor is a part of a process instance monitor.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
ActivityInstances()	M	Returns the activity instances which are represented by the block instance monitor, that is, which are part of the activity instance of type *Block*. The activity instances contain both primary and secondary values.	118
ControlConnectorInstances()	M	Returns the control connector instances which are represented by the block instance monitor, that is, which are part of the activity instance of type *Block*.	118

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
ObtainBlockInstanceMonitor()	Returns the block instance monitor for an activity instance of type *Block*. The activity instance is part of the set of activity instances represented by the block instance monitor.	273
ObtainProcessInstanceMonitor()	Returns the process instance monitor for an activity instance of type *Process*. The activity instance is part of the set of activity instances represented by the block instance monitor.	275
Refresh()	Refreshes the block instance monitor from the MQ Workflow execution server.	278

Container

A container represents an input or output data container of a process instance or work item. **All functions/methods of a container are applicable to read-only and read/write containers.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
IsEmpty()	Indicates whether no container information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	45
AllLeaves()	M	Returns all leaf elements of the container including the MQ Workflow predefined members.	45

Accessor methods	Type	Description	Page
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.	55
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.	55
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.	55
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.	55
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.	55
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.	55
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.	55
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.	55
FloatValue()	F	Returns the value of the specified container leaf element in the C-language. The leaf is of type FLOAT.	55
getBuffer()	C	Returns the value of the specified container leaf element in Java. The leaf is of type BINARY.	55
getBuffer2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type BINARY.	55
getDouble()	F	Returns the value of the specified container leaf element in Java. The leaf is of type FLOAT.	55
getDouble2()	F	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type FLOAT.	55

Accessor methods	Type	Description	Page
GetElement()	O	Provides access to a container element.	54
getLong()	I	Returns the value of the specified container leaf element in Java. The leaf is of type LONG.	55
getLong2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type LONG.	55
getString()	C	Returns the value of the specified container leaf element in Java. The leaf is of type STRING.	55
getString2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type STRING.	55
LeafCount()	I	Returns the number of user-defined leaf elements of the container.	55
Leaves()	M	Returns all user-defined leaf elements of the container.	55
LongValue()	I	Returns the value of the specified container leaf element in the C-language. The leaf is of type LONG.	55
MemberCount()	I	Returns the number of structural members in the container.	55
StringLength()	I	Returns the length of the value of the specified container leaf element in the C-language. The leaf is of type STRING.	55
StringValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is of type STRING.	55
StructMembers()	M	Returns the structural members of the container.	45
Type()	C	Returns the type of the container, that is, the data structure name.	45
Value()	C/I/F/N	Returns the value of the specified container leaf element in the C++ language.	55

Refer to “Activity implementation functions/methods” on page 128 for detailed descriptions of activity implementation functions/methods.

Activity implementation methods	Description	Page
InContainer()	Accesses the input container from within an activity implementation; for Java see the ExecutionAgent.	281

Activity implementation methods	Description	Page
OutContainer()	Accesses the output container from within an activity implementation; for Java see the ExecutionAgent.	283
RemoteInContainer()	Accesses the input container from within a program started by an activity implementation; for Java see the ExecutionAgent.	285
RemoteOutContainer()	Accesses the output container from within a program started by an activity implementation; for Java see the ExecutionAgent.	288
SetOutContainer()	Sets the output container from within an activity implementation; for Java see the ExecutionAgent.	290
SetRemoteOutContainer()	Sets the output container from within a program started by an activity implementation; for Java see the ExecutionAgent.	292

Container array

A container array represents an array of containers in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	32
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34
RemoveAt()	Removes the element at the indicated position.	34

Container element

A container element represents an arbitrary element of a container.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a container element object.	80

Basic methods	Description	Page
Copy()	Allocates and initializes the storage for a container element object by copying.	83
Deallocate()	Deallocates the storage for a container element object.	84
destructor()	Destructs a container element object.	84
Equal()	Compares two container elements.	82
operator=()	Assigns a container element to another one.	82
operator==()	Compares two container elements.	82
IsEmpty()	Indicates whether no container element information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	60
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.	60
ArrayElements()	M	Returns the array elements of the container element.	49
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.	60
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.	60
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.	60

Accessor methods	Type	Description	Page
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.	60
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.	60
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.	60
Cardinality()	I	Returns the number of array elements of the container element.	49
FloatValue()	?	Returns the value of the specified container element leaf element in the C-language. The leaf is of type FLOAT.	60
FullName()	C	Returns the fully-qualified dotted name of the container element.	49
getBuffer()	C	Returns the value of the specified container element leaf element in Java. The leaf is of type BINARY.	55
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.	55
getDouble()	F	Returns the value of the specified container element leaf element in Java. The leaf is of type FLOAT.	55
getDouble2()	FC	Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type FLOAT.	55
GetElement()	O	Provides access to an element of the container element.	54
getLong()	I	Returns the value of the specified container element leaf element in Java. The leaf is of type LONG.	55
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.	55
getString()	C	Returns the value of the specified container element leaf element in Java. The leaf is of type STRING.	55

Accessor methods	Type	Description	Page
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.	55
IsArray()	B	Indicates whether the container element is an array.49.	
IsLeaf()	B	Indicates whether the container element is a leaf.	49
IsStruct()	B	Indicates whether the container element is a structure itself.	49
LeafCount()	I	Returns the number of leaf elements of the container element.	49
Leaves()	M	Returns all leaf elements of the container element.	49
LongValue()	I	Returns the value of the specified container element leaf element in the C-language. The leaf is of type LONG.	60
MemberCount()	I	Returns the number of structural members in the container element.	49
Name()	C	Returns the name of the container element.	49
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.	60
StringValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is of type STRING..	60
StructMembers()	M	Returns the structural members of the container element.	49
Type()	C	Returns the type of the container element, that is, the data structure name.	49
Value()	C/I/F/N	Returns the value of the specified container element leaf element in the C++ language.	60

Container element array

A container element array represents the result of a query for container elements in ActiveX.

Refer to “ActiveX arrays” on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	32
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34
RemoveAt()	Removes the element at the indicated position.	34

Container element vector

A container element vector represents the result of a query for container elements in the C-language.

Refer to "C-language vectors" on page 27 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.

Control connector array

A control connector array represents the result of a query for control connector instances in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Control connector instance

A control connector instance object represents a control connector between two activity instances and its state.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a control connector instance object.	80
Copy()	Allocates and initializes the storage for a control connector instance object by copying.	83
Deallocate()	Deallocates the storage for a control connector instance object.	84
destructor()	Destructs a control connector instance object.	84
Equal()	Compares two control connector instance objects on the basis of their source and target activity instances.	82
IsEmpty()	Indicates whether no control connector instance information is available.	85
Kind()	States the kind of the control connector instance, whether it is a transition condition or the "otherwise" connector.	86
operator=()	Assigns a control connector instance object to this one.	82
operator==()	Compares two control connector instance objects on the basis of their source and target activity instances.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	118
Name()	C	Returns the name associated with the control connector instance.	116
NameIsNull()	B	Indicates whether a name is set.	121
PersistentOidOfSourceActivity()	C	Returns the object ID of the activity instance which is the source of this control connector instance.	116
PersistentOidOfTargetActivity()	C	Returns the object ID of the activity instance which is the target of this control connector instance.	116

Accessor methods	Type	Description	Page
State()	E	Returns the state of the control connector instance, whether it is evaluated, and the result of evaluation.	93
TransitionCondition()	C	Returns the transition condition of the control connector instance.	116
TransitionConditionIsNull()	B	Indicates whether a transition condition is set.	121

Control connector instance vector

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 27 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.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods. Following methods are only available in the C++ language.

Basic methods	Description	Page
constructor()	Constructs a date/time object.	80
destructor()	Destructs a date/time object.	84
operator=()	Assigns a date/time object to another one.	82

Basic methods	Description	Page
operator==()	Compares two date/time objects.	82
operator string()	Returns the string representation of the date/time object.	116
IsEmpty()	Indicates whether no date/time information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	115
Hour()	I	Returns the hours of the date/time object.	115
Minute()	I	Returns the minutes of the date/time object.	115
Month()	I	Returns the month of the date/time object.	115
Second()	I	Returns the sceonds of the date/time object.	115
Year()	I	Returns the year of the date/time object.	115

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.	116
FmcjDateTimeCurrentTime	D	Returns the current date/time.	92
FmcjDateTimeIsValid	B	Indicates whether the passed date/time is a valid date/time.	91

Dll options

A DllOptions object represents the program implementation definitions for a dynamic link library.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a DLL options object.	80
Copy()	Allocates and initializes the storage for a DLL options object by copying.	83
Deallocate()	Deallocates the storage for a DLL options object.	84
destructor()	Destructs a DLL options object.	84
IsEmpty()	Indicates whether no DLL options information is available.	85
operator=()	Assigns a DLL options object to this one.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
ExecuteFenced()	B	States whether the DLL should run in a separate address space.	91
ExecuteFencedIsNull()	B	Indicates whether execute fended is set.	121
KeepLoaded()	B	States whether the DLL should stay loaded.	91
KeepLoadedIsNull()	B	Indicates whether keep loaded is set.	121
PathAndFileName()	C	Returns the path and file name of the DLL.	116

ExecutionAgent/FmcjPEA

A PEA or ExecutionAgent object represents an MQ Workflow program execution agent.

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. Because the following information is retrieved from the program execution agent, a distinction between primary and secondary properties is not applicable.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method declaration can be

found in a general format at the indicated page.

Accessor methods	Type	Description	Page
ProgramID()	C	Returns the program identification by which the invoked activity implementation is known to the program execution agent.	116
RemoteUserID()	C	Returns the user identification on whose behalf the activity implementation who started this program was originally started.	116
UserID()	C	Returns the user identification on whose behalf the activity implementation was started.	116

Refer to "Activity implementation functions/methods" on page 128 for detailed descriptions of activity implementation functions/methods.

Activity implementation methods	Description	Page
InContainer()	Accesses the input container from within an activity implementation in Java; for non-Java see the Container.	281
OutContainer()	Accesses the output container from within an activity implementation in Java; for non-Java see the Container.	283
RemoteInContainer()	Accesses the input container from within a program started by an activity implementation in Java; for non-Java see the Container.	285
RemoteOutContainer()	Accesses the output container from within a program started by an activity implementation in Java; for non-Java see the Container.	288
SetOutContainer()	Sets the output container from within an activity implementation in Java; for non-Java see the Container.	290
SetRemoteOutContainer()	Sets the output container from within a program started by an activity implementation in Java; for non-Java see the Container.	292

Execution data

An execution data object represents data sent from an MQ Workflow execution server.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an execution data object.	80
Copy()	Allocates and initializes the storage for an execution data object by copying.	83
Deallocate()	Deallocates the storage for an execution data object.	84
destructor()	Destructs an execution data object.	84
IsEmpty()	Indicates whether no execution data information is available.	85
Kind()	Returns the kind of the data, whether it is describing a work item creation, deletion, and so on.	86
operator=()	Assigns an execution data object to this one.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	120
ErrorFromData()	P	Creates an error description object from the execution data.	120
PersistentOid()	C	Returns a representation of the object ID of the object described by the execution data.	116
ProcessInstanceNotificationFromData()	P	Creates a process instance notification from the execution data.	120
ReadOnlyContainerFromData()	P	Creates a container object from the execution data.	120
WorkitemFromData()	P	Creates a work item from the execution data.	120
UserContext()	C	Returns the user context.	116

Accessor methods	Type	Description	Page
UserContextIsNull()	B	States whether a user context had been specified.	121

Execution service

An execution service object represents a user session to an execution server. **All functions/methods provided by FmcjService are also applicable.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
Allocate()	Allocates the storage for an execution service object. The execution service to connect to is taken from the MQ Workflow user's or configuration profile in the currently set configuration.	80
AllocateForSystem()	Allocates the storage for the specified execution service object. The execution service to connect to is taken from the specified parameters in the currently set configuration.	80
constructor()	Constructs an execution service object.	80
Copy()	Allocates and initializes the storage for an execution service object by copying.	83
Deallocate()	Deallocates the storage for an execution service object.	84
destructor()	Destructs an execution service object.	84
Equal()	Compares two execution service objects if they represent the same session.	82
operator=()	Assigns an execution service object to this one.	82
operator==()	Compares two execution service objects if they represent the same session.	82

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
CreateProcessInstanceList()	Creates a new process instance list on the execution server.	296
CreateProcessTemplateList()	Creates a new process template list on the execution server.	303

Action methods	Description	Page
CreateWorklist()	Creates a new worklist on the execution server.	310
Logoff()	Logs off from the connected execution server.	319
Logon()	Logs on to the execution server.	321
Logon2()	Logs on to the execution server in Java and provides additional parameters.	321
persistentActivityInstanceNotification()	Retrieves the activity instance notification specified by the passed object identification in the Java API.	267
persistentProcessInstance()	Retrieves the process instance specified by the passed object identification in the Java API.	444
persistentProcessInstanceNotification()	Retrieves the process instance notification specified by the passed object identification in the Java API.	471
persistentProcessTemplate()	Retrieves the process template specified by the passed object identification in the Java API.	493
persistentWorkItem()	Retrieves the work item specified by the passed object identification in the Java API.	534
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications the logged-on user has access to.	333
QueryItems()	Retrieves the work items or notifications the logged-on user has access to.	341
QueryProcessInstanceLists()	Retrieves the process instance lists the logged-on user has access to.	347
QueryProcessInstanceNotifications()	Retrieves the process instance notifications the logged-on user has access to.	349
QueryProcessInstances()	Retrieves the process instances the logged-on user has access to.	356
QueryProcessTemplateLists()	Retrieves the process template lists the logged-on user has access to.	362
QueryProcessTemplates()	Retrieves the process templates the logged-on user has access to.	365
QueryWorkitems()	Retrieves the work items the logged-on user has access to.	370

Action methods	Description	Page
QueryWorklists()	Retrieves the worklists the logged-on user has access to.	376
Receive()	Receives execution data sent by an MQ Workflow execution server.	379
TerminateReceive()	Places information in the client input queue to indicate that receiving execution data sent by an MQ Workflow execution server can end.	384

Refer to "Activity implementation functions/methods" on page 128 for detailed descriptions of activity implementation function/methods.

Activity implementation methods	Description	Page
Passthrough()	Establishes a session between an activity implementation and an execution server.	326
RemotePassthrough()	Establishes a session between a program started by an activity implementation and an execution server.	382

Refer to "Program execution management functions/methods" on page 130 for detailed descriptions of program execution management functions/methods.

Management methods	Description	Page
PEAShutDown()	Requests to shut down the user-associated program execution agent.	329
PEAStartUp()	Starts the user-associated program execution agent.	331

Execution service array

An execution service array is an ActiveX means of holding an execution service.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array; the current configuration profile is searched for the specified system and system group.	32
AddDefault()	Adds the execution service to the array; system and system group are taken from the current configuration profile.	32
GetAt()	Returns the element at the indicated position.	33

Accessor methods	Description	Page
GetSize()	Returns the number of elements in the array.	34
RemoveAt()	Removes the element at the indicated position.	34

Events	Description	Page
ExecutionServiceRemove()	Removes the execution service from the array.	35
NewExecutionService()	Adds a new execution service to the array.	35

Exe options

An ExeOptions object represents the program implementation definitions for an executable.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an EXE options object.	80
Copy()	Allocates and initializes the storage for an EXE options object by copying.	83
Deallocate()	Deallocates the storage for an EXE options object.	84
destructor()	Destructs an EXE options object.	84
operator=()	Assigns an EXE options object to this one.	82
IsEmpty()	Indicates whether no EXE options information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
AutomaticClose()	B	States whether the window in which the EXE starts should close when the EXE ends.	91
AutomaticCloseIsNull()	B	Indicates whether automatic close is set.	121

Accessor methods	Type	Description	Page
Environment()	C	States the environment settings for the EXE.	116
EnvironmentIsNull()	B	Indicates whether an environment is set.	121
InheritEnvironment()	B	States whether the environment settings should be merged with the operating system environment settings.	91
PathAndFileName()	C	Returns the path and file name of the EXE.	116
RunInXTerm()	B	States whether the EXE should start in a separate xterm.	91
RunInXTermIsNull()	B	Indicates whether run in xterm is set.	121
StartInForeGround()	B	States whether the EXE should start in the foreground.	91
StartInForeGroundIsNull()	B	Indicates whether start in foreground is set.	121
WindowStyle()	O	States the initial window style.	119
WindowStyleIsNull()	B	Indicates whether a window style is set.	121
WorkingDirectoryName()	C	States the working directory for the EXE.	116
WorkingDirectoryNameIsNull()	B	Indicates whether a working directory is set.	121

External service options

An ExternalOptions object represents the program implementation definitions for an external service.

Refer to “Basic functions/methods” on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an External options object.	80
Copy()	Allocates and initializes the storage for an External options object by copying.	83
Deallocate()	Deallocates the storage for an External options object.	84
destructor()	Destructs an External options object.	84
operator=()	Assigns an External options object to this one.	82

Basic methods	Description	Page
IsEmpty()	Indicates whether no External options information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
BackwardMappingFormatIsNull()	B	Indicates whether a backward mapping format is set.	121
BackwardMappingParameters()	M	Returns backward mapping parameters, if any.	118
BackwardMappingParametersIsNull()	B	Indicates whether backward mapping parameters are set.	121
CodePage()	I	Specifies the code page of the service.	115
CodePageIsNull()	B	Indicates whether a code page is set.	121
ExecutableName()	C	Specifies the executable to be invoked by the invocation type and service.	116
ExecutableType()	C	Identifes the type of the executable.	116
ForwardMappingFormat()	C	Specifies the format for the mapping from an MQ Workflow container to the structure the executable uses.	116
ForwardMappingFormatIsNull()	B	Indicates whether a forward mapping format is set.	121
ForwardMappingParameters()	M	Returns forward mapping parameters, if any.	118
ForwardMappingParametersIsNull()	B	Indicates whether forward mapping parameters are set.	121

Accessor methods	Type	Description	Page
InvocationType()	C	Specifies the invocation mechanism to invoke the executable on the service.	116
IsLocalUser()	B	Returns whether a local user is to be resolved instead of using the MQ Workflow user ID.	91
IsMappingRoutineCall()	B	Specifies whether forward and backward mapping routines are to be called.	91
IsSecurityRoutineCall()	B	Specifies whether a security routine is to be called.	91
MappingType()	C	Identifies the type of mapping that should occur.	116
MappingTypeIsNull()	B	Indicates whether a mapping type is set.	121
ServiceName()	C	Identifies the service that is to be called.	116
ServiceType()	C	Identifies the type of service to be called, for example, CICS(R) or IMS(TM).	116
TimeoutPeriod()	E	Specifies how long the program execution agent should wait for a response from the started service, forever, a time period, or never.	93
TimeoutInterval()	I	Specifies the timeout interval.	115
TimeoutIntervalIsNull()	B	Indicates whether a timeout interval is set.	121

FmcError

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 functions/methods” on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an Error object.	80
Copy()	Allocates and initializes the storage for an Error object by copying.	83

Basic methods	Description	Page
Deallocate()	Deallocates the storage for an Error object.	84
destructor()	Destructs an Error object.	84
Equal()	Compares two Error objects on the basis of their return codes and parameters.	82
IsEmpty()	Indicates whether no Error information is available.	85
operator=()	Assigns an Error object to this one.	82
operator==()	Compares two Error objects on the basis of their return codes and parameters.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
Parameters()	M	Returns the parameters of the error; these are to be incorporated into the message text.	118
Rc()	I	Returns the return code remembered in the error object.	115

FmcException

An FmcException object represents a description of an exception thrown by Java.

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
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.	116
origin()	C	Returns the module that threw the exception.	116
Parameters()	M	Returns the parameters of the error; these are already incorporated into the message text.	118
Rc()	I	Returns the return code remembered in the error object.	115

Global

An API global object serves to group global MQ Workflow API functions/methods.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description
Connect()	Initializes the API in the current thread.
Disconnect()	Deinitializes the API in the current thread.

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116

Accessor methods	Type	Description	Page
SetConfigurationID()	C	Sets the configuration ID to be used for profile access. Can only be set before the first profile usage.	116

Implementation data

An implementation data object represents the program implementation definitions.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an implementation data object.	80
Copy()	Allocates and initializes the storage for an implementation data object by copying.	83
Deallocate()	Deallocates the storage for an implementation data object.	84
destructor()	Destructs an implementation data object.	84
operator=()	Assigns an implementation data object to this one.	82
IsEmpty()	Indicates whether no implementation data information is available.	85
Kind()	States the actual kind of the implementation data, whether it is a DLL or an EXE.	86

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
CommandLineParameters()	M	Returns the command line parameters to be passed to the invoked program.	118
CommandLineParametersIsNull()	B	Indicates whether command line parameters are set.	121
DllOptions()	P	Returns the description of a DLL, if the implementation is a DLL.	120

Accessor methods	Type	Description	Page
ExeOptions()	P	Returns the description of an EXE, if the implementation is an EXE.	120
ExternalOptions()	P	Returns the description of external options, if the implementation is an external service.	120
options()	P	Returns the description of an EXE, a DLL, or an external service in Java.	120
Platform()	E	Returns the operating system platform this implementation data describes.	93

Instance monitor

An instance monitor object represents a monitor in the ActiveX 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 "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	118
ControlConnectorInstances()	M	Returns the control connector instances which are represented by the instance monitor.	118

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
ObtainInstanceMonitor()	Returns the instance monitor for an activity instance of type *Block* or *Process*. The activity instance is part of the set of activity instances represented by the instance monitor.	273
Refresh()	Refreshes the instance monitor from the MQ Workflow execution server.	278

Item

An item represents a work item, an activity instance notification, or a process instance notification. This means that **all functions/methods of an item are also applicable to work items, activity instance notifications, and process instance notifications.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs an item object.	80
Copy()	Allocates and initializes the storage for an item object by copying.	83
Deallocate()	Deallocates the storage for an item object.	84
destructor()	Destructs an item object.	84
Equal()	Compares two items.	82
IsComplete()	Indicates whether the complete item information is available.	84
IsEmpty()	Indicates whether no item information is available.	85
Kind()	States the actual kind of the item, whether it is a work item or some kind of notification.	86
operator=()	Assigns an item to this one.	82
operator==()	Compares two items.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor and mutator functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific item.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method 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.	116
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	P/D	Returns the creation time of the item.	92
Description()	P/C	Returns the description of the item.	116
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the item.	116
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the ending time of the item.	92
EndTimeIsNull()	S/B	Indicates whether an end time is set.	121
Icon()	P/C	Returns the icon associated with the item.	116
InContainerName()	S/C	Returns the name of the input container of the item.	116
LastModificationTime()	P/D	Returns the last time a primary attribute of the item was changed.	92
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.	116
OutContainerName()	S/C	Returns the name of the output container of the item.	116
Owner()	P/C	Returns the user ID of the owner of the item.	116

Accessor methods	Set/ Type	Description	Page
PersistentOid()	P/C	Returns a representation of the object identification of the item.	116
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the item.	116
ProcessInstanceName()	P/C	Returns the name of the process instance the item is part of.	116
ProcessInstanceState()	P/E	Returns the state of the process instance the item is part of.	93
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	116
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the item is part of.	116
ReceivedAs()	P/E	Returns the reason why the item was received.	93
ReceivedTime()	P/D	Returns the time when the item was received.	92
StartTime()	P/D	Returns the start time of the item.	92
StartTimeIsNull()	P/B	Indicates whether a start time is set.	121

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.	124

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
Delete()	Deletes an item.	393
ObtainProcessInstanceMonitor()	Retrieves the process instance monitor for the process instance the item is part of.	395
ProcessInstance()	Retrieves the process instance the item is part of.	399
Refresh()	Retrieves the complete information of the item.	401
SetDescription()	Sets the description of the item.	403

Action methods	Description	Page
SetName()	Sets the name of the item.	406
Transfer()	Transfers an item to the specified user.	409

Item vector

An item vector represents the result of a query for items in the C-language.

Refer to "C-language vectors" on page 27 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 functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

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 function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
MessageText()	C	Returns an NLS regarding formatted message based on the message ID. Any parameters passed will be incorporated.	116

Persistent list

A persistent list represents a persistent list definition. **All functions/methods of a persistent list are also applicable toprocess instance lists, process template lists, and worklists.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
IsEmpty()	Indicates whether no persistent list information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
DescriptionIsNull()	B	Indicates whether a description is set.	121
Filter()	C	Returns the filter of the persistent list.	116
FilterIsNull()	B	Indicates whether a filter is set.	121
Name()	C	Returns the name of the persistent list.	116
OwnerOfList()	C	Returns the user ID of the owner of the persistent list.	116
OwnerOfListIsNull()	B	Indicates whether an owner is set; a public list does not have an owner.	121
SortCriteria()	C	Returns the sort criteria of the persistent list.	116
SortCriteriaIsNull()	B	Indicates whether sort criteria are set.	121
Threshold()	I	Returns the threshold of the persistent list.	115
ThresholdIsNull()	B	Indicates whether a threshold is set.	121
Type()	C	Returns the type of the persistent list, whether it is a public or private list.	116

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
Delete()	Deletes the persistent list.	413
Refresh()	Refreshes the persistent list.	416
SetDescription()	Sets the description of the persistent list.	418
SetFilter()	Sets the filter of the persistent list.	420

Action methods	Description	Page
SetSortCriteria()	Sets the sort criteria of the persistent list.	423
SetThreshold()	Sets the threshold of the persistent list.	425

Person

A person object represents the settings of the logged-on user.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a person object.	80
Copy()	Allocates and initializes the storage for a person object by copying.	83
Deallocate()	Deallocates the storage for a person object.	84
destructor()	Destructs a person object.	84
Equal()	Compares two persons.	82
operator=()	Assigns a person to this one.	82
operator==()	Compares two persons.	82
IsComplete()	Indicates whether the complete person information is available.	84
IsEmpty()	Indicates whether no person information is available.	85

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when persons are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific person.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object (O). The function/method 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.	118
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.	118
Description()	P/C	Returns the description of the person.	116
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
FirstName()	P/C	Returns the first name of the person.	116
FirstNameIsNull()	P/B	Indicates whether a first name is set.	121
IsAbsent()	P/B	Indicates whether the person is absent.	91
IsAdminForCategory()	P/B	Indicates whether the person has administrator rights for the specified category. Returns false if the category does not exist.	91
IsAdministrator()	S/B	Indicates whether the person is an administrator.	91
IsAuthorizedForAllCategories()	P/B	Indicates whether the person is said to be authorized for all categories either with basic and/or administration rights.	91

Accessor methods	Set/ Type	Description	Page
IsAuthorizedForAllCategoriesAsAdmin()	P/B	Indicates whether the person is said to be authorized for all categories as administrator.	91
IsAuthorizedForAllPersons()	P/B	Indicates whether the person is authorized to see the items of all persons.	91
IsAuthorizedForAuthorizationDefinition()	P/B	Indicates whether the person is authorized to define authorizations.	91
IsAuthorizedForOperationAdministration()	P/B	Indicates whether the person is authorized for operational administrations.	91
IsAuthorizedForProcessDefinition()	P/B	Indicates whether the person is authorized to define process models.	91
IsAuthorizedForStaffDefinition()	P/B	Indicates whether the person is authorized to define persons.	91
IsAuthorizedForTopologyDefinition()	P/B	Indicates whether the person is authorized to define topological data.	91
IsManager()	S/B	Indicates whether the person is a manager.	91
IsResetAbsence()	P/B	Indicates whether the absence flag should be reset when the person logs on.	91
LastName()	P/C	Returns the last name of the person.	116
LastNameIsNull()	P/B	Indicates whether a last name is set.	121
Level()	P/I	Returns the level of the person.	115
Manager()	S/C	Returns the user identification of the person's manager.	116
ManagerIsNull()	S/B	Indicates whether the person's manager is set.	121
MiddleName()	P/C	Returns the middle name of the person.	116

Accessor methods	Set/ Type	Description	Page
MiddleNameIsNull()	P/B	Indicates whether a middle name is set.	121
NamesOfManagedOrganizations()	S/M	Returns the names of organizations the person manages.	118
NamesOfRoles()	P/M	Returns the names of roles the person belongs to.	118
NamesOfRolesToCoordinate()	S/M	Returns the names of roles the person can coordinate.	118
OrganizationName()	P/C	Returns the name of the organization the person belongs to.	116
OrganizationNameIsNull()	P/B	Indicates whether an organization name is set.	121
PersonID()	P/C	Returns the person ID of the person.	116
PersonIDIsNull()	P/B	Indicates whether a person ID is set.	121
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.	118
PersonsAuthorizedForMe()	S/M	Returns the persons who are authorized for this person.	118
PersonsToStandInFor()	S/M	Returns the persons for whom this person stands in.	118
Phone()	P/C	Returns the phone number of the person.	116
PhoneIsNull()	P/B	Indicates whether a phone is set.	121
SecondPhone()	P/C	Returns the alternate phone number of the person.	116
SecondPhoneIsNull()	P/B	Indicates whether an alternate phone is set.	121
Substitute()	P/C	Returns the substitute of the person.	116

Accessor methods	Set/ Type	Description	Page
SubstituteIsNull()	P/B	Indicates whether a substitute is set.	121
SystemName()	P/C	Returns the home system of the person.	116
UserID()	P/C	Returns the user identification of the person.	116

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
Refresh()	Retrieves the complete person information from the server.	429
SetAbsence()	Sets the absent flag of the logged-on user to the specified value.	431
SetSubstitute()	Sets the substitute of the logged-on user to the specified value.	433

Point

A point object represents a bend point of a control connector.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a point object.	80
Copy()	Allocates and initializes the storage for a point object by copying.	83
Deallocate()	Deallocates the storage for a point object.	84
destructor()	Destructs a point object.	84
Equal()	Compares two point objects on the basis of their contents.	82
IsEmpty()	Indicates whether no point information is available.	85
operator=()	Assigns a point object to this one.	82
operator==()	Compares two point objects on the basis of their contents.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	115
YPosition()	I	Returns the y-coordinate of the point.	115

Point array

A point array represents the result of a query for bend points in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Point vector

A point vector represents the result of a query for points in the C-language.

Refer to "C-language vectors" on page 27 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.

Process instance

A process instance object represents an instance of a workflow process template.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a process instance object.	80
Copy()	Allocates and initializes the storage for a process instance object by copying.	83
Deallocate()	Deallocates the storage for a process instance object.	84
destructor()	Destructs a process instance object.	84
Equal()	Compares two process instances.	82
IsComplete()	Indicates whether the complete process instance information is available.	84
IsEmpty()	Indicates whether no process instance information is available.	85
operator=()	Assigns a process instance to this one.	82
operator==()	Compares two process instances.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

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.

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 function/method 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.	93
Category()	P/C	Returns the category of the process instance.	116
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	S/D	Returns the creation time of the process instance.	92
Creator()	S/C	Returns the creator of the process instance.	116
Description()	P/C	Returns the description of the process instance.	116

Accessor methods	Set/ Type	Description	Page
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the process instance.	116
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the end time of the process instance.	92
EndTimeIsNull()	S/B	Indicates whether an end time is set.	121
Icon()	P/C	Returns the icon associated with the process instance.	116
InContainerName()	S/C	Returns the name of the input container of the process instance.	116
InContainerNeeded()	P/B	Indicates whether an input container is needed to start the process instance.	91
LastModificationTime()	P/D	Returns the last time a primary attribute of the process instance was changed.	92
LastStateChangeTime()	P/D	Returns the last time the state of the process instance was changed.	92
Name()	P/C	Returns the name of the process instance.	116
NotificationTime()	S/D	Returns the notification time of the process instance.	92
NotificationTimeIsNull()	S/B	Indicates whether a notification time is set.	121
NotifiedPerson()	S/C	Returns the person who received the notification.	116
NotifiedPersonIsNull()	S/B	Indicates whether a notified person is set.	121
OrganizationName()	S/C	Returns the name of the organization of the process instance.	116
OrganizationNameIsNull()	S/B	Indicates whether an organization name is set.	121
OutContainerName()	S/C	Returns the name of the output container of the process instance.	116

Accessor methods	Set/ Type	Description	Page
ParentName()	P/C	Returns the name of the parent process instance of this process instance.	116
ParentNameIsNull()	P/B	Indicates whether a parent name is set.	121
PersistentOid()	P/C	Returns a representation of the object identification of the process instance.	116
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the process instance.	116
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	121
ProcessTemplateName()	P/C	Returns the name of the process template the process instance is derived from.	116
RoleName()	S/C	Returns the name of the role of the process instance.	116
RoleNameIsNull()	S/B	Indicates whether a role is set.	121
Starter()	S/C	Returns the starter of the process instance.	116
StarterIsNull()	S/B	Indicates whether a starter is set.	121
StartTime()	S/D	Returns the start time of the process instance.	92
StartTimeIsNull()	S/B	Indicates whether a start time is set.	121
State()	P/E	Returns the state of the process instance.	93
StateOfNotification()	S/E	Returns the notification state of the process instance.	93
SuspensionExpirationTime()	P/D	Returns the suspension expiration time of the process instance.	92
SuspensionExpirationTimeIsNull()	P/B	Indicates whether the suspension expiration time is set.	121
SuspensionTime()	P/D	Returns the time the process instance was suspended.	92
SuspensionTimeIsNull()	P/B	Indicates whether the suspension time is set.	121

Accessor methods	Set/ Type	Description	Page
SystemGroupName()	P/C	Returns the name of the system group where the process instance runs.	116
SystemName()	P/C	Returns the name of the system where the process instance runs.	116
TopLevelName()	P/C	Returns the name of the top level process instance of this process instance.	116

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
Delete()	Deletes the process instance.	437
InContainer()	Retrieves the input container of the process instance.	440
ObtainMonitor()	Retrieves the process instance monitor for the process instance.	442
PersistentObject()	Retrieves the process instance specified by the passed object identification.	444
Refresh()	Retrieves the complete information of the process instance.	447
Restart()	Restarts the process instance.	449
Resume()	Resumes the execution of a suspended process instance.	451
SetDescription()	Sets the description of the process instance.	453
SetName()	Sets the name of the process instance.	456
Start()	Starts the process instance.	458
Start2()	Starts the process instance in Java and provides an input container.	458
Suspend()	Suspends the process instance.	460
Suspend2()	Suspends the process instance in Java until the specified calendar date.	460
SuspendUntil()	Suspends the process instance until the specified time.	460
Terminate()	Terminates the process instance.	463

Process instance list

A process instance list represents a group of process instances. **All functions/methods of a persistent list are also applicable to process instance lists.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a process instance list object.	80
Copy()	Allocates and initializes the storage for a process instance list object by copying.	83
Deallocate()	Deallocates the storage for a process instance list object.	84
destructor()	Destructs a process instance list object.	84
Equal()	Compares two process instance lists.	82
operator=()	Assigns a process instance list to this one.	82
operator==()	Compares two process instance lists.	82

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
QueryProcessInstances()	Retrieves the process instances qualifying via the process instance list.	467

Process instance list array

A process instance list array represents the result of a query for process instance lists in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Events	Description	Page
NewProcessInstanceList()	Adds a new process instance list to the array.	35
ProcessInstanceListRemove()	Removes the process instance list from the array.	35

Process instance list vector

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 27 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.

Process instance monitor

A process instance monitor object represents a monitor of a process instance. **All functions/methods of FmcjBlockInstanceMonitor are also applicable to process instance monitors.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
Deallocate()	Deallocates the storage for a process instance monitor object. All block instance monitors contained are also deallocated.	84
destructor()	Destructs a process instance monitor object. All block instance monitors contained are also destructed.	84

Process instance notification

A process instance notification represents a notification raised for a process instance. **All functions/methods of an FmcjItem are also applicable to process instance notifications.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a process instance notification object.	80

Basic methods	Description	Page
Copy()	Allocates and initializes the storage for a process instance notification object by copying.	83
Deallocate()	Deallocates the storage for a process instance notification object.	84
destructor()	Destructs a process instance notification object.	84
Kind()	In the C++ language, states that the object is a process instance notification.	86
operator=()	Assigns a process instance notification to this one.	82
operator==()	Compares two process instance notifications.	82

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	395
PersistentObject()	Retrieves the specified process instance notification.	471

Process instance notification array

A process instance notification array represents the result of a query for process instance notifications in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Process instance notification vector

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 27 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.

Process instance vector

A process instance vector represents the result of a query for process instances in the C-language.

Refer to "C-language vectors" on page 27 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.

Process template

A process template object represents the Runtime equivalent of a Buildtime workflow process model.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a process template object.	80
Copy()	Allocates and initializes the storage for a process template object by copying.	83
Deallocate()	Deallocates the storage for a process template object.	84
destructor()	Destructs a process template object.	84

Basic methods	Description	Page
Equal()	Compares two process templates.	82
IsComplete()	Indicates whether the complete process template information is available.	84
IsEmpty()	Indicates whether no process template information is available.	85
operator=()	Assigns a process template to this one.	82
operator==()	Compares two process templates.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process templates are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific process template.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), a multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method 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.	93
Category()	P/C	Returns the category of the process template.	116
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	P/D	Returns the creation time of the process template.	92
Description()	P/C	Returns the description of the process template.	116
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the process template.	116
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
Icon()	P/C	Returns the icon associated with the process template.	116
InContainerName()	S/C	Returns the name of the input container of the process template.	116

Accessor methods	Set/ Type	Description	Page
InContainerNeeded()	P/B	Indicates whether an input container is needed to start an instance of the process template.	91
LastModificationTime()	P/D	Returns the last time a primary attribute of the process template was changed.	92
Name()	P/C	Returns the name of the process template.	116
OrganizationName()	S/C	Returns the name of the organization of the process template.	116
OrganizationNameIsNull()	S/B	Indicates whether an organization name is set.	121
OutContainerName()	S/C	Returns the name of the output container of the process template.	116
PersistentOid()	P/C	Returns a representation of the object identification of the process template.	116
ProcessAdmin()	S/C	Returns the user ID of the process administrator of an instance of the process template.	116
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	121
RoleName()	S/C	Returns the name of the role of the process template.	116
RoleNameIsNull()	S/B	Indicates whether a role is set.	121
ValidFromTime()	P/D	Returns the time when the process template becomes valid.	92

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
CreateAndStartInstance()	Creates and starts an instance of the process template.	475
CreateAndStartInstance2()	Creates and starts an instance of the process template in Java and provides an input container.	475
CreateInstance()	Creates an instance of the process template.	480
Delete()	Deletes the specified process template.	483
Delete2()	Deletes the specified process template versions in Java.	483

Action methods	Description	Page
ExecuteProcessInstance()	Creates and executes an instance from the specified process template.	486
ExecuteProcessInstanceAsync()	Creates and executes an instance from the specified process template; an answer is not waited for.	486
InContainer()	Retrieves the input container of the process template.	491
PersistentObject()	Retrieves the process template specified by the passed object identification.	493
Refresh()	Retrieves the complete information of the process template.	495

Process template list

A process template list represents a group of process templates. **All functions/methods of a persistent list are also applicable to process template lists.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a process template list object.	80
Copy()	Allocates and initializes the storage for a process template list object by copying.	83
Deallocate()	Deallocates the storage for a process template list object.	84
Equal()	Compares two process template lists.	82
destructor()	Destructs a process template list object.	84
operator=()	Assigns a process template list to this one.	82
operator==()	Compares two process template lists.	82

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
QueryProcessTemplates()	Retrieves the process templates qualifying via the process template list.	499

Process template list array

A process template list array represents the result of a query for process template lists in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Events	Description	Page
NewProcessTemplateList()	Adds a new process template list to the array.	35
ProcessTemplateListRemove()	Removes the specified process template list from the array.	35

Process template list vector

A process template list vector represents the result of a query for process template lists in the C-language.

Refer to "C-language vectors" on page 27 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.

Process template vector

A process template vector represents the result of a query for process templates in the C-language.

Refer to "C-language vectors" on page 27 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.

Program data

A program data object represents the program implementation definitions.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a program data object.	80
Copy()	Allocates and initializes the storage for a program data object by copying.	83
Deallocate()	Deallocates the storage for a program data object.	84
destructor()	Destructs a program data object.	84
Equal()	Compares two program data objects if they belong to the same work item.	82
IsEmpty()	Indicates whether no program data information is available yet.	85
operator=()	Assigns a program data object to this one.	82
operator==()	Compares two program data objects if they belong to the same work item.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
Description()	C	Returns the description of the implementing program.	116
DescriptionIsNull()	B	Indicates whether a description is set.	121
Icon()	C	Returns the icon associated with the implementing program.	116
Implementations()	M	Returns the implementation definitions of the program.	118
InContainer()	P	Returns the input container of the program.	120
IsUnattended()	B	States whether the program can run unattended.	91
OutContainer()	P	Returns the output container of the program.	120

ReadOnly container

A read-only container represents an input data container of a work item. **All functions/methods of a container are applicable to read-only containers.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a read-only container object.	80
Copy()	Allocates and initializes the storage for a read-only container object by copying.	83
Deallocate()	Deallocates the storage for a read-only container object.	84
Equal()	Compares two read-only containers.	82
destructor()	Destructs a read-only container object.	84
operator=()	Assigns a read-only container to this one.	82
operator==()	Compares two read-only containers.	82

ReadWrite container

A read/write container represents an input container of a process instance or an output container of a work item. **All functions/methods of a container are applicable to read/write containers.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a read/write container object.	80
Copy()	Allocates and initializes the storage for a read/write container object by copying.	83
Deallocate()	Deallocates the storage for a read/write container object.	84
Equal()	Compares two read/write containers.	82
destructor()	Destructs a read/write container object.	84
operator=()	Assigns a read/write container to another one.	82
operator==()	Compares two read/write containers.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

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 function/method 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.	64
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.	64
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.	64
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.	64
SetBinaryValue()	N	Sets the value of the specified container leaf element in the C-language. The leaf element is of type BINARY.	64
SetBuffer()	N	Sets the value of the specified container leaf element in Java. The leaf element is of type BINARY.	64
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.	64
SetDouble()	F	Sets the value of the specified container leaf element in Java. The leaf element is of type FLOAT.	64

Accessor methods	Type	Description	Page
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.	64
SetFloatValue()	F	Sets the value of the specified container leaf element in the C-language. The leaf element is of type FLOAT.	64
SetLong()	I	Sets the value of the specified container leaf element in Java. The leaf element is of type LONG.	64
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.	64
SetLongValue()	I	Sets the value of the specified container leaf element in the C-language. The leaf element is of type LONG.	64
SetString()	N	Sets the value of the specified container leaf element in Java. The leaf element is of type STRING.	64
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.	64
SetStringValue()	C	Sets the value of the specified container leaf element in the C-language. The leaf element is of type STRING.	64
SetValue()	N/F/C/I	Sets the value of the specified container leaf element in the C++ language.	64

Result object

A result object represents the result of a function/method call in the C++ and C-language.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
destructor	Destructs the C++ representation of the result object.	84

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	116
ObjectOfCurrentThread()	P	Returns the result object associated with the thread from where this function/method is called.	
Origin()	C	Returns the origin of the result, that is, file, line, function.	116
Parameters()	M	Returns the parameters of the result; these are already incorporated in the message text.	118
Rc()	I	Returns the return code remembered in the result object.	115

Service

A service object represents common aspects of MQ Workflow service objects. **All functions/methods of a service are also applicable to execution services.**

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method declaration can be found in a general format at the indicated page.

Accessor methods	Type	Description	Page
IsLoggedOn()	B	Indicates whether a user is logged on via this service object.	91
SetTimeout()	I	Sets the time the client will wait for a server to answer.	122
SystemGroupName()	C	Returns the name of the system group where the server resides.	116

Accessor methods	Type	Description	Page
SystemName()	C	Returns the name of the system where the server resides.	116
Timeout()	I	Returns the time the client will wait for a server to answer.	115
UserID()	C	Returns the user identification of the logged-on user.	116

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
Refresh()	Refreshes information from the server, especially the logged-on status.	503
SetPassword()	Sets the password of the logged-on user.	505
UserSettings()	Retrieves the user settings of the logged-on user.	507

String array

A string array represents a list of strings in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
Add()	Adds the element to the array.	32
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34
RemoveAll()	Removes all elements.	34
RemoveAt()	Removes the element at the indicated position.	34
SetAt()	Sets the element at the indicated position.	34

String vector

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 27 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.
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.

Symbol layout

A symbol layout object represents graphical information of a named icon.

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a symbol layout object.	80
Copy()	Allocates and initializes the storage for a symbol layout object by copying.	83
Deallocate()	Deallocates the storage for a symbol layout object.	84
destructor()	Destructs a symbol layout object.	84
Equal()	Compares two symbol layout objects on the basis of their contents.	82
IsEmpty()	Indicates whether no symbol layout information is available.	85
operator=()	Assigns a symbol layout object to this one.	82
operator==()	Compares two symbol layout objects on the basis of their contents.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	115
XPositionOfName()	I	Returns the x-coordinate of the name associated to the icon.	115
YPosition()	I	Returns the y-coordinate of the named icon.	115
YPositionOfName()	I	Returns the y-coordinate of the name associated to the icon.	115

Work item

A work item represents an activity instance assigned to a user in order to be worked on. **All functions/methods of an Item are also applicable to work items.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a work item object.	80

Basic methods	Description	Page
Copy()	Allocates and initializes the storage for a work item object by copying.	83
Deallocate()	Deallocates the storage for a work item object.	84
destructor()	Destructs a work item object.	84
Kind()	In the C++ language, states that the object is a work item.	86
operator=()	Assigns a work item to this one.	82
operator==()	Compares two work items.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when work items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific work item.

The value in the **Type** column states the type of the property returned, whether it is a boolean (B), a character string (C), a date/time value (D), an enumeration (E), an integer (I), multi-valued property (M), a pointer to some object (P), or an object itself (O). The function/method 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.	93
ErrorReason()	S/O	Returns an error object describing the reason why the associated activity instance is in state InError.	119
ErrorReasonIsNull()	S/B	Indicates whether an error reason is set.	121
ExitCondition()	S/C	Returns the exit condition of the work item.	116
FirstNotificationTime()	S/D	Returns the time the first notification for the work item is to occur or has occurred.	92
FirstNotificationTimeIsNull()	S/B	Indicates whether a first notification time is set.	121

Accessor methods	Set/ Type	Description	Page
Implementation()	P/C	Returns the name of the implementing program of the associated activity instance.	116
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	121
ManualExitMode()	S/B	Returns whether the exit mode of the work item is manual.	91
ManualStartMode()	S/B	Returns whether the start mode of the work item is manual.	91
Priority()	P/I	Returns the priority of the work item.	115
SecondNotificationTime()	S/D	Returns the time the second notification for the work item is to occur or has occurred.	92
SecondNotificationTimeIsNull()	S/B	Indicates whether a second notification time is set.	121
Staff()	S/M	Returns all persons owning a work item for the associated activity instance.	118
StartCondition()	S/C	Returns the start condition of the work item.	116
State	P/E	Returns the state of the work item.	93
StateOfNotification()	S/E	Returns the notification state of the work item.	93
SupportTools()	P/M	Returns the support tools associated with the work item.	118
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	121

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
CancelCheckOut()	Cancels the check out of the work item.	514
CheckIn()	Checks in the work item.	516
CheckOut()	Checks out the work item.	518
Finish()	Finishes a manual exit work item.	524
ForceFinish()	Force finishes the work item.	526
ForceRestart()	Force restarts the work item.	528

Action methods	Description	Page
InContainer()	Retrieves the input container of the work item.	530
ObtainInstanceMonitor()	Returns the instance monitor for the associated process instance in ActiveX.	395
OutContainer()	Retrieves the output container of the work item.	532
PersistentObject()	Retrieves the specified work item.	534
Restart()	Restarts the work item.	536
Start()	Starts the work item.	538
StartTool()	Starts the specified support tool.	540
Terminate()	Terminates the work item.	542

Work item array

A work item array represents the result of a query for work items in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34

Work item vector

A workitem vector represents the result of a query for work items in the C-language.

Refer to "C-language vectors" on page 27 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 functions/methods of a persistent list are also applicable to worklists.**

Refer to "Basic functions/methods" on page 79 for detailed descriptions of basic functions/methods.

Basic methods	Description	Page
constructor()	Constructs a worklist object.	80
Copy()	Allocates and initializes the storage for a worklist object by copying.	83
Deallocate()	Deallocates the storage for a worklist object.	84
destructor()	Destructs a worklist object.	84
Equal()	Compares two worklists.	82
operator=()	Assigns a worklist to another one.	82
operator==()	Compares two worklists.	82

Refer to "Accessor functions/methods" on page 89 for detailed descriptions of accessor functions/methods. 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 function/method 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.	91

Refer to "Action functions/methods" on page 128 for detailed descriptions of action functions/methods.

Action methods	Description	Page
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications qualifying via the worklist.	545
QueryItems()	Retrieves all items qualifying via the worklist.	548
QueryProcessInstanceNotifications()	Retrieves the process instance notifications qualifying via the worklist.	551
QueryWorkitems()	Retrieves the work items qualifying via the worklist.	554

Work list array

A worklist array represents the result of a query for worklists in ActiveX.

Refer to "ActiveX arrays" on page 32 for detailed descriptions of array accessor methods.

Accessor methods	Description	Page
GetAt()	Returns the element at the indicated position.	33
GetSize()	Returns the number of elements in the array.	34
Events	**Description**	**Page**
NewWorklist()	Adds a new worklist to the array.	35
WorklistRemove()	Removes the specified worklist from the array.	35

Worklist vector

A worklist vector represents the result of a query for worklists in the C-language.

Refer to "C-language vectors" on page 27 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 function/method 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 6. Programming interfaces

The following chapters describe the MQ Workflow application programming interfaces for action or activity implementation functions/methods in alphabetical order.

Each entry contains a functional description of the API function/method 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
States the name of the file to be included respectively the name of the package to be imported for the API function/method declaration.

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 29. 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 "Activity instance" on page 187 for a complete list of functions/methods.

ObtainProcessInstanceMonitor()/ ObtainInstanceMonitor

This function/method retrieves the process instance 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 block instance monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes

- See "Action functions/methods" on page 128 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 FmcjActivityInstanceObtainProcessInstanceMonitor(
       FmcjActivityInstanceHandle            hdlInstance,
       bool                                  deep,
       FmcjProcessInstanceMonitorHandle *    monitor)
```

C++ language signature

```
APIRET ObtainProcessInstanceMonitor(
       FmcjProcessInstanceMonitor &          monitor,
       bool                                  deep= false ) const
```

Java signature

```
public abstract
ProcessInstanceMonitor obtainProcessInstanceMonitor( 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 process instance monitor is to be retrieved.

monitor Input/Output. The address of the handle to the process instance monitor respectively the process 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*/ProcessInstanceMonitor*/ ProcessInstanceMonitor
A pointer to the process instance monitor respectively the process instance monitor.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The activity instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SubProcessInstance()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)

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

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

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

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 30. 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 "Activity instance notification" on page 191 for a complete list of functions/methods.

PersistentObject()

This function/method retrieves the activity instance notification identified by the passed object identifier from the MQ Workflow execution server (action call).

The MQ Workflow execution server from which the activity instance notification is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information (primary and secondary) of the activity instance notification.

In C++, when the activity instance notification object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the activity instance notification handle already points to some object. In Java, an activity instance notification is newly created; the execution service acts as a factory.

Usage notes

- See "Action functions/methods" on page 128 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.ExecutionService

ActiveX signature

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationPersistentObject(
       FmcjExecutionServiceHandle                    service,
       char const *                                  oid,
       FmcjActivityInstanceNotificationHandle *      hdlItem )
```

C++ language signature

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java signature

```
public abstract
  ActivityInstanceNotification
  ExecutionService.persistentActivityInstanceNotification( String oid )
throws FmcException
```

Parameters

hdlItem Input/Output. The address of the handle to the activity instance notification object to be set.

oid Input. The object identifier of the activity instance notification to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

ActivityInstanceNotification
The activity instance notification retrieved.

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The activity instance notification does no longer exist.

FMC_ERROR_INVALID_OID(805)
The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

StartTool()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 31. Block instance monitor actions

An FmcjBlockInstanceMonitor or a BlockInstanceMonitor object represents a monitor for an activity instance of type *Block*.

Note: The ownership of a block instance monitor stays with the embracing process instance monitor. A block instance monitor is automatically deleted when the process instance monitor is deleted. After that action, using the block instance monitor handle or object is invalid.

An FmcjBlockInstanceMonitor or a BlockInstanceMonitor object represents the common aspects of monitors. In the C++ language, FmcjBlockInstanceMonitor is thus the superclass of the FmcjProcessInstanceMonitor class and provides for all common properties and methods. In the Java language, BlockInstanceMonitor is thus a superclass of the ProcessInstanceMonitor class and provides for all common properties and methods. Similarly, in the C-language, common implementations of functions are taken from FmcjBlockInstanceMonitor. That is, common functions start with the prefix FmcjBlockInstanceMonitor; they are also defined starting with the prefix FmcjProcessInstanceMonitor. In ActiveX, inheritance is not supported. All methods are defined on class InstanceMonitor - see "Chapter 34. Instance monitor actions" on page 387.

The following sections describe the actions which can be applied on a block instance monitor. See "Block instance monitor" on page 197 for a complete list of functions/methods.

ObtainBlockInstanceMonitor()

This function/method retrieves the block instance monitor for the specified activity instance from the MQ Workflow execution server (action call).

The specified activity instance must be of type *Block* and be part of this block instance monitor.

Usage notes

- See "Action functions/methods" on page 128 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 Not applicable - see "Chapter 34. Instance monitor actions" on page 387.

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.BlockInstanceMonitor

C-language signature

```
FmcjBlockInstanceMonitorHandle
    FMC_APIENTRY FmcjBlockInstanceMonitorObtainBlockInstanceMonitor(
       FmcjBlockInstanceMonitorHandle  hdlMonitor,
       FmcjActivityInstanceHandle      activity )
```

C++ language signature

```
FmcjBlockInstanceMonitor *
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity ) const

APIRET
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity,
                            FmcjBlockInstanceMonitor &   monitor ) const
```

Java signature

```
public abstract
    BlockInstanceMonitor obtainBlockInstanceMonitor(
       ActivityInstance activity ) throws FmcException
```

Parameters

activity Input. The activity instance of type Block whose block instance monitor is to be retrieved.

hdlMonitor Input. The block instance monitor containing the activity instance of type Block.

monitor Input/Output. The block instance monitor retrieved.

Return type

APIRET The result of calling this function/method - see return codes below.

FmcjBlockInstanceMonitor*/ Handle/ BlockInstanceMonitor
The block instance monitor respectively a pointer or handle to the block instance monitor.

APIRET or the MQ Workflow **result object** can return the following codes respectively the following FmcExceptions can be thrown:

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The specified activity instance is not described by the block instance monitor.

FMC_ERROR_WRONG_KIND(501)
The specified activity instance is not of type Block.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

ObtainProcessInstanceMonitor()

This function/method retrieves the process instance monitor for the specified activity instance from the MQ Workflow execution server (action call).

The specified activity instance must be of type *Process* and be part of this block instance monitor.

When the deep option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also fetched from the server.

Note: Deep is currently not supported.

Usage notes

- See "Action functions/methods" on page 128 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	Not applicable - see "Chapter 34. Instance monitor actions" on page 387.
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.BlockInstanceMonitor

C-language signature

```
FmcjProcessInstanceMonitorHandle
    FMC_APIENTRY FmcjBlockInstanceMonitorObtainProcessInstanceMonitor(
        FmcjBlockInstanceMonitorHandle  hdlMonitor,
        FmcjActivityInstanceHandle      activity,
        bool                            deep      )
```

C++ language signature

```
FmcjProcessInstanceMonitor *
    ObtainProcessInstanceMonitor(
       FmcjActivityInstance const &     activity,
       bool                             deep= false ) const

APIRET ObtainProcessInstanceMonitor(
       FmcjActivityInstance const &     activity,
       FmcjProcessInstanceMonitor &     monitor,
       bool                             deep= false   ) const
```

Java signature

```
public abstract
  ProcessInstanceMonitor
  obtainProcessBlockInstanceMonitor( ActivityInstance activity,
                                     boolean          deep    )
throws FmcException
```

Parameters

activity Input. The activity instance of type Process whose process instance monitor is to be retrieved.

deep Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is currently ignored.

hdlMonitor Input. The block instance monitor containing the activity instance of type Process.

monitor Output. The process instance monitor retrieved.

Return type

APIRET The result of calling this function/method - see return codes below.

FmcjProcessInstanceMonitor*/ Handle/ ProcessInstanceMonitor
The process instance monitor respectively a pointer or handle to the process instance monitor.

APIRET or the MQ Workflow **result object** can return the following codes respectively the following FmcExceptions can be thrown:

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The specified activity instance is not described by the block instance monitor.

FMC_ERROR_WRONG_KIND(501)
The specified activity instance is not of type Process.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

Refresh()

This function/method refreshes the block instance monitor from the MQ Workflow execution server (action call).

All information about the block instance monitor is retrieved.

When the deep option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also refreshed from the server.

Note: Deep is currently not supported.

Usage notes

- See "Action functions/methods" on page 128 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 Not applicable - see "Chapter 34. Instance monitor actions" on page 387.

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.BlockInstanceMonitor

C-language signature

```
APIRET FMC_APIENTRY FmcjBlockInstanceMonitorRefresh(
       FmcjBlockInstanceMonitorHandle 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 monitor is also to be provided. Note, deep is currently ignored.

hdlMonitor Input. The handle of the block instance monitor to be refreshed.

Return type

APIRET The result of calling this function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0)	The function/method completed successfully.
FMC_ERROR(1)	A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)	The object has not yet been read from the database, that is, does not yet represent a persistent one.
FMC_ERROR_INVALID_HANDLE(130)	The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)	The process instance does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)	Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)	An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)	Timeout has occurred.

Chapter 32. Container activity implementation functions/methods

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 functions/methods 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 198 for a complete list of functions/methods on containers.

InContainer()

This function/method 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 notes

- See “Activity implementation functions/methods” on page 128 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 function/method - see return codes below.

ReadOnlyContainer
The input container of the activity implementation or support tool.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an input container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 677
- For a C++ example see "Programming an executable (C++)" on page 678

OutContainer()

This function/method retrieves the output container from the MQ Workflow program execution agent (activity implementation call).

It can be used from within an activity implementation.

Usage notes

- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

ReadWriteContainer
The output container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 677
- For a C++ example see "Programming an executable (C++)" on page 678

RemoteInContainer()

This function/method 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 notes

- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

ReadOnlyContainer
The input container of the activity implementation or support tool.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)
The program identification is invalid.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an input container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 function/method 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 notes

- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

ReadWriteContainer
The output container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_PROGRAMID(135)
The program identification is invalid.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 function/method 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 notes

- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 677
- For a C++ example see "Programming an executable (C++)" on page 678

SetRemoteOutContainer()

This function/method 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 notes
- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0)	The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_PROGRAM_EXECUTION(126)
The function/method 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 33. 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 functions/methods 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 "Execution service" on page 212 for a complete list of functions/methods.

CreateProcessInstanceList()

This function/method 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.

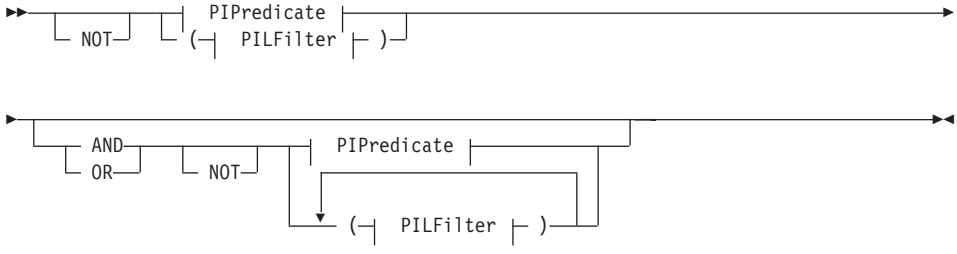
A process instance list filter is specified as a character string containing a filter on process instances (refer to "Appendix A. How to read the syntax diagrams" on page 781).

Notes:

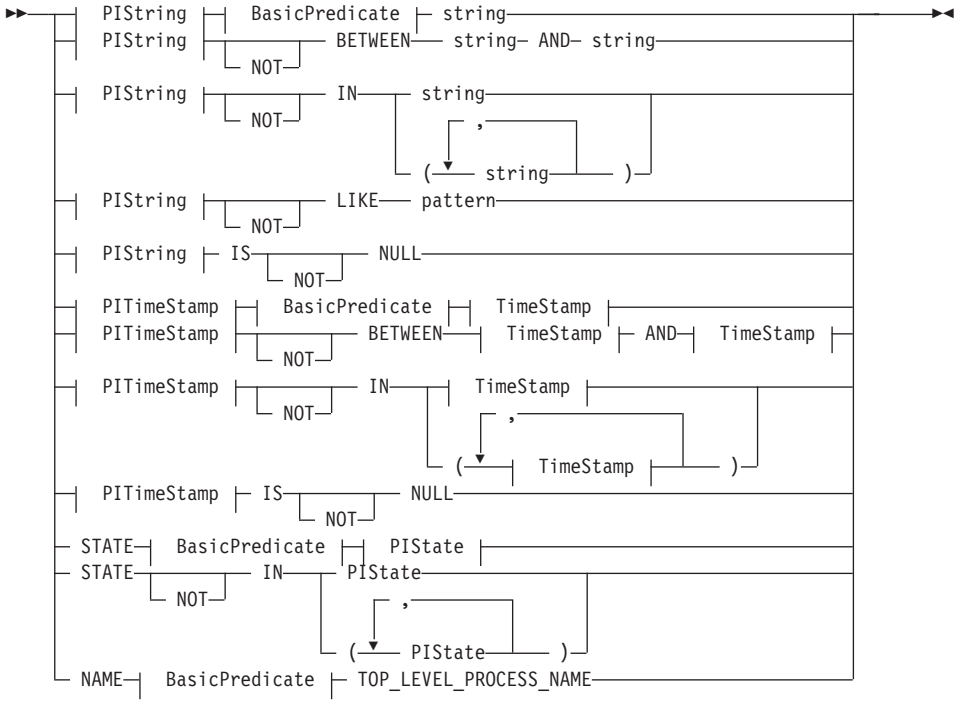
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

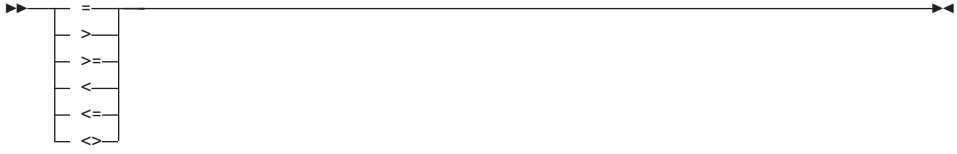
PILFilter



PIPredicate



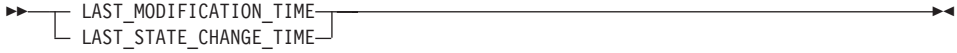
BasicPredicate



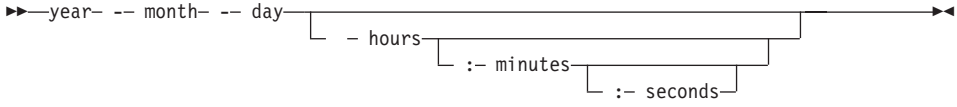
PIState

PIString

PITimeStamp



TimeStamp

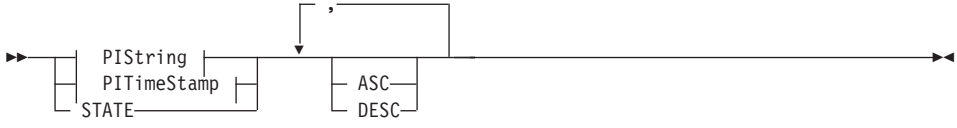


A process instance list sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PIState diagram.

PILOrderBy



Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessInstanceList
The newly created process instance list.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Create a process instance list (ActiveX)" on page 643.
- For a C-language example see "Create a process instance list (C-language)" on page 644.
- For a C++ example see "Create a process instance list (C++)" on page 646.
- For a Java example see "Create a process instance list (Java)" on page 647.

CreateProcessTemplateList()

This function/method 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.

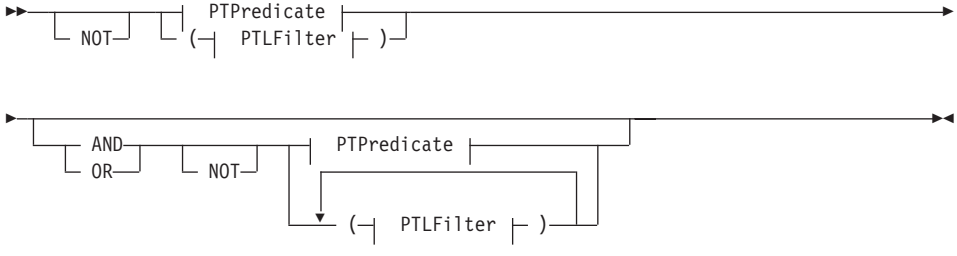
A process template list filter is specified as a character string containing a filter on process templates (refer to "Appendix A. How to read the syntax diagrams" on page 781).

Notes:

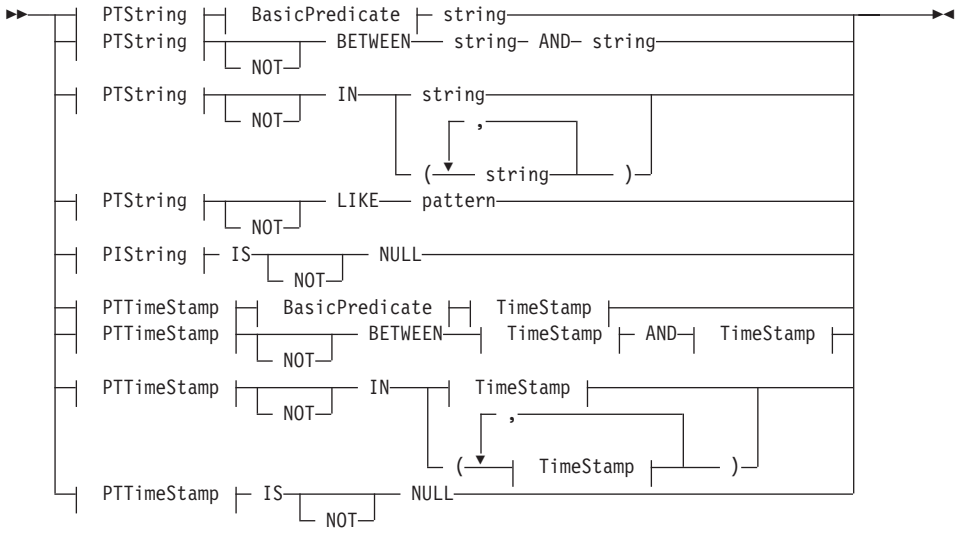
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

PTLFilter



PTPredicate

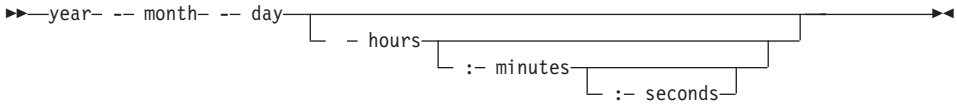


BasicPredicate

PTString

PTTimeStamp

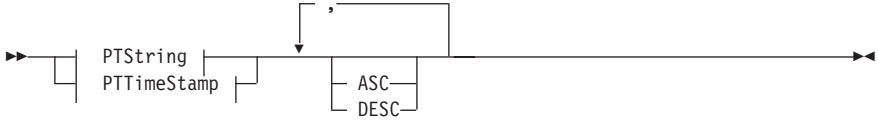
TimeStamp



A process template list sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTLOrderBy



Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessTemplateList
The newly created process template list.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Create a process instance list (ActiveX)” on page 643.
- For a C-language example see “Create a process instance list (C-language)” on page 644.
- For a C++ example see “Create a process instance list (C++)” on page 646.
- For a Java example see “Create a process instance list (Java)” on page 647.

CreateWorklist()

This function/method 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.

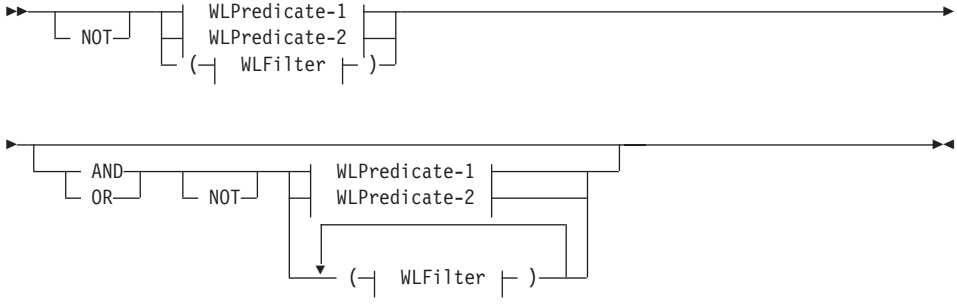
A worklist filter is specified as a character string containing a filter on the items in the worklist (refer to "Appendix A. How to read the syntax diagrams" on page 781).

Notes:

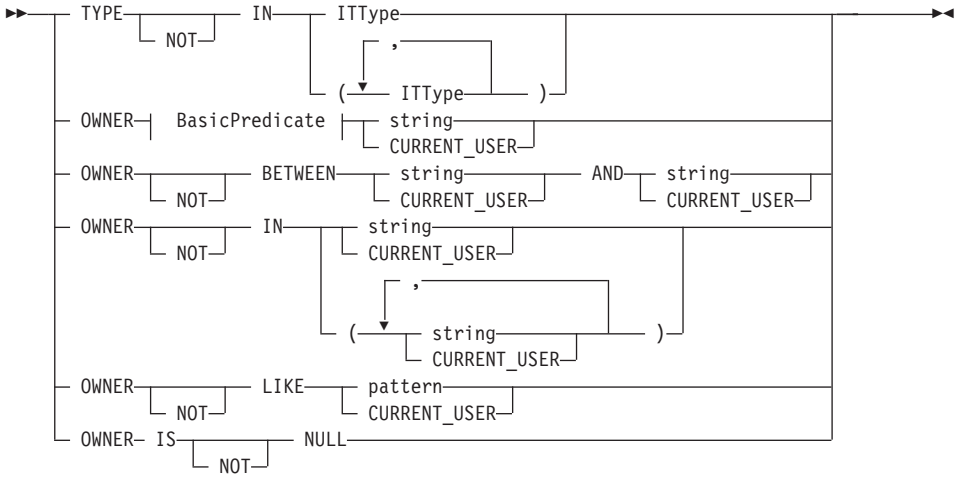
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

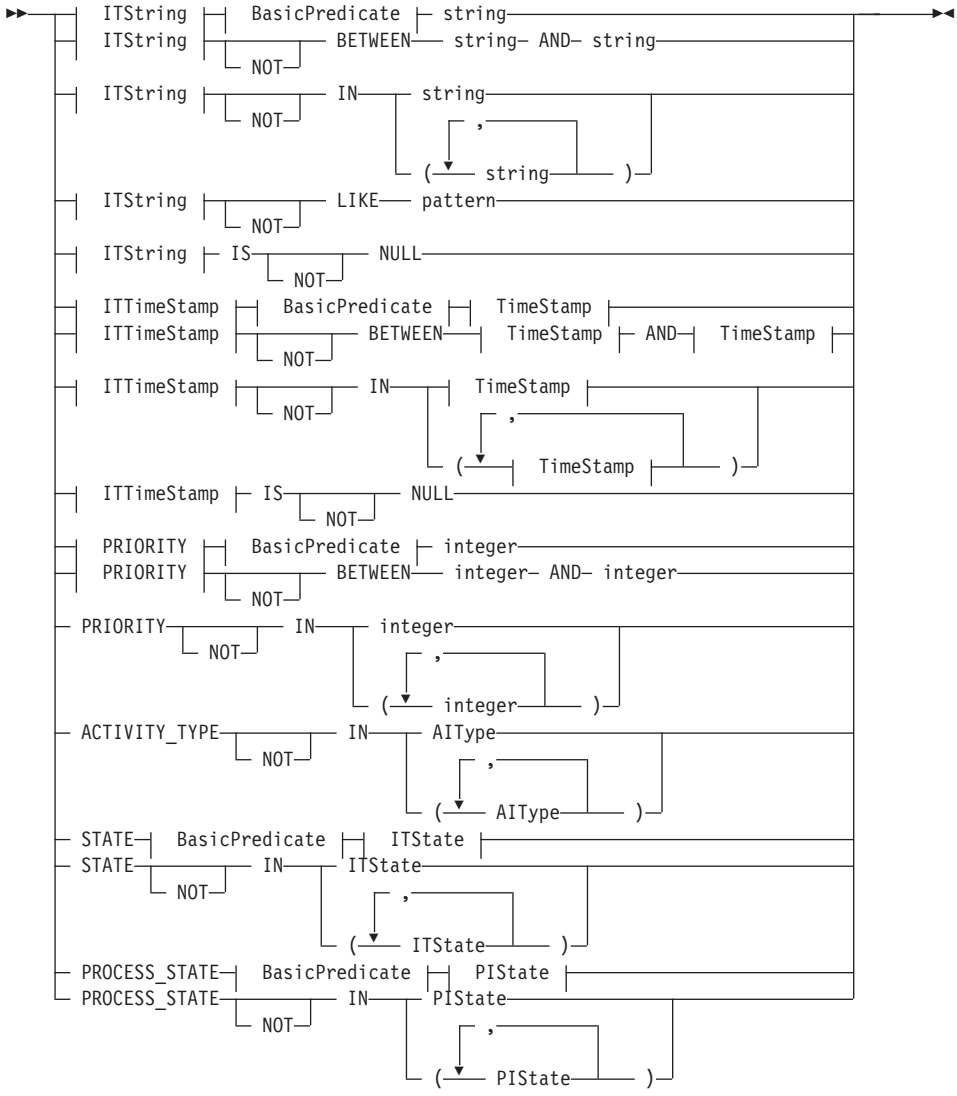
WLFilter



WLPredicate-1



WLPredicate-2



AIType

BasicPredicate

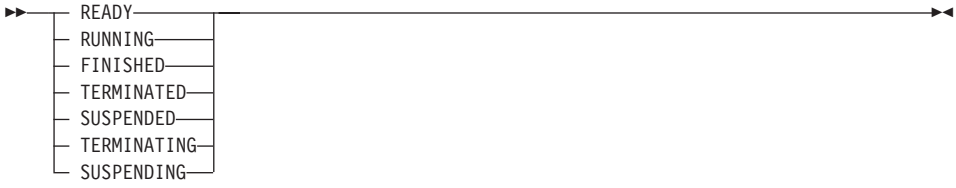
ITState

ITString

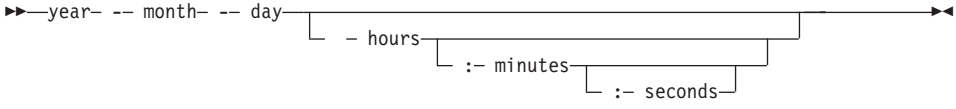
ITTimeStamp

ITType

PIState



TimeStamp



A worklist sort criterion is specified as a character string.

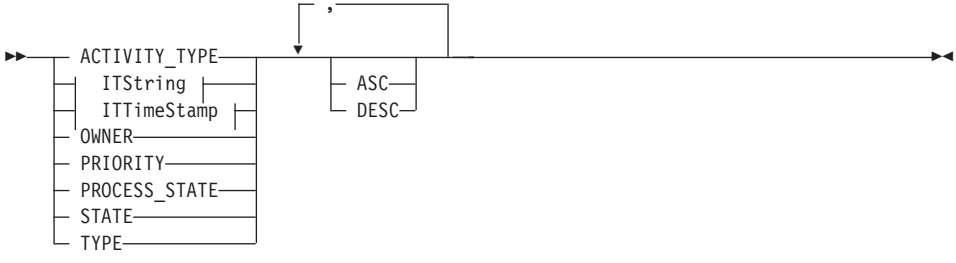
Note: The default sort order is ascending.

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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

WorkList
The newly created worklist.

Return codes/ FmcException

FMC_OK(0)
The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Create a process instance list (ActiveX)" on page 643.
- For a C-language example see "Create a process instance list (C-language)" on page 644.
- For a C++ example see "Create a process instance list (C++)" on page 646.
- For a Java example see "Create a process instance list (Java)" on page 647.

Logoff()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Part 8. Examples and scenarios" on page 637.

Logon()

This function/method 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 functions/methods of the MQ Workflow API.

The user ID to log on with must be a registered MQ Workflow user.

When the execution server supports *unified logon,* an empty password and user ID can be provided. The user ID to log on with is then retrieved from the operating system, that is, logon must have been performed at the client. The client is trusted and the execution server performs no password checking.

After a successful logon, the execution service object represents that single user session. A further request to log on with a different user ID will be rejected. You can, however, establish as many sessions as needed, even for the same user, using different execution service objects, one for each session.

At logon time, you can specify your mode of operation. When you are operating in 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 unsolicited messages pushed by the execution server - see "The push data access model" on page 18 for additional prerequisites.

There can only be a single present session for one user. The *present here* option can be used, to force that other present session logoff and to newly establish a present session here.

When you are operating in a *default* session mode, the execution server does not assume that you are able to react. Activity instances are not automatically started on your behalf and messages are not pushed to you. There can be multiple sessions for one user with the *default* session mode.

The following enumeration types can be used to specify the session mode:

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

See logon; allows for the sepcification of the session mode and absence setting.

Usage notes

- See "Action functions/methods" on page 128 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 )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogon (
       FmcjExecutionServiceHandle        service,
       char const *                      userID,
       char const *                      password,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator )
```

C++ language signature

```
APIRET Logon( string const & userID, string const & password )

APIRET Logon(
       string const &                   userID,
       string const &                   password,
       SessionMode                      sessionMode = Present,
       AbsenceIndicator                 absenceIndicator = NotSet )
```

Java signature

```
public abstract
void   logon ( String userID,   String password )

public abstract
void   logon2( String           userID,
               String           password,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator ) 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.

userID
Input. The user ID of the user on whose behalf a logon is to be made. Can be empty for unified logon.

Return type

long/ APIRET
The return code of calling this function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0)
The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_ALREADY_LOGGED_ON(11)
The user is already logged on with present mode or the execution service object already represents a different user session.

FMC_ERROR_BACK_LEVEL_VERSION(504)
The version of the client is out-of-date, that is, not supported by this server.

FMC_ERROR_INVALID_ABSENCE_SPEC(905)
An unknown absence setting has been specified.

FMC_ERROR_INVALID_SESSION_MODE(901)
An unknown session mode has been specified.

FMC_ERROR_NEWER_VERSION(505)
The version of the client is newer than the server version, that is, not supported.

FMC_ERROR_PASSWORD(12)
Incorrect password.

FMC_ERROR_PROFILE(124)
Required user or workstation profile entries cannot be found.

FMC_ERROR_USERID_UNKNOWN(10)
No user ID registered with 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Part 8. Examples and scenarios" on page 637.

Passthrough()

This function/method 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 notes

- See "Activity implementation functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Programming an executable (C-language)” on page 677.
- For a C++ example see “Programming an executable (C++)” on page 678.

PEAShutDown()

This function/method allows to shutdown the program execution agent associated to the logged on user (program execution management function/method 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 notes

- See "Program execution management functions/methods" on page 130 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

PEAStartUp()

This function/method 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 notes

- See "Program execution management functions/methods" on page 130 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

QueryActivityInstanceNotifications()

This function/method 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 function/method. 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.

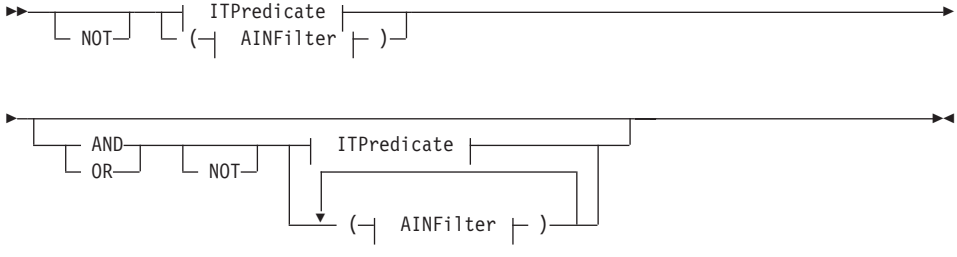
The activity instance notifications to be retrieved can be characterized by a filter. An activity instance notification filter is specified as a character string:

Notes:

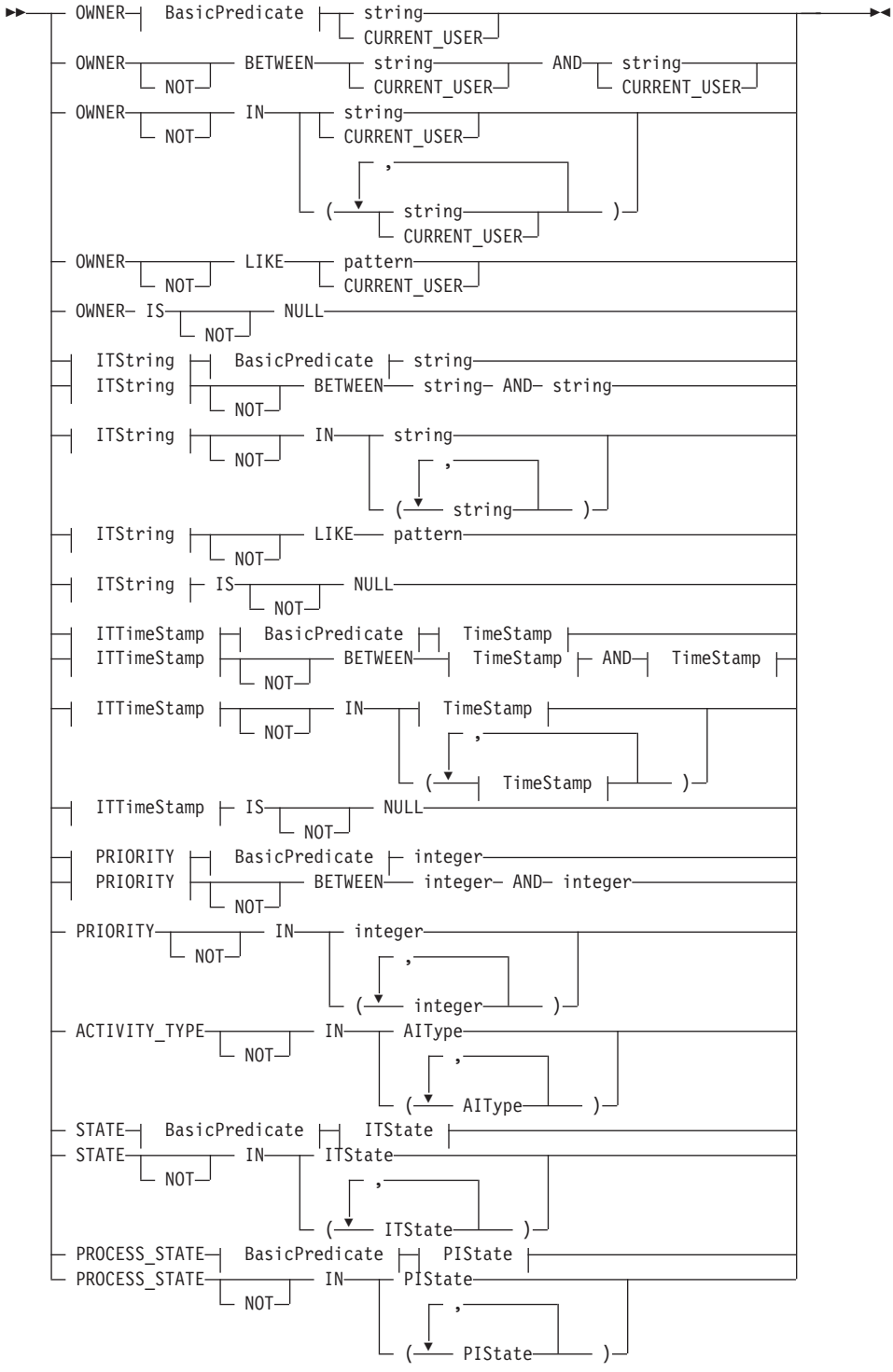
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

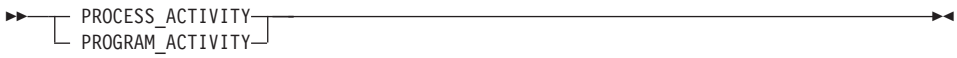
AINFilter



ITPredicate


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

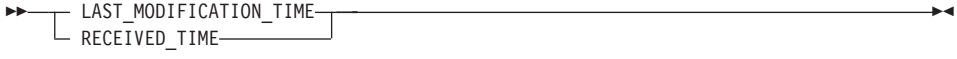


BasicPredicate

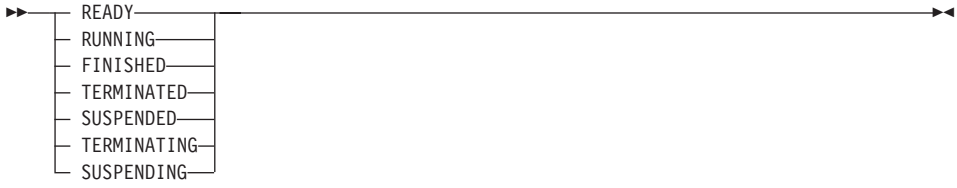
ITState

ITString

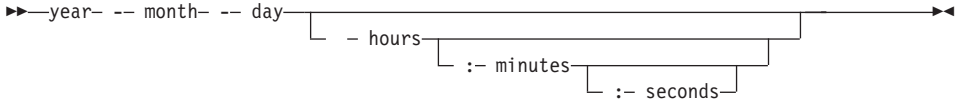
ITTimeStamp



PIState



TimeStamp

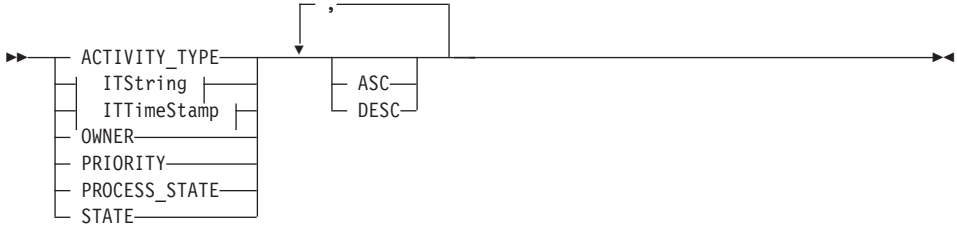


Activity instance notifications can be sorted. An activity instance notification sort criterion is specified as a character string.

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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ActivityInstanceNotification[]
The qualifying activity instance notifications.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 663.
- For a C++ example see "Query process instances (C++)" on page 664.
- For a Java example see "Query process instances (Java)" on page 665.

QueryItems()

This function/method 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 function/method. 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.

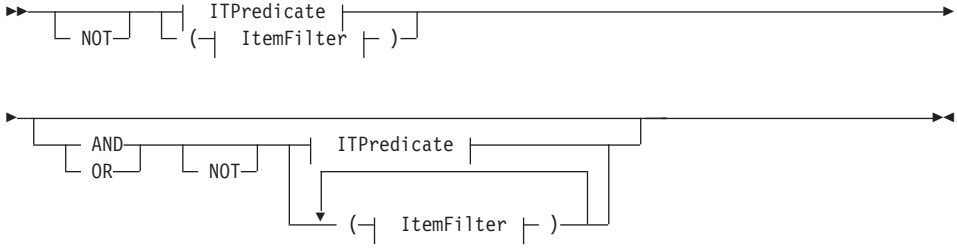
The items to be retrieved can be characterized by a filter. An item filter is specified as a character string.

Notes:

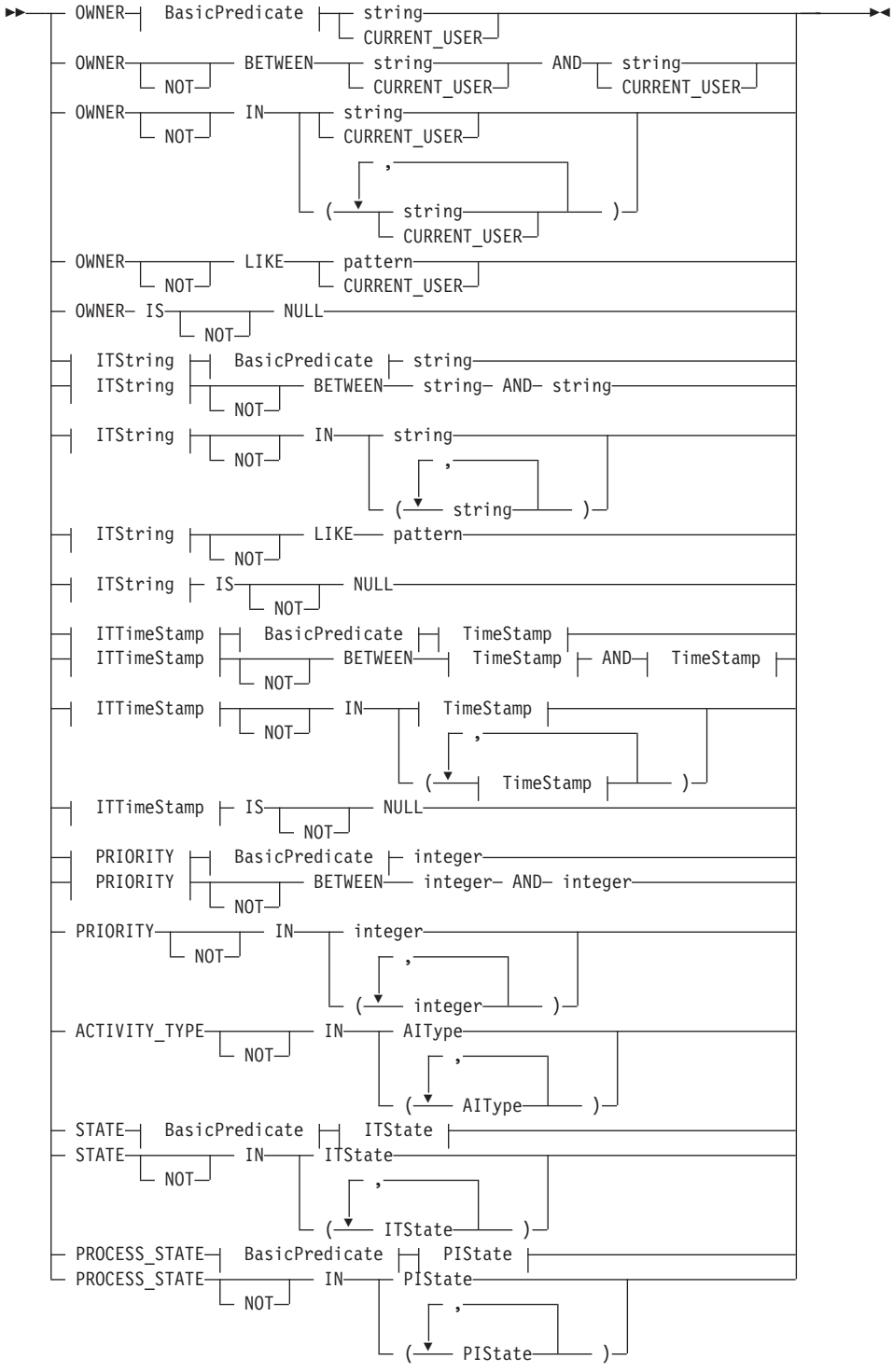
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

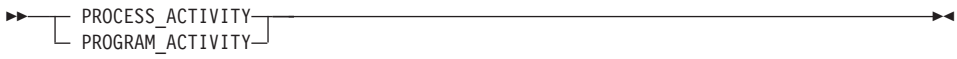
ItemFilter



ITPredicate


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

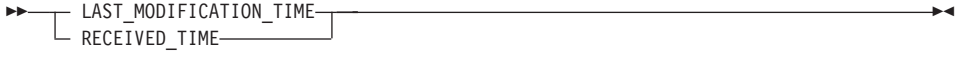


BasicPredicate

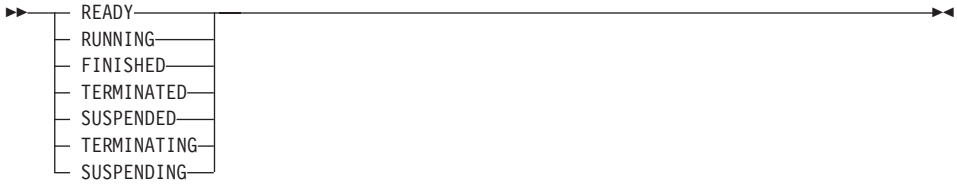
ITState

ITString

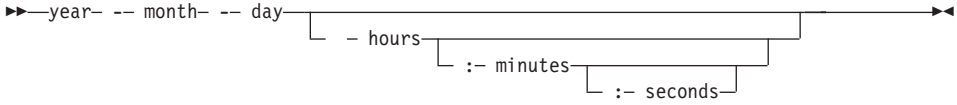
ITTimeStamp



PIState



TimeStamp



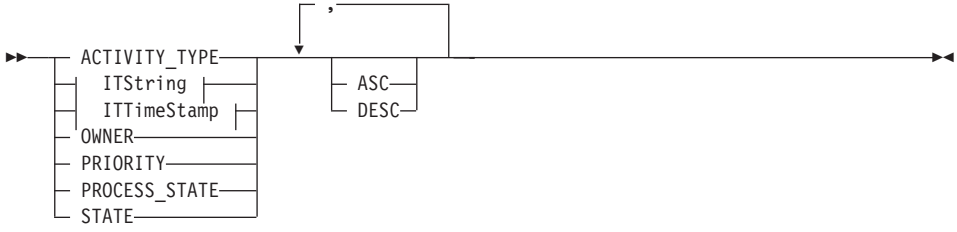
Items can be sorted. An item sort criterion is specified as a character string.

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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Item[] The qualifying items.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 663.
- For a C++ example see "Query process instances (C++)" on page 664.
- For a Java example see "Query process instances (Java)" on page 665.

QueryProcessInstanceLists()

This function/method 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 function/method. 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessInstanceList[]
The qualifying process instance lists.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query worklists (ActiveX)” on page 652.
- For a C-language example see “Query worklists (C-language)” on page 653.
- For a C++ example see “Query worklists (C++)” on page 655.
- For a Java example see “Query worklists (Java)” on page 657.

QueryProcessInstanceNotifications()

This function/method 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 function/method. 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.

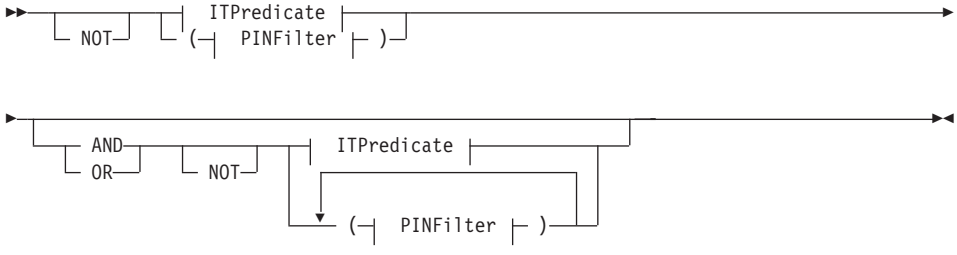
The process instance notifications to be retrieved can be characterized by a filter. A process instance notification filter is specified as a character string.

Notes:

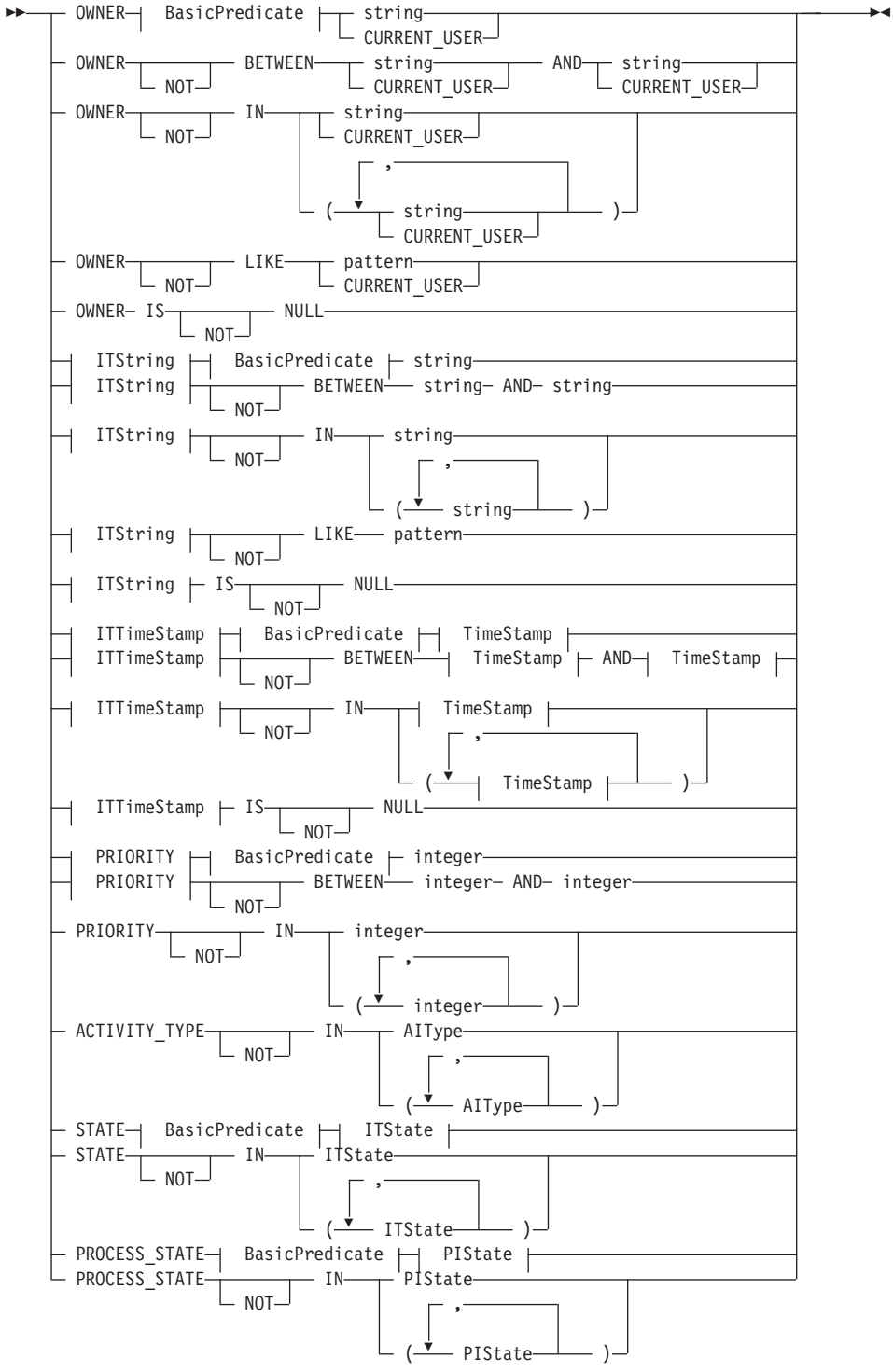
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

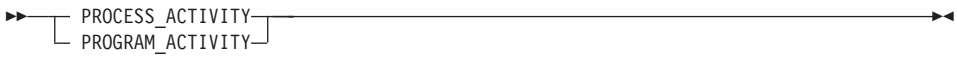
PINFilter



ITPredicate


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

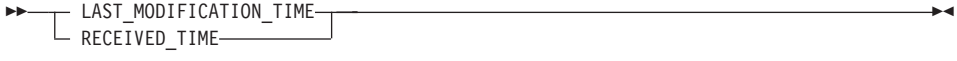


BasicPredicate

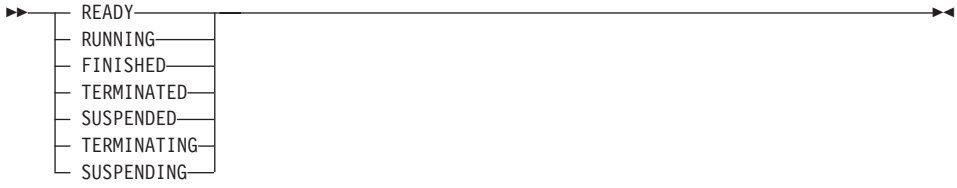
ITState

ITString

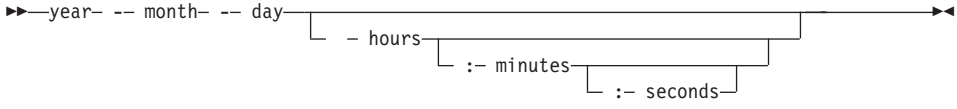
ITTimeStamp



PIState



TimeStamp

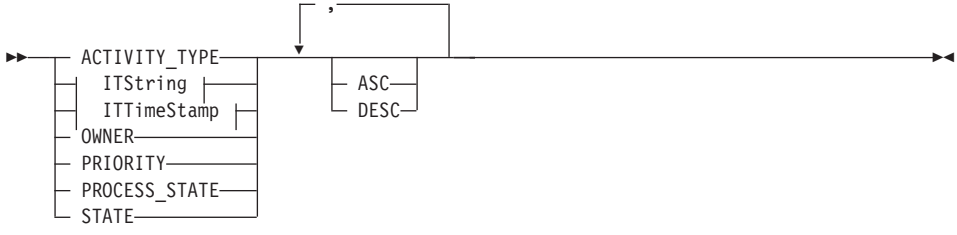


Process instance notifications can be sorted. A process instance notification sort criterion is specified as a character string.

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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessInstanceNotification[]
The qualifying process instance notifications.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 663.
- For a C++ example see "Query process instances (C++)" on page 664.
- For a Java example see "Query process instances (Java)" on page 665.

QueryProcessInstances()

This function/method 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 function/method. 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.

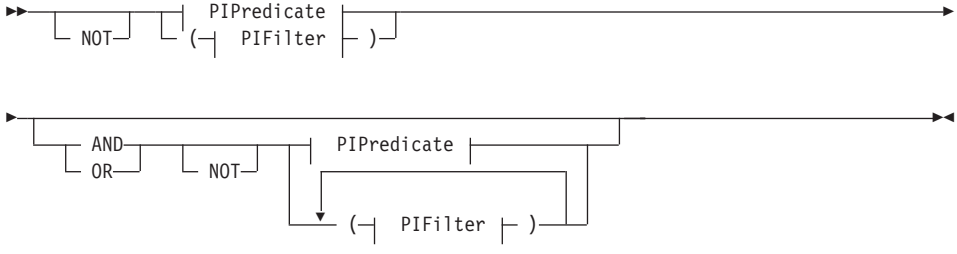
A filter on process instances is specified as a character string containing a filter predicate:

Notes:

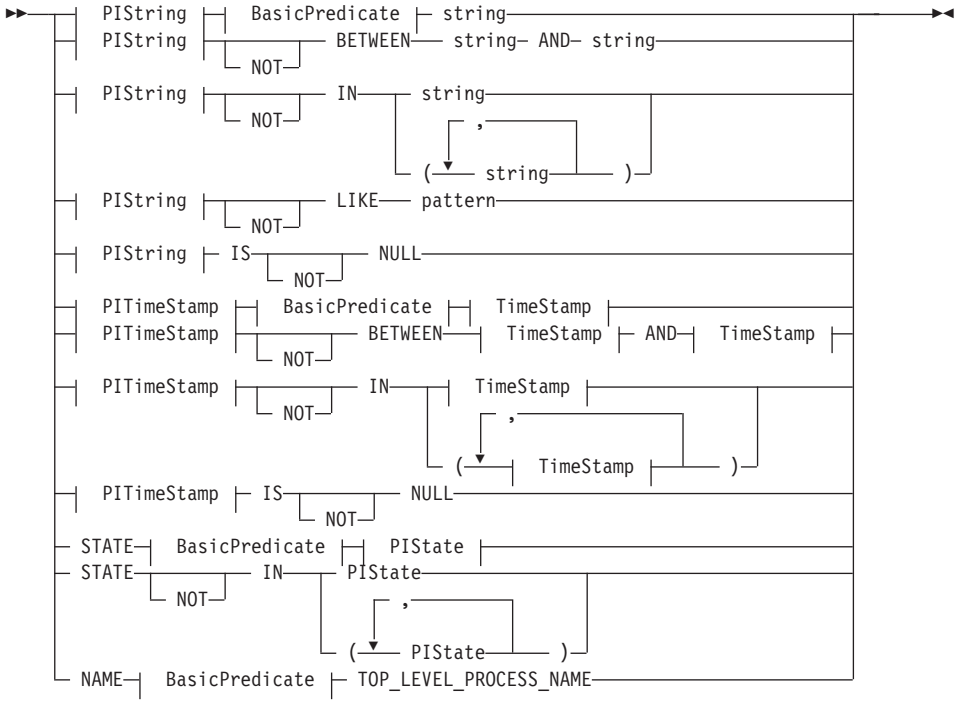
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

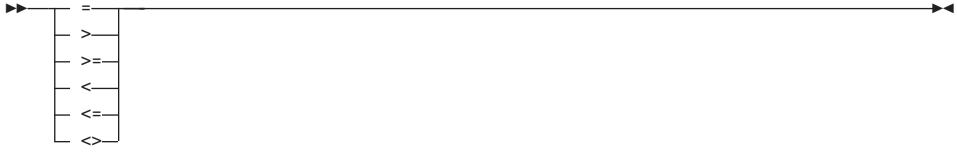
PIFilter



PIPredicate



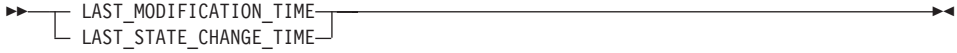
BasicPredicate



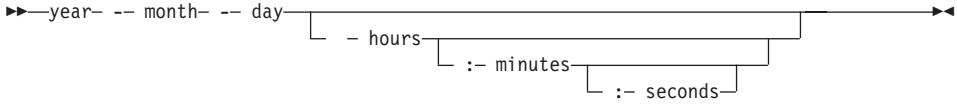
PIState

PIString

PITimeStamp



TimeStamp

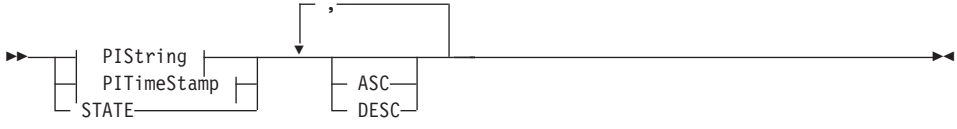


Process instances can be sorted. A process instance sort criterion is specified as a character string.

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
- StartTime
- State
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessInstance[]
The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 663.
- For a C++ example see "Query process instances (C++)" on page 664.
- For a Java example see "Query process instances (Java)" on page 665.

QueryProcessTemplateLists()

This function/method 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 function/method. 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessTemplateList[]
The qualifying process template lists.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query worklists (ActiveX)” on page 652.
- For a C-language example see “Query worklists (C-language)” on page 653.
- For a C++ example see “Query worklists (C++)” on page 655.
- For a Java example see “Query worklists (Java)” on page 657.

QueryProcessTemplates()

This function/method 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 function/method. 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.

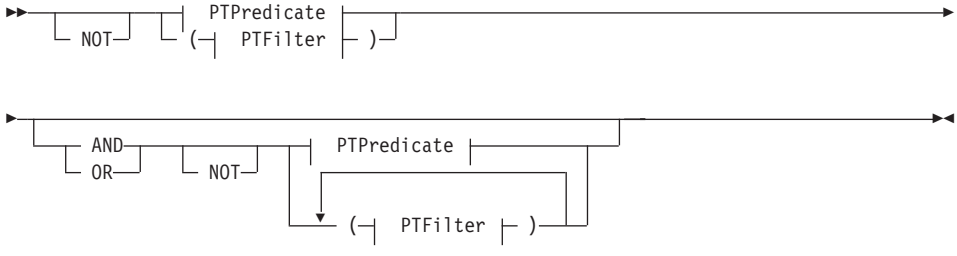
A filter on process templates is specified as a character string containing a filter predicate:

Notes:

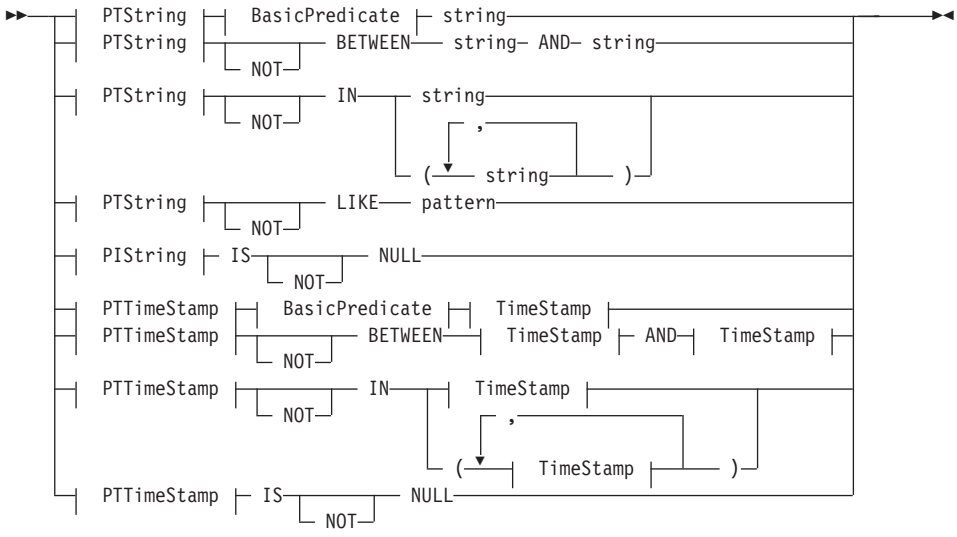
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

PTFilter



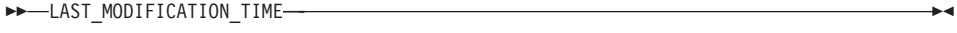
PTPredicate



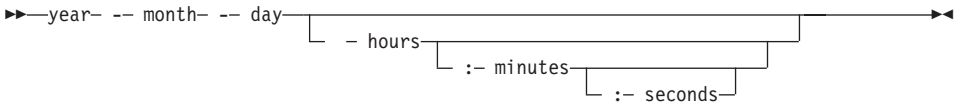
BasicPredicate

PTString

PTTimeStamp



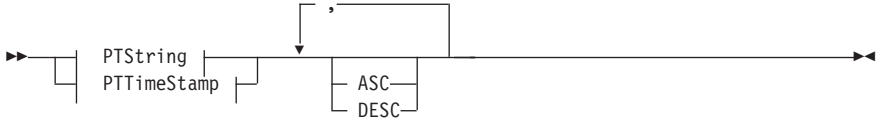
TimeStamp



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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessTemplate[]
The qualifying process templates.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 663.
- For a C++ example see “Query process instances (C++)” on page 664.

- For a Java example see "Query process instances (Java)" on page 665.

QueryWorkitems()

This function/method 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 function/method. 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.

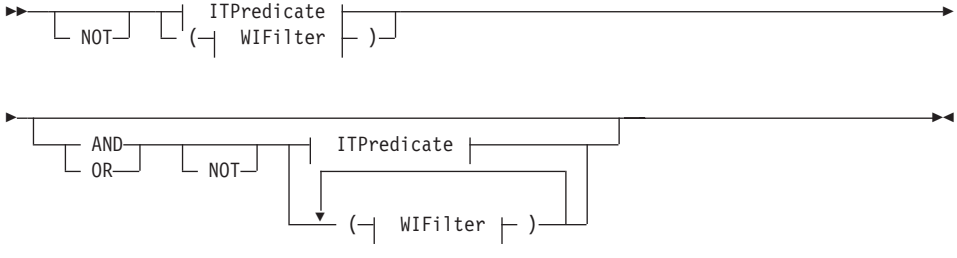
The work items to be retrieved can be characterized by a filter. A work item filter is specified as a character string:

Notes:

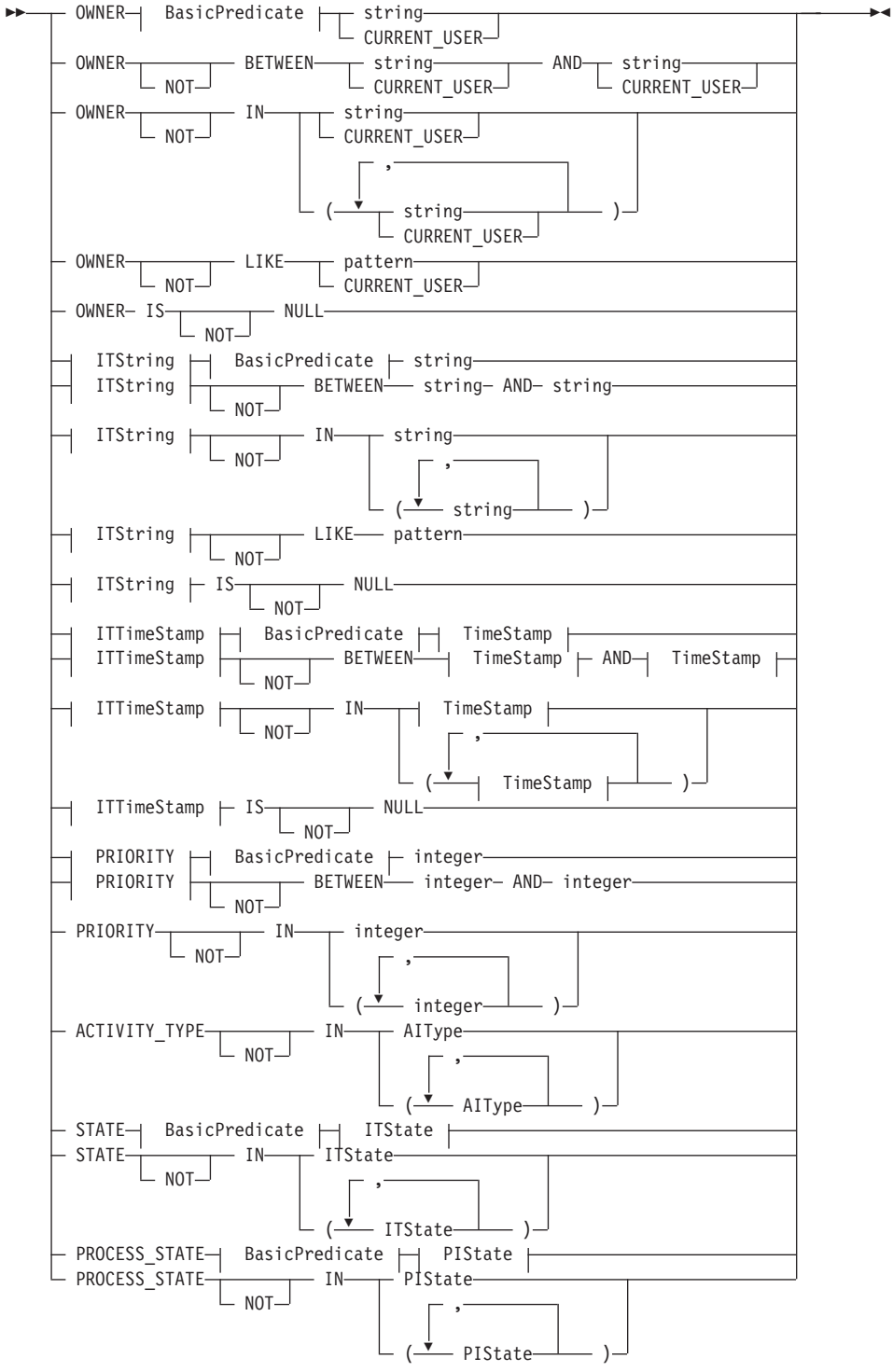
1. A *string* constant is to be enclosed in single quotes (').

   A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks.
2. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

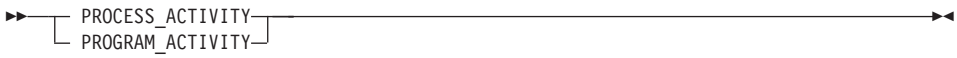
WIFilter



ITPredicate


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

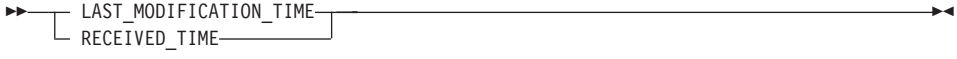


BasicPredicate

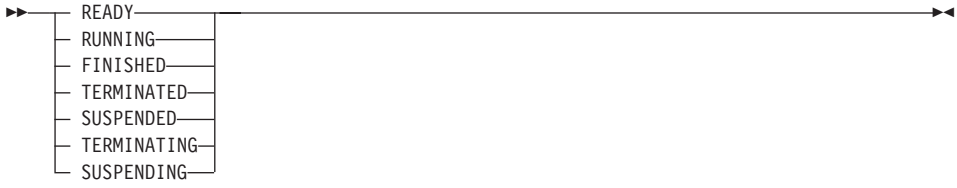
ITState

ITString

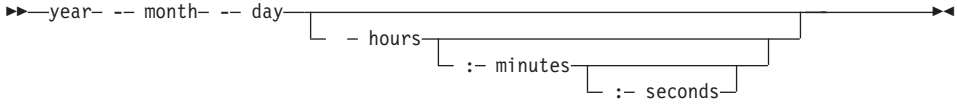
ITTimeStamp



PIState



TimeStamp



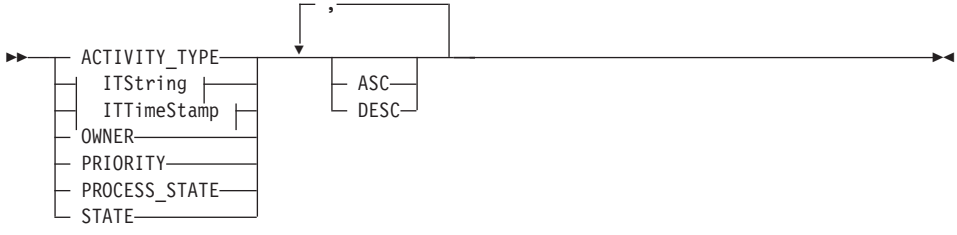
Work items can be sorted. A work item sort criterion is specified as a character string.

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 notes
- See "Action functions/methods" on page 128 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,
       FmcjWorkitemHandle *        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 function/method - see return codes below.

WorkItem[] The qualifying work items.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 663.
- For a C++ example see “Query process instances (C++)” on page 664.
- For a Java example see “Query process instances (Java)” on page 665.

QueryWorklists()

This function/method 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 function/method. 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

WorkList[] The qualifying worklists.

Return codes/ FmcException

FMC_OK(0) The function/method 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_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query worklists (ActiveX)" on page 652.
- For a C-language example see "Query worklists (C-language)" on page 653.
- For a C++ example see "Query worklists (C++)" on page 655.
- For a Java example see "Query worklists (Java)" on page 657.

Receive()

This function/method 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. 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 "Execution data" on page 210 for functions/methods 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

Error(1) Indicates that the execution data describes an error returned as the response on an asynchronous request.

C-language	Fmc_DART_Error

C++ FmcjExecutionData::Error

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(1001)
Indicates that a work item has been created or updated.

C-language Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

ActivityInstanceNotification(1002)
Indicates that an activity instance notification has been created or updated.

C-language Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

ProcessInstanceNotification(1003)
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

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

RemotePassthrough()

This function/method 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 that 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 notes
- See “Activity implementation functions/methods” on page 128 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	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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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 within 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.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

TerminateReceive()

This function/method 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, then a correlation ID is returned which can be used to explicitly receive this data.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes

FMC_OK(0)	The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 34. 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* in the ActiveX API.

The following sections describe the actions which can be applied on an instance monitor. See "Instance monitor" on page 222 for a complete list of functions/methods.

ObtainInstanceMonitor()

This function/method retrieves the instance monitor for the specified activity instance from the MQ Workflow execution server (action call).

The specified activity instance must be of type *Block* or of type *Process* and be part of this instance monitor.

Usage notes

- See "Action functions/methods" on page 128 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	not applicable - see "Chapter 31. Block instance monitor actions" on page 273

C++ not applicable - see "Chapter 31. Block instance monitor actions" on page 273

JAVA not applicable - see "Chapter 31. Block instance monitor actions" on page 273

ActiveX signature

```
InstanceMonitor *
ObtainInstanceMonitor( long *             returnCode,
                       ActivityInstance * activity,
                       boolean            deep    )
```

Parameters

activity Input. The activity instance whose instance monitor is to be retrieved.

deep Input. An indicator whether monitors of activity instances of type *Block* are also to be retrieved. Note that deep is currently not supported.

returnCode Input/Output. The result of calling this function/method - see return codes below.

Return type

InstanceMonitor*

The instance monitor retrieved.

Return codes

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)

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

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

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

FMC_ERROR_DOES_NOT_EXIST(118)

The specified activity instance is not described by the instance monitor or does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 not applicable - see "Chapter 31. Block instance monitor actions" on page 273

C++ not applicable - see "Chapter 31. Block instance monitor actions" on page 273

JAVA not applicable - see "Chapter 31. Block instance monitor actions" on page 273

C++ language signature

```
long Refresh( boolean deep )
```

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.

Return type

long The result of calling this function/method - see return codes below.

Return codes

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 35. 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 223 for a complete list of functions/methods.

Delete()

This function/method 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 notes

- See "Action functions/methods" on page 128 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.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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessInstanceMonitor()/ ObtainInstanceMonitor

This function/method retrieves the process 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 block instance monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes

- See "Action functions/methods" on page 128 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 FmcjItemObtainProcessInstanceMonitor(
       FmcjItemHandle                       hdlItem,
       bool                                 deep,
       FmcjProcessInstanceMonitorHandle *   monitor)

#define FmcjActivityInstanceNotificationObtainProcessInstanceMonitor
        FmcjItemObtainProcessInstanceMonitor
#define FmcjProcessInstanceNotificationObtainProcessInstanceMonitor
        FmcjItemObtainProcessInstanceMonitor
#define FmcjWorkitem
        FmcjItemObtainProcessInstanceMonitor
```

C++ language signature

```
APIRET ObtainProcessInstanceMonitor(
       FmcjProcessInstanceMonitor &    monitor,
       bool                            deep= false ) const
```

Java signature

```
public abstract
ProcessInstanceMonitor obtainProcessInstanceMonitor(
                       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 whose process instance monitor is to be retrieved.

monitor Input/Output. The address of the handle to the process instance monitor respectively the process instance monitor object to be set.

returnCode Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET The return code of calling this function/method - see return codes below.

ProcessInstanceMonitor*/ ProcessInstanceMonitor
A pointer to the process instance monitor or the process instance monitor the item is a part of.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ProcessInstance()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method - 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 function/method completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This function/method refreshes the item from the MQ Workflow execution server (action call).

All information about the item, primary and secondary, is retrieved.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException
FMC_OK(0) The function/method completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This function/method sets the description of the item to the specified value (action call).

If no description is provided, the description of the item is reset to the description of the associated activity instance or process instance.

The following rules apply for specifying an item description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetName()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Transfer()

This function/method transfers an item to the specified user (action call).

Notifications can always be transferred. A work item must be in states *Ready*, *InError*, *Executed*, *Suspending*, or *Suspended* and the associated process instance in states *Running*, *Suspending*, or *Suspended*.

The user who transfers the item must be the owner of the item or have work item authorization for the owner of the item and have work item authorization for the new owner.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 36. 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 “Persistent list” on page 226 for a complete list of functions/methods.

Delete()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh

This function/method 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 notes

- See "Action functions/methods" on page 128 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            FmcjWorklistRefresh
```

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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This function/method sets the description of the persistent list to the specified value (action call).

If no description is provided, the description of the persistent list is erased.

The following rules apply for specifying a persistent list description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetFilter()

This function/method sets the filter of the persistent list to the specified value (action call).

If no filter is provided, the current filter of the persistent list is erased. This means that all objects authorized for will be selected via this list.

Refer to the appropriate list creation for a description of a valid filter syntax.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetSortCriteria()

This function/method sets the sort criteria of the persistent list to the specified value (action call).

If no sort criteria are provided, the current sort criteria of the persistent list are erased. This means that objects selected via this list will not be sorted.

Refer to the appropriate list creation for a description of a valid sort criteria syntax.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetThreshold()

This function/method sets the threshold of the persistent list to the specified value (action call).

If no threshold is provided, the threshold of the persistent list is erased. This means that all objects contained in the list will be provided when queried.

Usage notes

- See “Action functions/methods” on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 37. 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 228 for a complete list of functions/methods.

Refresh()

This function/method refreshes the person from the MQ Workflow execution server (action call).

All information about the person, primary and secondary, is retrieved.

Usage notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

SetAbsence()

This function/method sets the absence indication of the logged-on user to the specified value (action call).

When a person is absent, this person does not participate in staff resolution, that is, this person does not get assigned any work items.

Usage notes

- See "Action functions/methods" on page 128 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 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

SetSubstitute()

This function/method 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.

Usage notes

- See "Action functions/methods" on page 128 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 SetSubstitute( BSTR substitute, boolean isNull )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjPersistentListSetSubstitute(
                    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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_USER(132)
The specified user ID does not correspond to the syntax rules or the user cannot be logged on and be the substitute at the same time.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_USERID_UNKNOWN(10)
The specified user ID is not a registered 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 38. 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:

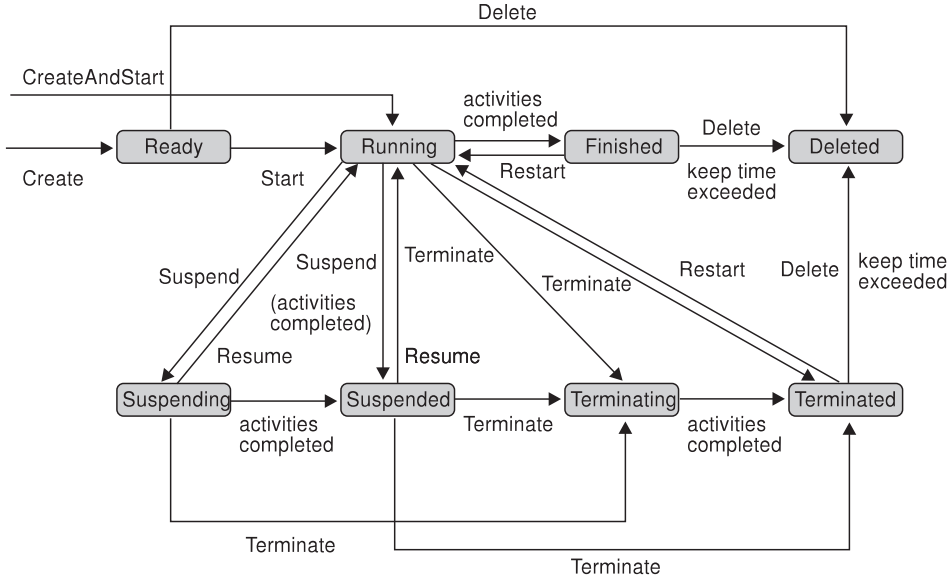


Figure 8. Process instance states

The following sections describe the actions which can be applied on a process instance. See "Process instance" on page 233 for a complete list of functions/methods.

Delete()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ReadWriteContainer
The input container of the process instance.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainMonitor()

This function/method 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 block instance monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the process instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance monitor handle already points to some object.

Usage notes
- See "Action functions/methods" on page 128 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 FmcjProcessInstanceObtainMonitor(
       FmcjProcessInstanceHandle          hdlInstance,
       bool                               deep,
       FmcjProcessInstanceMonitorHandle * monitor )
```

C++ language signature

```
APIRET ObtainMonitor( FmcjProcessInstanceMonitor & monitor,
                      bool                         deep= false )
```

Java signature

```
public abstract
ProcessInstanceMonitor obtainMonitor( 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 function/method - see return codes below.

InstanceMonitor*/ProcessInstanceMonitor
A pointer to the process instance monitor respectively the process instance monitor.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

PersistentObject()

This function/method retrieves the process instance identified by the passed object identifier from the MQ Workflow execution server (action call).

The MQ Workflow execution server from which the process instance is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information, primary and secondary, of the process instance.

In C++, when the process instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance handle already points to some object. In Java, a process instance is newly created; the execution service acts as a factory.

Usage notes

- See "Action functions/methods" on page 128 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.ExecutionService

ActiveX signature

```
long PersistentObject( ExecutionService service, BSTR oid )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstancePersistentObject(
       FmcjExecutionServiceHandle  service,
       char const *                oid,
       FmcjProcessInstanceHandle * hdlInstance )
```

C++ language signature

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java signature

```
public abstract
  ProcessInstance ExecutionService.persistentObject( String oid )
throws FmcException
```

Parameters

hdlInstance Input/Output. The address of the handle to the process instance object to be set.

oid Input. The object identifier of the process instance to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The result of calling this function/method - see return codes below.

ProcessInstance
The process instance retrieved.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_INVALID_OID(805)
The provided oid is invalid.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This function/method refreshes the process instance from the MQ Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This function/method 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 function/method.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Resume()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0)	The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This function/method sets the description of the process instance to the specified value (action call).

If no description is provided, the description of the process instance is erased.

The following rules apply for specifying a process instance description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage notes

- See “Action functions/methods” on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetName()

This function/method sets the name of the process instance to the specified value (action call).

The process instance must still be in the *Ready* state.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:

  `* ? " ; : . $`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Start()

This function/method 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.

See start; additionally allows to pass an input container.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Suspend()

This function/method suspends (temporarily stops) the process instance (action call).

The process instance must be in state *Running*. All non-autonomous subprocesses with respect to control autonomy are also suspended if the deep option is true. Autonomous subprocesses are not considered.

The process instance remains in state *Suspending* as long as there are running program activity implementations or suspending non-autonomous subprocesses. When the activity implementations completed their executions and when the non-autonomous subprocesses reached the *Suspended* state, the process instance is put into the *Suspended* state.

Optionally, a date may be specified up to when the process instance is suspended; it is then automatically resumed, together with the non-autonomous subprocesses, if the deep option had been specified.

See suspend; additionally allows to provide a date at which the process instance is automatically resumed.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1) A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This function/method terminates a process instance and all of its non-autonomous subprocesses (action call).

The process instance must be in states *Running*, *Suspended*, or *Suspending*.

The process instance is put into state terminating as long as there are running activity implementations or terminating non-autonomous subprocesses. When the activity implementations completed their executions or when the non-autonomous subprocesses terminated, the process instance is put into the *Terminated* state. When the process instance has reached the *Terminated* state, it is deleted depending on the setting of the "delete finished items" option.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 39. 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 “Process instance list” on page 238 for a complete list of functions/methods.

QueryProcessInstances()

This function/method 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 function/method.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessInstance[]
The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query worklists (C-language)" on page 653
- For a C++ example see "Query worklists (C++)" on page 655

Chapter 40. Process instance notification actions

An FmcjProcessInstanceNotification or a ProcessInstanceNotification object represents a notification on a process instance assigned to a user.

Other items assigned to users are activity instance notifications and work items. FmcjItem or Item represents the common properties of all items.

In the C++ language, FmcjProcessInstanceNotification is thus a subclass of the FmcjItem class and inherits all properties and methods. In the Java language, ProcessInstanceNotification 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 FmcjProcessInstanceNotification. In ActiveX, inheritance is not supported so that all functions are explicitly defined on ProcessInstanceNotification. Note, however, that they are described as Item actions.

A process instance notification is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on a process instance notification. See “Process instance notification” on page 239 for a complete list of functions/methods.

PersistentObject()

This function/method retrieves the process instance notification identified by the passed object identifier from the MQ Workflow execution server (action call).

The MQ Workflow execution server from which the process instance notification is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the process instance notification.

In C++, when the process instance notification object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance notification handle already points to some object. In Java, a process instance notification is newly created; the execution service acts as a factory.

Usage notes

- See "Action functions/methods" on page 128 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.ExecutionService

ActiveX signature

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceNotificationPersistentObject(
       FmcjExecutionServiceHandle              service,
       char const *                            oid,
       FmcjProcessInstanceNotificationHandle *  hdlItem )
```

C++ language signature

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java signature

```
public abstract
  ProcessInstanceNotification
  ExecutionService.processInstanceNotification( String oid )
throws FmcException
```

Parameters

hdlItem Input/Output. The address of the handle to the process instance notification object to be set.

oid Input. The object identifier of the process instance notification to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The result of calling this function/method - see return codes below.

ProcessInstanceNotification
The process instance notification retrieved.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance notification does no longer exist.

FMC_ERROR_INVALID_OID(805)
The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 41. 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 "Process template" on page 241 for a complete list of functions/methods.

CreateAndStartInstance()

This function/method 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 notes

- See "Action functions/methods" on page 128 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
```

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 function/method - 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template 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 function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

CreateInstance()

This function/method 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 notes

- See “Action functions/methods” on page 128 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 function/method - 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template 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 function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Delete()

This function/method 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.

See delete; additionally allows for specifying the versions to be deleted.

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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template or its specified versions do no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ExecuteProcessInstance()

This function/method creates a process instance from the specified process template and causes the execution of the resulting process instance (action call).

This function/method can be called synchronously and asynchronously. When called asynchronously, a user context can be specified to correlate the response received later. The correlation ID returned can be used to wait for the specific response.

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.

Usage notes

- See "Action functions/methods" on page 128 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	not supported

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 )
```

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. 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 function/method - see return codes below.

ProcessInstance*
A pointer to the newly created and executed process instance.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_INVALID_CORRELATION_ID
The specified correlation ID does not point to FMCJ_NO_CORRELID.

FMC_ERROR_INVALID_NAME(134)
The specified process instance name does not comply with the syntax rules.

FMC_ERROR_INVALID_USER_CONTEXT(819)
The specified user context is longer than 254 characters.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This function/method retrieves the input container associated with the process template from the MQ Workflow execution server (action call).

Usage notes

- See "Action functions/methods" on page 128 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 InContainer( Container * input )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateInContainer(
       FmcjProcessTemplateHandle      hdlTemplate,
       FmcjReadWriteContainerHandle * input )
```

C++ language signature

```
APIRET InContainer( FmcjReadWriteContainer & input )
```

Java signature

```
public abstract
ReadWriteContainer inContainer() 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 function/method - see return codes below.

ReadWriteContainer
The input container of the process template.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

PersistentObject()

This function/method retrieves the process template identified by the passed object identifier from the MQ Workflow execution server (action call).

The MQ Workflow execution server from which the process template is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the process template.

In C++, when the process template object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process template handle already points to some object. In Java, a process template is newly created; the execution service acts as a factory.

Usage notes
- See "Action functions/methods" on page 128 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.ExecutionService

ActiveX signature

```
long PersistentObject( ExecutionService service, BSTR oid )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplatePersistentObject(
       FmcjExecutionServiceHandle  service,
       char const *                oid,
       FmcjProcessTemplateHandle * hdlTemplate )
```

C++ language signature

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java signature

```
public abstract
  ProcessTemplate ExecutionService.processTemplate( String oid )
throws FmcException
```

Parameters

hdlTemplate Input/Output. The address of the handle to the process template object to be set.

oid Input. The object identifier of the process template to be retrieved.

service Input. The service object representing the session with the execution server.

Return type

long/ APIRET The result of calling this function/method - see return codes below.

ProcessTemplate

The process template retrieved.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_INVALID_OID(805)
The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This function/method refreshes the process template from the MQ Workflow execution server (action call).

All information about the process template - primary and secondary - is retrieved.

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 42. 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 “Process template list” on page 244 for a complete list of functions/methods.

QueryProcessTemplates()

This function/method 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 function/method.

Usage notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

ProcessTemplate[]

The qualifying process templates.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)

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

FMC_ERROR_EMPTY(122)

The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)

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

FMC_ERROR_DOES_NOT_EXIST(118)

The process template list 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query worklists (C-language)" on page 653
- For a C++ example see "Query worklists (C++)" on page 655

Chapter 43. 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 250 for a complete list of functions/methods.

Refresh()

This function/method refreshes the log on status from the server(action call).

Usage notes

- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetPassword()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 function/method - see return codes below.

Return codes/ FmcException

FMC_OK(0) The function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

UserSettings()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 function/method - 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 function/method 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Chapter 44. 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. Note that the actions and possible states are dependent on the process instance state, the work item is a part of.

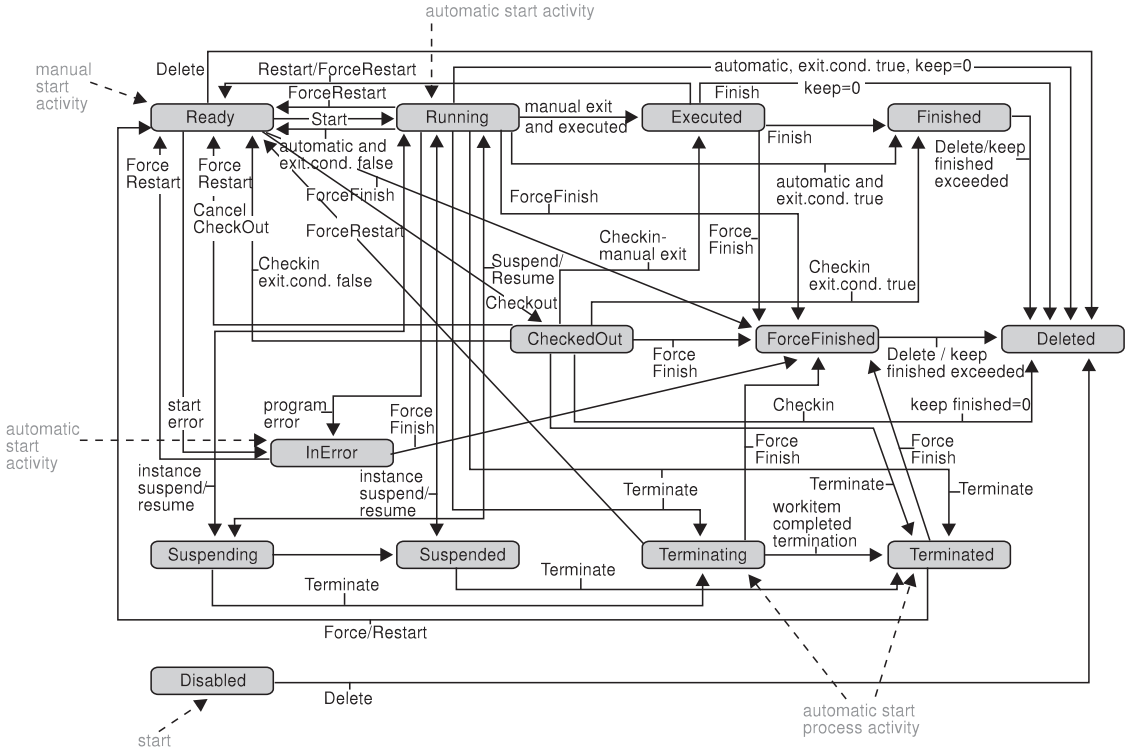


Figure 9. Work item states - process instance state running

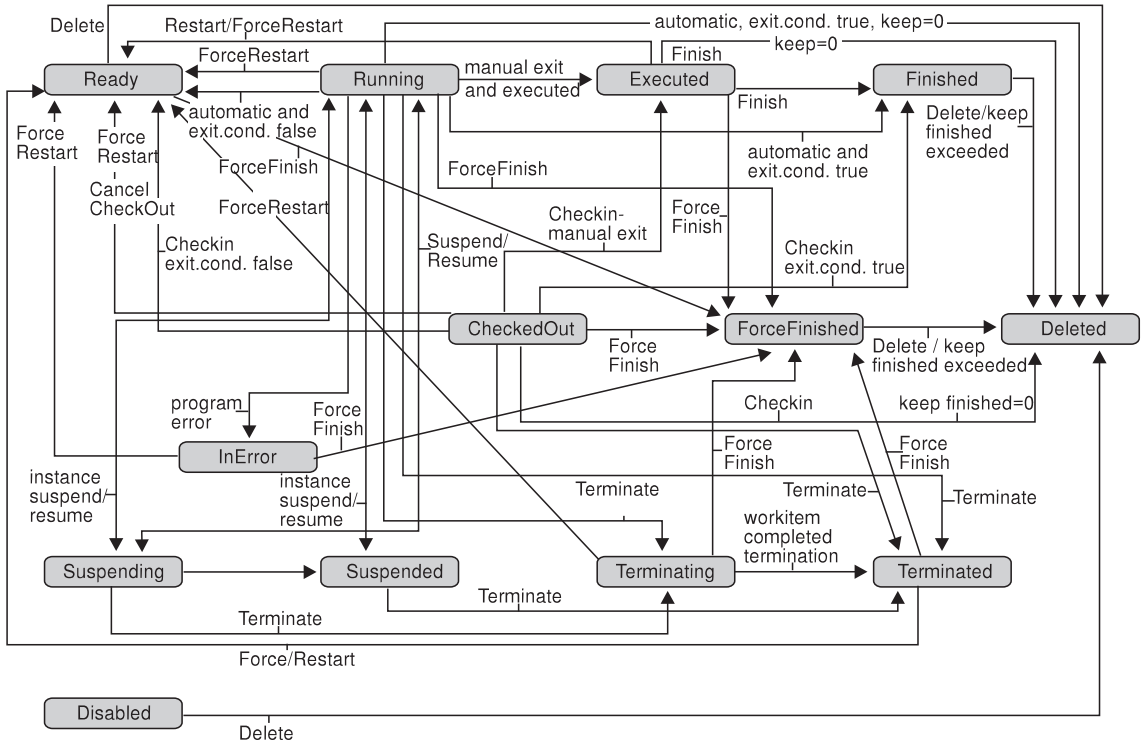


Figure 10. Work item states - process instance state suspending or suspended

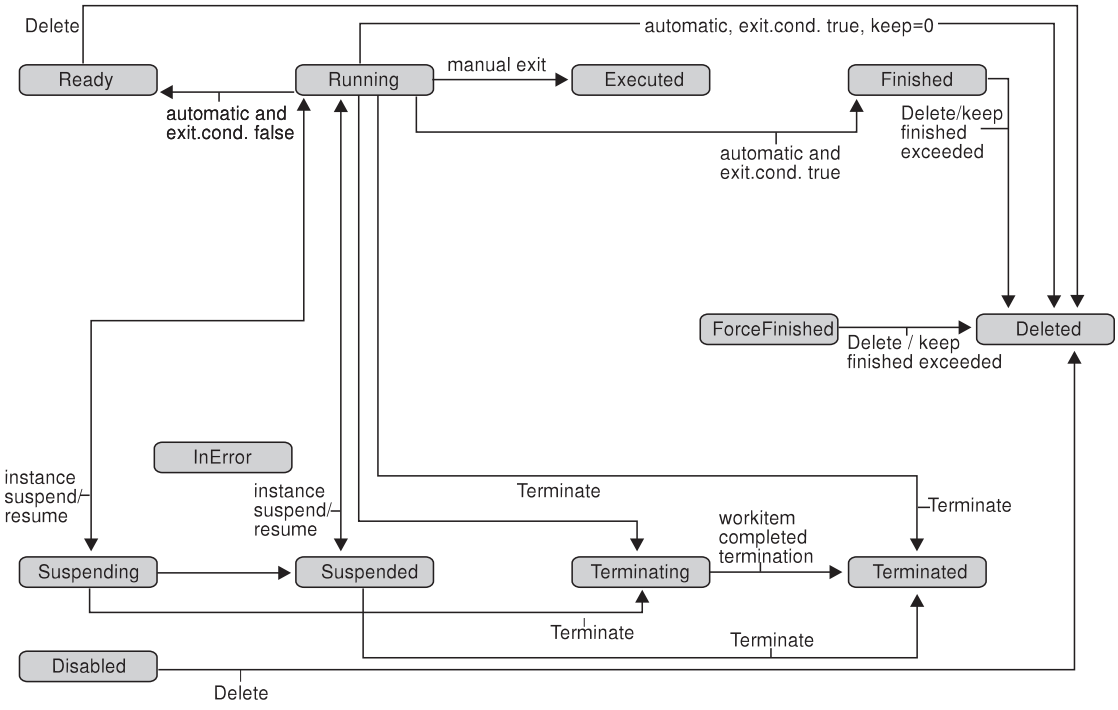


Figure 11. Work item states - process instance state terminating or terminated

The following sections describe the actions which can be applied on a work item. See "Work item" on page 253 for a complete list of functions/methods.

CancelCheckOut()

This function/method cancels the checkout of the work item (action call).

The work item must have been checked out and is put into the *Ready* state. The associated process instance must be in the *Running, Suspending, Suspended,* or *Terminating* state.

Usage notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The work item or process instance is not in a required state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An 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.

CheckIn()

This function/method allows for the check in of a work item that was previously checked out for user processing (action call).

Checking in a work item tells 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.

Usage notes
- See "Action functions/methods" on page 128 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 function/method- see below.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The work item is not checked out.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

CheckOut()

This function/method checks out a ready work item for user processing (action call).

The work item must be implemented by a program.

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 associated process instance must be in the *Running* state.

The caller can request program definitions for specific operating system platforms. The following enumeration types can be used to specify the requested program data.

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_WS_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_WS_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_WS_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_WS_HPUX
C++	FmcjImplementationData::HPUX
JAVA	Basis.HPUX

Windows95 indicates that the program definition for the Windows 95 platform is requested.

ActiveX	Basis_Windows95

	C-language	Fmc_WS_Windows95
	C++	FmcjImplementationData::Windows95
	JAVA	Basis.WINDOWS_95
WindowsNT	indicates that the program definition for the Windows NT platform is requested.	
	ActiveX	Basis_WindowsNT
	C-language	Fmc_WS_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_WS_OS390
	C++	FmcjImplementationData::OS390
	JAVA	Basis.WINDOWS_OS390
Solaris	indicates that the program definition for the Solaris platform is requested.	
	ActiveX	Basis_Solaris
	C-language	Fmc_WS_Solaris
	C++	FmcjImplementationData::Solaris
	JAVA	Basis.Solaris

See checkOut; additionally allows for specifying which program definitions to retrieve.

Usage notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

Finish()

This function/method ends the execution of a manual-exit work item (action call).

The work item must be in state *Executed*, that is, must have run at least once. The work item is then put into the *Finished* state. Depending on the "delete finished items" option, it is deleted.

Usage notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

ForceFinish()

This function/method ends the execution of a work item which is known to have completed in cases where MQ Workflow did not recognize this event (action call).

This situation can occur when the execution server aborted before it received the activity implementation completion message.

A work item implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in the states *Ready, Executed, InError*, or *Terminated*. The associated process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

The work item is then put into the *ForceFinished* state. The exit condition is considered to be true and navigation proceeds.

Depending on the "delete finished items" option, the work item is deleted.

Usage notes

- See "Action functions/methods" on page 128 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()
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemForceFinish( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET ForceFinish()
```

Java signature

```
public abstract
void   forceFinish() 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

ForceRestart()

This function/method forces MQ Workflow to enable the restart of a work item (action call).

A work item implemented by a program must be in states *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in states *Executed, InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

It is then reset into the *Ready* state. Note that automatic activity instances must now be started manually.

Usage notes
- See "Action functions/methods" on page 128 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()
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemForceRestart(
       FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET ForceRestart()
```

Java signature

```
public abstract
void   forceRestart() 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An 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.

InContainer()

This function/method retrieves the input container associated with the work item from the MQ Workflow execution server (action call).

Usage notes
- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This function/method retrieves the output container associated with the work item from the MQ Workflow execution server (action call).

Usage notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

PersistentObject()

This function/method retrieves the work item identified by the passed object identifier from the MQ Workflow execution server (action call).

The MQ Workflow execution server from which the work item is to be retrieved is identified by the execution service object. The transient object is then created or updated with all information - primary and secondary - of the work item.

In C++, when the work item object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the Work item handle already points to some object. In Java, a work item is newly created; the execution service acts as a factory.

Usage notes
- See "Action functions/methods" on page 128 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.ExecutionService

ActiveX signature

```
long PersistentObject( ExecutionService service, BSTR oid )
```

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemPersistentObject( FmcjExecutionServiceHandle service,
                              char const *               oid,
                              FmcjWorkitemHandle *       hdlWorkitem )
```

C++ language signature

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java signature

```
public abstract
  WorkItem ExecutionService.persistentWorkItem( String oid )
throws FmcException
```

Parameters

hdlWorkitem	Input/Output. The address of the handle to the work item object to be set.
oid	Input. The object identifier of the work item to be retrieved.
service	Input. The service object representing the session with the execution server.

Return type

long/ APIRET	The return code of calling this method - see below.
WorkItem	The work item retrieved.

Return codes/ FmcException

FMC_OK(0) The function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_INVALID_OID(805)
The provided oid is invalid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This function/method 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 notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An 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.

Start()

This function/method 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 notes
- See "Action functions/methods" on page 128 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
```

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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An 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.

StartTool()

This function/method 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 notes
- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The work item 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 function/method.
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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This function/method terminates a work item implemented by a program or process (action call).

If the work item is implemented by a program, it must be in the states *CheckedOut* or *Running* and the process instance must be in the states *Running, Suspending*, or *Suspended*. If the work item is implemented by a process, it must be in the states *Running, Suspending*, or *Suspended* and the process instance must be in the states *Running, Suspending, Suspended*, or *Terminating*.

A work item implemented by a process is terminated together with all its non-autonomous subprocesses with respect to control autonomy.

The work item is then put into the *Terminating* or *Terminated* state.

Depending on the “delete finished items” option, the work item is deleted.

When the *Terminated* state has been reached, the exit condition is considered to be false, the output container and especially the return code (_RC) are not set,

and navigation ends. If not yet deleted, navigation can be explicitly continued by a user with process administration rights, that is, ForceFinish() or ForceRestart() repair actions can be called.

Usage notes

- See "Action functions/methods" on page 128 for general information.

Authorization

Be the work item owner

For work items implemented by a process, additionally one of:

- Process administration authority
- Be the process administrator
- Be the system administrator

Required connection

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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the function/method.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An 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.

Chapter 45. Work list actions

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 257 for a complete list of functions/methods.

QueryActivityInstanceNotifications()

This function/method 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 function/method. 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 notes
- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The worklist 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query work items from a worklist (ActiveX)" on page 669
- For a C-language example see "Query work items from a worklist (C-language)" on page 670
- For a C++ example see "Query work items from a worklist (C++)" on page 672
- For a Java example see "Query work items from a worklist (Java)" on page 673

QueryItems()

This function/method 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 function/method. 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 notes
- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query work items from a worklist (C-language)" on page 670
- For a C++ example see "Query work items from a worklist (C++)" on page 672
- For a Java example see "Query work items from a worklist (Java)" on page 673

QueryProcessInstanceNotifications()

This function/method 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 function/method. 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 notes
- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see “Query work items from a worklist (ActiveX)” on page 669
- For a C-language example see “Query work items from a worklist (C-language)” on page 670
- For a C++ example see “Query work items from a worklist (C++)” on page 672
- For a Java example see “Query work items from a worklist (Java)” on page 673

QueryWorkitems()

This function/method 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 function/method. 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 notes

- See "Action functions/methods" on page 128 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 function/method completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, does not yet represent a persistent one.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist 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_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For an ActiveX example see "Query work items from a worklist (ActiveX)" on page 669
- For a C-language example see "Query work items from a worklist (C-language)" on page 670
- For a C++ example see "Query work items from a worklist (C++)" on page 672

- For a Java example see “Query work items from a worklist (Java)” on page 673

Part 7. 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 46. 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 47. 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 48. 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 49. 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 50. 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.

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.

NewActivityInstanceNotification

Creates an empty activity instance notification object.

Signature

```
ActivityInstanceNotification * NewActivityInstanceNotification()
```

Return type
ActivityInstanceNotification*
A pointer to the activity instance notification created.

NewProcessInstance

Creates an empty process instance object.

Signature

```
ProcessInstance * NewProcessInstance()
```

Return type
ProcessInstance*
A pointer to the process instance object 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.

NewWorkitem

Creates an empty work item object.

Signature

```
Workitem * NewWorkitem()
```

Return type

Workitem* A pointer to the work item object 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 51. 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 52. 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 53. 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 selected object of the process template list array, the process instance list, or the worklist array.

Signature

```
IDispatch * FindFirst( 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.

Return type

IDispatch* The object pointer. NULL (0) is returned if the selected object is not found.

FindNext

Returns the next selected object of the process template list array, the process instance list array, or the worklist array. FindFirst() must have been called before.

Signature

```
IDispatch * FindNext( 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.

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 array, the process instance list array, or the worklist array 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 54. 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 55. 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 56. 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 57. 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 52. Methods supported by all GUI controls" on page 579:

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 54. Events triggered by all GUI controls" on page 589 and "Chapter 55. Events triggered by all non-monitor GUI controls" on page 591:

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 58. 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 52. Methods supported by all GUI controls" on page 579 and "Chapter 53. Methods supported by all list controls" on page 583:

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 54. Events triggered by all GUI controls" on page 589, "Chapter 55. Events triggered by all non-monitor GUI controls" on page 591, and "Chapter 56. Events triggered by all list controls" on page 595.

Chapter 59. 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 52. Methods supported by all GUI controls" on page 579 and "Chapter 53. Methods supported by all list controls" on page 583:

ContextMenuRestart

Calls the context menu item 'Restart'.

This method restarts the selected list of process instances within the Control. See "Restart()" on page 449 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 451 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 451 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 458 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 460 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 460 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 463 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 54. Events triggered by all GUI controls" on page 589, "Chapter 55. Events triggered by all non-monitor GUI controls" on page 591, and "Chapter 56. Events triggered by all list controls" on page 595.

Chapter 60. 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 52. Methods supported by all GUI controls" on page 579 and "Chapter 53. Methods supported by all list controls" on page 583:

ContextMenuFinish

Calls the context menu item 'Finish'.

This method finishes the selected list of work items within the Control. See "Finish()" on page 524 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 526 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 528 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 536 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 proceesing 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 538 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 540 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 409 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 54. Events triggered by all GUI controls” on page 589, “Chapter 55. Events triggered by all non-monitor GUI controls” on page 591, and “Chapter 56. Events triggered by all list controls” on page 595.

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 61. 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 52. Methods supported by all GUI controls" on page 579.

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

```
long ConnectGUI( IDispatch * wfc )
```

Parameters

wfc Input. The pointer to the MQWorkflowCtrl object.

Return type

long If processing completes successfully, FMC_OK is returned.

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 54. Events triggered by all GUI controls" on page 589.

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 8. Examples and scenarios

Chapter 62. 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 Windows NT and ActiveX in \fmcwinnt\scenario\credit; for Windows 95 and ActiveX in \fmcwin95\scenario\credit; for Windows 98 and ActiveX in \fmcwin98\scenario\credit:

  – The FDL: fmccred.fdl

  – The activity implementations: fmcn6bnp.vbp, fmcn6bni.vbp, fmcn6bnp.vbp, fmcn6bnr.vbp

- A sample life insurance request

  For Windows NT and the C-language in \fmcwinnt\scenario\life; for Windows 95 and the C-language in \fmcwin95\scenario\life; for Windows 98 and the C-language in \fmcwin98\scenario\life:

  – The FDL: fmclife.fdl

  – The activity implementations: all *.c and *.h files starting with fmcsi

Chapter 63. 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.

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 Windows NT and ActiveX in \fmcwinnt\smp\vb\actimpl; for Windows 95 and ActiveX in \fmcwin95\smp\vb\actimpl; for Windows 98 and ActiveX in \fmcwin98\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 Windows NT and ActiveX in \fmcwinnt\smp\vb\rtc; for Windows 95 and ActiveX in \fmcwin95\smp\vb\rtc; for Windows 98 and ActiveX in \fmcwin98\smp\vb\rtc: fmcn6rtc.vbp
- Hello world

  For all supported platforms (except OS/2) 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

The following chapters additionally show some examples. They are intended to present the stated concept only.

Chapter 64. 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;
  }


  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%\n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }


 /* create a process instance list */
 rc = FmcjExecutionServiceCreateProcessInstanceList(
                          service,
                          name,
                          Fmc_LT_Private,
                          "USERID",
                          desc,
                          FmcjNoFilter,
                          FmcjNoSortCriteria,
                          &threshold,
                          &instanceList );
```

```
  if ( rc != FMC_OK)
    printf( "CreateProcessInstanceList returns: %u%\n",rc );
  else
    printf( "CreateProcessInstanceList okay\n" );

  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

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;

  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;
}
```

Create a process instance list (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class CreateProcInstList
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java CreateProcessInstanceList
                                  <agent> <LOC|RMI|OSA|IOR|COS>
                                  [userid] [password]");
      System.exit(0);
    }
```

```
try
    {
      // An agent bean representing 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);
      }
```

```
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.
service.logon(userid, passwd);
System.out.println("Logon successful");

String ListName      ="MyTenInstances";
String ListDesc      = "List contains no more than 10 instances";
String ListFilter    = "";
String ListSort      = "";
int    ListThreshold = 10;


try
{
  service.createProcessInstanceList( ListName, TypeOfList.PRIVATE,
                                     userid  , ListDesc, ListFilter,
                                     ListSort, ListThreshold);
  System.out.println("Private ProcessInstanceList created successfully");
}
catch(FmcException e)
{
   if ( e.rc == FmcException.FMC_ERROR_NOT_UNIQUE )
   {
     System.out.println("ProcessInstanceList: '" + ListName +
                        "' already exists");
   }
}
```

```
     finally
     {
        // Logoff from the execution service. This (like any other remote call)
        // may raise an FmcException indicating a communication failure.
        service.logoff();

        System.out.println("Logoff successful");
      }
    }



catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Chapter 65. 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]= "";


  FmcjGlobalConnect();

  /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%\n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%\n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }


  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &lists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%\n",rc );
  else
     printf( "QueryWorklists() returns okay\n" );
```

```
if (rc == FMC_OK)
{
  numWList= FmcjWorklistVectorSize(lists);
  printf ("Number of worklists returned : %u\n", numWList);
  for( i=1; i<= numWList; i++ )
  {
    worklist= FmcjWorklistVectorNextElement(lists);
    FmcjWorklistName( worklist, tInfo, 4097 );
    printf("- Name                          : %s\n",tInfo);



    enumValue= FmcjWorklistType(worklist);
    if ( enumValue == Fmc_LT_Private )
      printf("- Type                          : %s\n","private");
    if ( enumValue == Fmc_LT_Public )
      printf("- Type                          : %s\n","public");

    FmcjWorklistOwnerOfList( worklist, tInfo, 4097 );
    printf("- OwnerOfList                   : %s\n",tInfo);
    printf("- OwnerOfList is null ?         : %u\n",
              FmcjWorklistOwnerOfListIsNull(worklist) );


    FmcjWorklistDescription( worklist, tInfo, 4097 );
    printf("- Description                   : %s\n",tInfo);
    printf("- Description is null ?         : %u\n",
              FmcjWorklistDescriptionIsNull(worklist) );


    FmcjWorklistFilter( worklist, tInfo, 4097 );
    printf("- Filter                        : %s\n",tInfo);
    printf("- Filter is null ?              : %u\n",
              FmcjWorklistFilterIsNull(worklist) );


    FmcjWorklistSortCriteria( worklist, tInfo, 4097 );
    printf("- SortCriteria                  : %s\n",tInfo);
    printf("- SortCriteria is null ?        : %u\n",
              FmcjWorklistSortCriteriaIsNull(worklist) );


    printf("- Threshold                     : %u\n",
              FmcjWorklistThreshold(worklist) );
    printf("- Threshold is null ?           : %u\n",
              FmcjWorklistThresholdIsNull(worklist) );
                                /* deallocate just read object  */
    FmcjWorklistDeallocate(&worklist);
  }
  FmcjWorklistVectorDeallocate(&lists);
}
```

```
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Query worklists (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();

  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }


  // query worklists

  vector<FmcjWorklist> lists;
  FmcjWorklist         worklist;
  rc = service.QueryWorklists( lists );
  if ( rc != FMC_OK)
     cout << "QueryWorklists() returns: " << rc << endl;
  else
     cout << "QueryWorklists returns okay" << endl;


  if (rc == FMC_OK)
  {
    unsigned int numWList= lists.size();
    cout << "Number of worklists returned : " << numWList << endl;


    for( unsigned long i=0; i< numWList; i++ )
    {
      worklist= lists[i];
      cout << "Name              : " << worklist.Name()              << endl;
```

```
        cout << "Type              : " <<
                ((worklist.Type() == FmcjPersistentList::Private) ? "private" :
                 (worklist.Type() == FmcjPersistentList::Public)  ? "public"  :
                 "not set" )           << endl;


       cout << "Owner             : " << worklist.OwnerOfList()       << endl;
        cout << "Owner null ?      : " << worklist.OwnerOfListIsNull() << endl;


        cout << "Description       : " << worklist.Description()       << endl;
        cout << "Description null ?: " << worklist.DescriptionIsNull() << endl;


       cout << "Filter            : " << worklist.Filter()            << endl;
        cout << "Filter null ?     : " << worklist.FilterIsNull()      << endl;
        cout << "SortCriteria      : " << worklist.SortCriteria()      << endl;
        cout << "SortCriteria null?: " << worklist.SortCriteriaIsNull()<< endl;


        cout << "Threshold         : " << worklist.Threshold()         << endl;
        cout << "Threshold null ?  : " << worklist.ThresholdIsNull()   << endl;
        cout << endl;     }     cout << endl;   }


  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

Query worklists (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class QueryWorkLists
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password
    //
    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java QueryWorkLists [userid] [password]");
      System.exit(0);
    }
```

```
try
{
  // An agent bean representing 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);
  }



  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");



     // Query the set of worklists the logged on user can access.
     WorkList[] worklists = service.queryWorkLists();

     if (worklists.length == 0)
     {
       System.out.println("  No worklist found");
     }
     else
     {
       System.out.println("  Number of worklists returned: " + worklists.length
);


       // Iterate over the worklists, printing out their names.
       for (int ndx = 0; ndx < worklists.length; ndx++)
       {
         System.out.println("     Name               :" + worklists[ndx].name());

         if (worklists[ndx].type() == TypeOfList.PUBLIC )
         {
           System.out.println("     Type               :Public " );
         }
         else if (worklists[ndx].type() == TypeOfList.PRIVATE )
         {
           System.out.println("     Type               :Private" );
         }
         else
         {
           System.out.println("     Type               :NotSet " );
         }
```

```
            System.out.println("    Owner              :" + worklists[ndx].ownerOfList());
            System.out.println("    Description        :"+ worklists[ndx].description());
            System.out.println("    Filter             :"+ worklists[ndx].filter());
            System.out.println("    SortCriteria       :"+ worklists[ndx].sortCriteria());
            System.out.println("    Threshold          :"+ worklists[ndx].threshold());
            System.out.println("    ");
          }

        }/* End if*/



        // Logoff from the execution service. This (like any other remote call)
        // may raise an FmcException indicating a communication failure.
        service.logoff();

        System.out.println("Logoff successful");
      }


    catch(FmcException e)
      {
        // Catch and report details about the FmcException
        System.out.println("FmcException occured");
        System.out.println("  RC         : " + e.rc);
        System.out.println("  Origin     : " + e.origin);
        System.out.println("  MessageText: " + e.messageText);
        System.out.println("  Exception  : " + e.getMessage());
        System.out.println("  Parameters : ");
        for ( int i = 0; i < e.parameters.length ; i++)
        {
          System.out.println("    " + e.parameters[i] );
        }
        System.out.println("  StackTrace : ");
        e.printStackTrace();
      }


     catch(Exception e)
      {
        // Catch and report any exception that occurred.
        e.printStackTrace();
      }

      System.exit(0);
    }
  }
```

Chapter 66. 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" );
```

```
  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;
}
```

Query process instances (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();

  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }
```

```
  // query process instances

  vector<FmcjProcessInstance> instances;

  rc = service.QueryProcessInstances(
                   FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
                   instances );
  if ( rc != FMC_OK)
     cout << "QueryProcessInstances returns: " << rc << endl;
  else
     cout << "QueryProcessInstances okay" << endl;



  if ( rc == FMC_OK )
  {
    cout << "Number of instances returned: " << instances.size() << endl;

    for ( int i=0; i < instances.size(); i++ )
      cout << "- Name: " << instances[i].Name() << endl;
  }

  service.Logoff();

  FmcjGlobal::Disconnect();
  return 0;
}
```

Query process instances (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryProcInst
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryProcessInstances [userid] [password]");
      System.exit(0);
    }
```

```
try
    {
      // An agent bean representing 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);
      }
```

```
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");


  // Query a set of processinstances (30 at maximum), sort them by name
  ProcessInstance[] procInstances =
    service.queryProcessInstances("","NAME DESC", 30);

  if (procInstances.length == 0)
  {
    System.out.println("  No process instances found");
  }
  else
  {
    System.out.println("Number of instances returned: " + procInstances.length);


    // Iterate over the process instances, printing out their names.
    for (int ndx = 0; ndx < procInstances.length; ndx++)
    {
      System.out.println("   - Name: " + procInstances[ndx].name());
    }
  }
```

```
   // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();

      System.out.println("Logoff successful");
    }



catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


    catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

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(pil.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" );
```

```
  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;
}
```

Query work items from a worklist (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();


  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }


  // query worklists

  vector<FmcjWorklist> lists;
  FmcjWorklist         worklist;

  rc = service.QueryWorklists( lists );
  if ( rc != FMC_OK)
     cout << "QueryWorklists() returns: " << rc << endl;
  else
     cout << "QueryWorklists returns okay" << endl;

  if (rc == FMC_OK)
  {
    unsigned int numWList= lists.size();
    cout << "Number of worklists returned : " << numWList << endl;
    if ( numWList == 0 )
    {
      cout << "No worklist found" << endl;
      return -1;
    }
```

```
    worklist= lists[0];
    cout << "Name              : " << worklist.Name()              << endl;

    vector<FmcjWorkitem> wVector;
    FmcjWorkitem         workitem;

    rc= worklist.QueryWorkitems( wVector );
    cout << "Query workitems of list returns: " << rc << endl;
    cout << "Number of workitems              " << wVector.size() << endl;



    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;
}
```

Query work items from a worklist (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryWorkItems
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryWorkitems [userid] [password]");
      System.exit(0);
    }
```

```
try
{
  // An agent bean representing 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);
  }
```

```
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");



// Query the set of worklists the logged on user can access.
WorkList[] worklists = service.queryWorkLists();

if (worklists.length == 0)
{
  System.out.println("  No worklist found");
}
else
{
  System.out.println("  Number of worklists returned: " + worklists.length);

  WorkList worklist = worklists[0];
  System.out.println("  Name: "+worklist.name());


  // Query the set of workitems in the first worklist.
  WorkItem[] workitems = worklist.queryWorkItems();
  System.out.println("  Number of workitems: " + workitems.length);

  // Iterate over the workitems, printing out their names.
  for (int ndx = 0; ndx < workitems.length; ndx++)
  {
    System.out.println("    " + workitems[ndx].name());
  }
}/* End if*/
```

```
      // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();

      System.out.println("Logoff successful");
    }


 catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


   catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Chapter 67. 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);


  FmcjGlobalConnect();

/*-- retrieve the input container from the PEA who started the program --*/
  rc = FmcjContainerInContainer( &input );
  fprintf(file1, "Get Input Container - rc:  %u\n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }

  fprintf(file1, "Input Container Name: %s\n",
          FmcjReadOnlyContainerType(input, stringBuffer, 4097));
```

```
/*-- retrieve the output container from the PEA who started the program -*/
  rc = FmcjContainerOutContainer( &output );
  fprintf(file1, "Get Output Container - rc:  %u\n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }

  fprintf(file1, "Output Container Name: %s\n",
          FmcjReadWriteContainerType(output, stringBuffer, 4097));



/*----- Modify output values ------------------------------------------*/
  rc= FmcjReadWriteContainerSetLongValue(output, "aFieldInTheOutput",42);
  fprintf(file1, "\nSetting long value returns rc: %u\n", rc);

  ...

/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainerSetOutContainer( output );
  fprintf(file1, "\nSet Output Container - rc: %u\n",rc);
  fflush(file1);

  FmcjGlobalDisconnect();
  fclose(file1);
  return 0;                             // _RC passed to MQ Workflow
}
```

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;
```

```
  FmcjGlobal::Connect();

/*-- retrieve the input container from the PEA who started the program --*/
  FmcjReadOnlyContainer input;

  APIRET rc = FmcjContainer::InContainer( input );
  file1 << "Get Input Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }

  file1 << "Input Container Name: " << input.Type() << endl;



/*-- retrieve the output container from the PEA who started the program -*/
  FmcjReadWriteContainer  output;

  rc = FmcjContainer::OutContainer( output );
  file1 << "Get Output Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }

  file1 << "Output Container Name: " << output.Type() << endl;
/*----- Modify output values ----------------------------------------*/
  rc= output.SetValue("aFieldInTheOutput",42L);
  file1 << "Setting long value returns rc: " << rc << endl;

  ...


/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainer::SetOutContainer( output );
  file1 << "Set Output Container - rc: " << rc << endl;

  FmcjGlobal::Disconnect();
  file1.close();
  return 0;                             // _RC passed to MQ Workflow
}
```

Part 9. Using the Lotus Notes API

This part provides an overview of the API provided for the integration of MQ Workflow and Lotus Notes.

The Lotus Notes API is a C– language API. It is intended to be used within LotusScript code. It is possible to call it from C code directly. However, it is not recommended to do this because most of the Lotus Notes API functions require parameters that are available inside Lotus Notes only.

The Lotus Notes API is available for the OS/2, Windows NT, and Windows 95 platforms.

Chapter 68. Requirements

The MQ Workflow C++ API must be installed.

Header and library files

All LotusScript functions developed for use with the Lotus Notes API must include the external LotusScript files for the Lotus Notes API. These files are already installed on your system if you selected to install the API files during the MQ Workflow installation procedure. The files are EXMP4API.LSS and EXMP4ARC.LSS, and by default are installed in the \fmc\os2\lnc, fmc\winnt\lnc, or \fmc\win95\lnc subdirectory on the selected drive.

All C programs developed for use with the Lotus Notes API must include the Lotus Notes API header files. These files are already installed on your system if you selected to install the API files during the installation procedure. The files are FMC4API.H, FMC4GLO.H, and FMC4ARC.H, and by default are installed in the \fmc\os2\api, fmc\winnt\api, or \fmc\win95\api subdirectory on the selected drive. Applications must include the file FMC4API.H. All other required header files are implicitly included by FMC4API.H.

If these files were not installed when MQ Workflow was initially installed, refer to the *IBM MQSeries Workflow: Installation Guide* for instructions on how to install them.

DLL and shared library files

Ensure that the directory where the MQ Workflow DLL files reside is specified in your environment settings. This is the LIBPATH statement of your CONFIG.SYS file. This is done automatically by a standard MQ Workflow installation.

Compiling

The compilers supported for the Lotus Notes API are:

- For OS/2: IBM VisualAge for C++ 3.0
- For Windows NT or Windows 95: Microsoft Visual C++ 5.0

They are required only if you intend to use the Lotus Notes API within your own C programs.

Chapter 69. Coding examples

When you install the Lotus Notes database templates, the files FMC4R*xxx*.NTF and FMC4S*xxx*.NTF are provided (where *xxx* represents the language identifier, for example, ENU for US English). FMC4R*xxx*.NTF is the database-design template for the MQ Workflow Client for Lotus Notes and FMC4S*xxx*.NTF is a sample database template. These templates use the external LotusScript files EXMP4API.LSS and EXMP4ARC.LSS, which contain the definitions of the MQ Workflow API calls and return codes.

Before working with the provided templates, copy the Lotus Notes database templates to the Lotus Notes data directory, which is usually `NOTES\DATA`, and the external LotusScript files to the Lotus Notes directory, which is usually `NOTES`.

The database design template FMC4R*xxx*.NTF provides the main functions of an MQ Workflow Runtime client by means of the Lotus Notes API. The database template FMC4S*xxx*.NTF provides, in addition to the main functions, some examples on how to implement MQ Workflow activities in Lotus Notes and how to add other functions to the standard functions. All templates are delivered as source code. You can use them as coding examples or modify the templates directly. If you modify the files provided with MQ Workflow, several considerations apply. These are covered in the following sections.

Sample FDL

Import FMC4SMP.FDL into your MQ Workflow database and translate the process NotesCreditRequest before you work with the FMC4S*xxx*.NTF sample.

Overall database design

FMC4R*xxx*.NTF and FMC4S*xxx*.NTF are design templates. FMC4S*xxx*.NTF inherits its design from FMC4R*xxx*.NTF. If you want to customize these templates, consider the following:

- You need at least Lotus Notes development experience.
- If you change the templates, also change the database-template names (IBM_MQSeries_Workflow_XXX_3.1 and IBM_MQSeries_Workflow_XXX_3.1_Sample).
- Only the unchanged templates are part of the product and are directly supported. If you change the templates, it is your responsibility to fix any errors resulting from these changes.

Forms to display the settings of MQ Workflow objects

The sample templates use several forms (which use subforms in most cases) to display the settings information for the different MQ Workflow objects. If an API call creates a Notes document as a representation of an MQ Workflow object, the **Form** item within the document is filled with the respective values (see "Chapter 81. Fields used by the Client for Lotus Notes" on page 753). The forms **(User settings), (Process template), (Process instance), (Work item), (Work-item notification),** and **(Process-instance notification)** contain the respective form aliases.

Each of these forms contains some logic for displaying a document with the respective form where the settings action from the view is used. The actual form name stored in the document is not changed.

In addition, the forms **(User settings)** and **(Work item)** contain the logic for invoking the respective update API function during the save action of the form.

You can change the look and feel of these forms, but be careful when changing the processing logic.

Forms used within dialogs

Some forms are used within dialogs invoked from specific actions:

- **(User settings - Change password)**
- **(Work item - Start support tool)**
- **(Work item - Transfer)**

You can change the look and feel of these forms, but be careful when changing the processing logic.

Forms used to create standard objects

The **Application settings** form allows you to create an application-settings document, if the original document was deleted. The form contains logic to prevent the creation of more than one application-settings document.

The **(User settings)** form allows you to create additional user-settings documents. In a user-settings document, you can specify the information that is necessary for connecting to an MQ Workflow system. The form contains the logic to prevent the creation of more than one user-settings document for the same user ID and MQ Workflow system group. It also contains the logic to

display only those MQ Workflow system groups that have been defined in the MQ Workflow profile at the user's workstation.

The **(Distribute MQ Workflow template)** form is used to supply the information necessary for creating new databases from the current database template. The form contains the logic to delete all documents created with this form before, thus deleting any documents no longer needed.

You can change the look and feel of these forms, but be careful when changing the processing logic.

Forms used to start an MQ Workflow process instance

The **Credit Request** form is used to start an MQ Workflow process with start data. First, the user is prompted for some data. The form contains the action button **Start Credit Request**. With this button, you can initiate the following actions:

1. Search for the correct template document.
2. Create an instance using the Lotus Notes API.
3. Access the instance document.
4. When the instance document is found, add the data from the form and start the instance using the Lotus Notes API.

This form is only an example for how you could start an MQ Workflow process with input data. The provided code does not handle all possible error situations, but shows the necessary minimum logic.

In a production environment, you may want to add functions to such a form. For example, you could add some additional actions to scan documents and store these documents within a central repository. You could pass the key to this information to the MQ Workflow process, so that it is available via the input container during later activity implementations.

Forms used to implement an MQ Workflow activity

With the **ExmWorkitemStart** and **ExmWorkitemCheckout** agents, the following naming convention is used to decide whether an activity is implemented inside Lotus Notes: If the activity implementation is *NOTESFORMxyz*, it is assumed that the activity is implemented by a form named *xyz*.

The forms **AssessCreditRisk, RequestCreditApproval, AcceptCredit,** and **RejectCredit** are examples. They are used within the *NotesCreditRequest* process defined in FMC4SMP.FDL.

These forms contain fields to show the values of the input container as well as fields to set the output container. In addition, they contain the actions:

- **Done**, which saves the user input and invokes the API function for checking in the work item.
- **Save - Rework later**, which saves the user input and closes the document.
- **Undo**, which invokes the API function to force the restart of the work item. This undoes the checkout of the work item.

These forms are examples for MQ Workflow activity implementation inside Lotus Notes. The provided code does not handle all possible error situations but shows the minimum logic necessary.

In a production environment, you may want to add a lot of functions to such forms. For example, you could add some actions to view process-relevant documents, such as scanned documents, or you could enhance the possibilities to use the document for printing or mailing, or you could add an action to call the customer automatically via phone.

Views used by the Lotus Notes API

All views named **MQ Workflow API -** *viewname* are used by the Lotus Notes API and must *not* be changed.

Views and folders used by the end user

The views and folders visible to the end user show settings, process templates, process instances, and work items with predefined columns. The columns can be resized and used for resorting. The view selection is based on `_ExmDocType` and `_ExmState`. The view and folder actions invoke agents suitable for the kind of documents selected for the view.

You can change the views and folders shown to the end user in any way. You can add fields, omit fields, change the order of the fields and the sorting or categorizing criteria. **Work items\By Activity** shows an example of a categorized view.

Note: The views do not show the user ID and database associated with the different MQ Workflow objects in the view. If users log on to more than

one MQ Workflow system or if they do not always log on to the same MQ Workflow system, you might want to add these columns to the view.

You can also remove those actions that should not be available for the end user, or replace those actions with actions more suitable to your end users' needs.

Agents

Agents provide an easy way to add actions to your own views or forms. You may also want to create agents for combined actions, for example, create a process instance and start it with some default data.

With the agents supplied with the sample database templates, you can invoke one Lotus Notes API function per agent. The agents are designed to work on the currently selected document (cursor selection). Multiple selection is not supported. The **resync** agents, however, work on all documents of a specific type. The provided agents include basic error handling.

You can change the agents to your needs, but consider that they are used from the actions of the different views and folders, too.

Navigators

The **MQ Workflow** navigator supplied with the samples shows graphical representations for the different views and folders. When you click on the respective hotspot, it invokes the different views and folders.

You can change the navigator to your needs, for example, add additional hotspots, or change the actions behind existing hotspots.

Outlook

The supplied sample databases show a few possibilities of the combination of Lotus Notes and the Lotus Notes API. Many completely different scenarios are possible.

These are a few ideas:

- **Hide MQ Workflow worklists from the end user.**

  Use a graphical navigator as primary end-user interface. Create graphical representations for the different types of activities the end user is to work with. If the end user clicks on the hotspot, use a LotusScript to search for

such an activity in the end user's worklist. If one is found, check it out, and display it inside a form. If nothing is found, refresh the worklist and search again. Provide a **Next** action inside the form, which checks in the current work item and searches for the next.

- **Add more automation features using server-based agents.**

  If you allow time-triggered agents running on the Notes server to access the user ID and password information inside the database, you could use such agents to automatically log on to MQ Workflow at specific times, refresh the worklist of the user, and process the work items found based on special algorithms (for example, send an e-mail as notification, automatically check out all those work items that have only one person assigned, check in work items with specific flags set inside the corresponding document).

- **Separate MQ Workflow database and Application database.**

  You could create your own Lotus Notes database with your forms and agents, which access the MQ Workflow database and retrieve the data using LotusScript. The advantages would be, for example, that the code is separate and that MQ Workflow is hidden from the end user.

Chapter 70. Restrictions

The following restrictions apply to the MQ Workflow Client for Lotus Notes:

- The sum of the length of all fields (standard fields, input container fields, output container fields) within the Notes document created by the Lotus Notes API is restricted by the maximum summary-buffer size of Lotus Notes.
- Working with the worklists of other users is not supported.
- The Lotus Notes API supports only the pull model for worklist refreshes, that is, the user (or program) has to refresh the worklist explicitly to see changes made by other users. This is also true for the state changes of a running program started via the Program Execution client even if this program has been started from the Client for Lotus Notes.
- There are reserved names:
  - All field names within documents starting with `_EXM` are reserved for the Lotus Notes API.
  - All field names within documents starting with `_Impl` are reserved for the specific implementations within the MQ Workflow Client for Lotus Notes database templates.

Refer to "Chapter 81. Fields used by the Client for Lotus Notes" on page 753 for more information about the fields.

Chapter 71. Data types and functions

MQ Workflow data types are mapped to the following Lotus Notes data types and vice versa:

MQ Workflow data type	Lotus Notes data type
STRING	TEXT
ARRAY of STRING	TEXTLIST
FLOAT	NUMBER
LONG	NUMBER

The return code for all successful function calls is zero (0).

The API functions are divided into the following groups:

General Notes actions
General Notes actions are used to log on to or log off from MQ Workflow, to change the MQ Workflow password, to check whether a user is logged on, to list the MQ Workflow system groups, and to update the user settings in the MQ Workflow database.

Process-template actions
Process-template actions are used to create process instances.

Process-instance actions
Process-instance actions are used to delete, restart, resume, start, suspend, or terminate a process instance.

Process-instance notification actions
Process-instance notification actions are used to delete a process-instance notification.

Work-item actions
Work-item actions are used to check in or check out a work item, to delete, monitor, restart, start, terminate, transfer a work item, to get support tools, to start a support tool for a work item, or to update the attributes of a work item in the MQ Workflow database.

Work-item notification actions
Work-item notification actions are used to delete a work-item notification.

Replication actions
Replication actions are used to replicate per session the MQ Workflow user settings, the process instances, the process-instance notifications, the process templates, and the work items, and to replicate per work item the work-item input container.

Chapter 72. Mapping container data

This section describes how an MQ Workflow input container is mapped with values from a Lotus Notes document and vice versa.

Depending on the workflow model, the functions ExmnStartInstance and ExmnCheckInWorkitem must fill an MQ Workflow container. Also depending on the workflow model, the function ExmnCheckOutWorkitem retrieves container data for the work item that is to be processed in Lotus Notes.

If the process instance that is to be started or the work item that is to be checked in requires an input container, the leaves of the container are filled with the values of the Lotus Notes items with the same names as the data members in the MQ Workflow container. If a Lotus Notes item does not exist, the respective data member is not set in the container.

An example for fully qualified names is shown in Figure 12.

```
FlowMark data structure          Fully qualified item name
-----------------------          -------------------------
Adress[2]
    Name
       Firstname                 Adress[0].Name.Firstname
       Lastname                  Adress[0].Name.Lastname
    City                         Adress[0].City
    Age                          Adress[0].Age
                                 Adress[1].Name.Firstname
                                 Adress[1].Name.Lastname
                                 Adress[1].City
                                 Adress[1].Age
```

Figure 12. Example for fully qualified names

Chapter 73. General hints

1. If a parameter is specified as pointer, the Lotus Notes API functions do not check whether this pointer points to a valid address. So make sure you use valid pointers. An invalid pointer can cause a trap or undefined behavior.
2. Whenever a Lotus Notes server and a Lotus Notes database must be specified in a function call, this function reads from or writes to (that is, updates) the Lotus Notes database. If the Lotus Notes database is not on a server but local, specify a NULL value or an empty string for the Lotus Notes server parameter. If a Lotus Notes database is specified, the function updates the Lotus Notes database. When you invoke such a function, consider performance. Avoid updating the Lotus Notes database immediately unless it is necessary.

   If a Lotus Notes database is specified:
   a. Execute the MQ Workflow request
   b. Replicate the specified MQ Workflow object to the Lotus Notes database

   In this case, the return code EXMN_API_ERROR_REPLICATION_INTERNAL means that the MQ Workflow request completed successfully, but the replication was not successful. If the data must be replicated in this case, invoke the respective ExmnReplicate function separately.
3. If the error EXMN_API_ERROR_TIMEOUT occurs, retry the function. If the error occurs often, increase the time-out value by setting or modifying the environment variable APITimeOut in the MQ Workflow user or machine profile.

Chapter 74. General Notes actions

General Notes actions are:

ExmnChangePassword
Change MQ Workflow password.

ExmnIsLoggedOn
Check whether a user is logged on.

ExmnListDatabases
List MQ Workflow system groups.

ExmnLogoff
Log off from MQ Workflow.

ExmnLogon
Log on to MQ Workflow.

ExmnUpdateUserSettings
Update user settings in MQ Workflow.

Change the MQ Workflow password

Use the ExmnChangePassword function to change the password of an MQ Workflow user.

```
NAPIRET ExmnChangePassword(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszFMOldPassword,
                           char * pszFMNewPassword)
```

Figure 13. Change the MQ Workflow password

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID for which the password is to be changed.
pszFMDatabase	I	Provide the name of the MQ Workflow database.
pszFMOldPassword	I	Provide the current MQ Workflow user password.
pszFMNewPassword	I	Provide the new MQ Workflow user password.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_OLD_PASSWORD
The password specified as current password is not correct.

EXMN_API_ERROR_PASSWORD
Password does not comply with the FlowMark syntax rules for password.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark(R) internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

Check whether a user is logged on

Use the ExmnIsLoggedOn function to check whether an ExmnLogon call has already been issued.

```
NAPIRET ExmnIsLoggedOn(char * pszFMUserID,
                       char * pszFMDatabase)
```

Figure 14. Check logon

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID for which the logon check is to be done.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Return Codes:

EXMN_API_OK
The specified user is logged on.

EXM_API_ERROR_NOT_LOGGED_ON
No ExmnLogon function call was issued for the specified user.

List MQ Workflow system groups

Use the ExmnListDatabases function to list all MQ Workflow sytem group entries that are specified in the MQ Workflow configuration profile. To enable logon to MQ Workflow, the MQ Workflow system must be defined in the MQ Workflow user or machine profile.

```
NAPIRET ExmnListDatabases(long   lFMDBBufferSize,
                          long * plFMDBNeededBufferSize,
                          char * pszFMDatabaseList)
```

Figure 15. List MQ Workflow databases

Parameter	I/O	Description
lFMDBBufferSize	I	The size of the allocated memory of *pszFMDatabaseList*.
plFMDBNeededBufferSize	O	The parameter returns the actual buffer size for *pszFMDatabaseList*.
pszFMDatabaseList	O	Returns the list of all configured MQ Workflow system groups. The list is returned as null-terminated string where the individual system group names are separated with a dot (·). For example, if you specified the system groups EXMDB, EXMDB1, and EXMDB2 in your profile, the ExmnListDatabases function returns the string `EXMDB.EXMDB1.EXMDB2`.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_PROFILE
No system group entry has been found in the MQ Workflow user or machine profile.

EXMN_API_ERROR_BUFFER_TOO_SMALL
The buffer specified for *pszFMDatabaseList* is too small.

Reallocate the buffer with at least the size returned in *plFMDBNeededBufferSize* and call this function again.

Log off from MQ Workflow

Use the ExmnLogoff function to terminate an MQ Workflow session that was established with ExmnLogon.

```
NAPIRET ExmnLogoff(char * pszFMUserID,
                   char * pszFMDatabase)
```

Figure 16. Log off from MQ Workflow

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID that is to be logged off.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Log on to MQ Workflow

Use the ExmnLogon function to log on to MQ Workflow. This function also updates the user settings in the specified Lotus Notes database.

```
NAPIRET ExmnLogon(char * pszFMUserID,
                  char * pszFMPassword,
                  char * pszFMDatabase,
                  char * pszNotesServer,
                  char * pszFMUSNotesDB)
```

Figure 17. Log on to MQ Workflow

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMPassword	I	Provide the MQ Workflow user password.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMUSNotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMUSNotesDB	I	Provide the name of the Lotus Notes database where the MQ Workflow user settings are stored. If the data is not to be updated by this function, specify a null pointer or an empty string for *pszFMUSNotesDB*. If a valid Lotus Notes database name was specified, this function updates this database with the values specified in MQ Workflow.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_USERID_UNKNOWN
Unknown user ID.

EXMN_API_ERROR_PASSWORD
Incorrect password.

EXMN_API_ERROR_ALREADY_LOGGED_ON
User already logged on to the specified system group.

EXMN_API_ERROR_PROFILE
Profile data for system group not found.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_REPLICATION_INTERNAL
The replication of the user settings in the Lotus Notes database failed after a successful logon to FlowMark.

To update the database, invoke ExmnReplicateFMUserSettings.

Update user settings in MQ Workflow

Use the ExmnUpdateUserSettings function to update the MQ Workflow user settings with the values of the fields specified in the Lotus Notes database.

```
NAPIRET ExmnUpdateUserSettings(char * pszFMUserId,
                               char * pszFMDatabase,
                               char * pszNotesServer,
                               char * pszFMUSNotesDB)
```

Figure 18. Update MQ Workflow user settings

Parameter	I/O	Description
pszFMUserId	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database *pszFMDatabase* is located. If the database is local, specify a null pointer or an empty string.
pszFMUSNotesDB	I	Provide the name of the Lotus Notes database where the MQ Workflow user settings are stored. If the data is not to be updated by this function, specify a null pointer or an empty string for *pszFMUSNotesDB*. If a valid Lotus Notes Database is specified, this function updates the MQ Workflow database with the values specified in this Lotus Notes database.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Chapter 75. Process-template actions

Process-template actions are:

ExmnCreateInstance
Create process instance.

Create a process instance

Use the ExmnCreateInstance function to create a new process instance in MQ Workflow. This function uses a replicate function to update the Lotus Notes database with the new process instance if a Lotus Notes database is specified.

```
NAPIRET ExmnCreateInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszTemplateOID,
                           char * pszProcessInstanceName,
                           long   lInstanceOIDBufLen,
                           long * plInstanceOIDLen,
                           char * pszInstanceOID,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           long   lInstanceNIDBufLen,
                           char * pszInstanceNID)
```

Figure 19. Create a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszTemplateOID	I	Provide the identifier of the process-template object from which an instance is to be created. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszProcessInstanceName	I	Provide the name for the process instance that is to be created. If you do not provide an instance name, a default name is used (*ProcessTemplateNameXXX* where *XXX* is a representation of the object ID).
lInstanceOIDBufLen	I	Provide the size of the buffer allocated for *pszInstanceOID*. The recommended size is 200 bytes.

Parameter	I/O	Description
plInstanceOIDLen	O	This parameter contains the actual length of the process-instance identifier. If the ExmnCreateInstance function returns EXMN_API_ERROR_BUFFER_TOO_SMALL, reallocate the buffer for *pszInstanceOID* with the length returned in *plInstanceOIDLen*.
pszInstanceOID	O	This parameter contains the identifier of the newly created MQ Workflow process instance. The size of this buffer has to be specified in *lInstanceOIDBufLen*.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. Specify a null pointer or an empty string for *pszFMPINotesDB* if the Lotus Notes data is not to be updated by this function.
lInstanceNIDBufLen	I	Provide the size of the buffer allocated for *pszInstanceNID* (size should be at least 8 for the current release of Lotus Notes).
pszInstanceNID	O	Provide the NoteID of the new process-instance document.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
The parsed instance object references an undefined location.

EXMN_API_ERROR_BUFFER_TOO_SMALL
The buffer for *pszInstanceOID* was allocated too small. The instance is created and replicated to the Lotus Notes database but the identifier of the new instance could not be returned.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_NOT_UNIQUE
The name of the process instance is not unique.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_DOES_NOT_EXIST
The process template no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The process template is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The replication of the process instance in the Lotus Notes database failed after successful creation of the instance. To update the Lotus Notes database, invoke ExmnReplicateInstances.

Chapter 76. Process-instance actions

Process-instance actions are:

ExmnDeleteInstance
Delete process instance.

ExmnRestartInstance
Restart process instance.

ExmnResumeInstance
Resume process instance.

ExmnStartInstance
Start process instance.

ExmnSuspendInstance
Suspend process instance.

ExmnTerminateInstance
Terminate process instance.

Delete a process instance

Use the ExmnDeleteInstance function to delete a finished or terminated process instance from MQ Workflow.

Note: The authorization for deleting the process instance is checked by MQ Workflow.

```
NAPIRET ExmnDeleteInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszInstanceOID,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           char * pszInstanceNID)
```

Figure 20. Delete a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
pszInstanceOID	I	Provide the identifier of the process-instance object that is to be deleted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances if the Lotus Notes database is to be updated. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be deleted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The process instance was deleted in MQ Workflow but the deletion of the specified process instance in the Lotus Notes database failed. To update the Lotus Notes database, invoke the function ExmnReplicateInstances.

Restart a process instance

Use the ExmnRestartInstance function to restart an MQ Workflow process. Only top-level processes can be restarted.

```
NAPIRET ExmnRestartInstance(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszInstanceOID,
                            char * pszNotesServer,
                            char * pszFMPINotesDB,
                            char * pszInstanceNID)
```

Figure 21. Restart a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszInstanceOID	I	Provide the identifier of the process-instance object that is to be restarted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be restarted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The process instance is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed. To update the Lotus Notes database, use the function ExmnReplicateInstances.

Resume a process instance

Use the ExmnResumeInstance function to resume processing of a suspended MQ Workflow process instance.

```
NAPIRET ExmnResumeInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszInstanceOID,
                           short  sInDeep,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           char * pszInstanceNID)
```

Figure 22. Resume a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszInstanceOID	I	Provide the identifier of the process instance that is to be resumed. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
sInDeep	I	If *sInDeep* is set to true, processing of all subprocesses is also resumed.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be resumed.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed. To update the Lotus Notes database, use the function ExmnReplicateInstances.

Start a process instance

Use the ExmnStartInstance function to start an MQ Workflow process instance. The user who is currently logged on becomes the process starter. If the process requires an input container, the ExmnStartInstance function fills it with the information provided in the Lotus Notes document that is specified with *pszInstanceOID*. How the document fields are mapped to the MQ Workflow input container is described in "Chapter 72. Mapping container data" on page 695.

This function also initiates an update of the process status in the Lotus Notes database.

```
NAPIRET ExmnStartInstance(char * pszFMUserID,
                          char * pszFMDatabase,
                          char * pszInstanceOID,
                          char * pszNotesServer,
                          char * pszFMPINotesDB,
                          char * pszInstanceNID)
```

Figure 23. Start a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszInstanceOID	I	Provide the identifier to the process-instance object that is to be started. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. If an input container is required, you must provide the name of this database. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be started.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
The parsed container object references an undefined location or the container is empty.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_MEMBER_NOT_SET
Process requires input fields in the Lotus Notes database that were not specified.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The process instance is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed.

Suspend a process instance

Use the ExmnSuspendInstance function to suspend (that is, temporarily stop) an MQ Workflow process instance.

```
NAPIRET ExmnSuspendInstance(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszInstanceOID,
                            short  sInDeep,
                            char * pszNotesServer,
                            char * pszFMPINotesDB,
                            char * pszInstanceNID)
```

Figure 24. Suspend a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszInstanceOID	I	Provide the identifier of the process instance that is to be suspended. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
sInDeep	I	If *sInDeep* is set to true, processing of all subprocesses is also suspended.

Parameter	I/O	Description
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be suspended.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed.

Terminate a process instance

Use the ExmnTerminateInstance function to terminate an MQ Workflow process instance. The process instance is then marked as terminated on the MQ Workflow Runtime server.

```
NAPIRET ExmnTerminateInstance(char * pszFMUserID,
                              char * pszFMDatabase,
                              char * pszInstanceOID,
                              char * pszNotesServer,
                              char * pszFMPINotesDB,
                              char * pszInstanceNID)
```

Figure 25. Terminate a process instance

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszInstanceOID	I	Provide the identifier of the process-instance object that is to be terminated. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszInstanceNID	I	Provide the identifier of the process-instance object (in the Lotus Notes database) that is to be terminated.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed.

Chapter 77. Process-instance notification actions

Process-instance notification actions are:

ExmnDeletePINotification
Delete process-instance notification.

Delete process-instance notification

Use the ExmnDeletePINotification function to delete a finished process-instance notification from MQ Workflow.

```
NAPIRET ExmnDeletePINotification(char * pszFMUserID,
                                 char * pszFMDatabase,
                                 char * pszPINotificationOID,
                                 char * pszNotesServer,
                                 char * pszFMWINotesDB,
                                 char * pszPINotificationNID)
```

Figure 26. Delete process-instance notification

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszPINotificationOID	I	Provide the identifier of the process-instance notification object that is to be deleted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszPINotificationNID	I	Provide the identifier of the process-instance notification object (in the Lotus Notes database) that is to be deleted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified process instance in the Lotus Notes database failed.

Chapter 78. Work-item actions

Work-item actions are:

ExmnCheckInWorkitem
Check in work item.

ExmnCheckOutWorkitem
Check out work item.

ExmnDeleteWorkitem
Delete work item.

ExmnGetSupportTool
Get support tools.

ExmnManualExitWorkitem
Manual exit from work item.

ExmnRestartWorkitem
Restart work item.

ExmnStartSupportTool
Start support tool for a work item.

ExmnStartWorkitem
Start work item.

ExmnTerminateWorkitem
Terminate work item.

ExmnTransferWorkitem
Transfer work item.

ExmnUpdateWorkitem
Update the attributes of a work item.

Check in a work item

Use the ExmnCheckInWorkitem function to check in an MQ Workflow work item that was checked out for user processing. If container data is required, this function fills the container with the corresponding items from the Lotus Notes document. How the document fields are mapped to the MQ Workflow container is described in “Chapter 72. Mapping container data” on page 695.

```
NAPIRET ExmnCheckInWorkitem(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            long   sReturnCode,
                            char * pszNotesServer,
                            char * pszFMWINotesDB,
                            char * pszWorkitemNID)
```

Figure 27. Check in a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work-item object that is to be checked in. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
sReturnCode	I	Contains the return code that is set by the Lotus Notes application.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be checked in.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state, not checked in.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the deletion of the specified work item in the Lotus Notes database failed.

Check out a work item

Use the ExmnCheckOutWorkitem function to check out an MQ Workflow work item for user processing. This means that processing is not done by MQ Workflow's inherent tool-invocation mechanism. MQ Workflow assumes that processing is done by user-specific means.

When the user processing is complete, the work item must be checked in again for further workflow processing. If any container data is associated with the checked-out work item, the ExmnCheckOutWorkitem function creates the corresponding Lotus Notes items with the fully qualified names. For more information about container data refer to "Chapter 72. Mapping container data" on page 695.

```
NAPIRET ExmnCheckOutWorkitem(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszNotesServer,
                             char * pszFMWINotesDB,
                             char * pszWorkitemNID)
```

Figure 28. Check out a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work-item object that is to be checked out. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be checked out.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state, not checked out.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the deletion of the specified work item in the Lotus Notes database failed.

Delete a work item

Use the ExmnDeleteWorkitem function to delete a finished work item from MQ Workflow. Specify a Lotus Notes database to delete the work item also from the Lotus Notes database.

```
NAPIRET ExmnDeleteWorkitem(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszWorkItemOID,
                           char * pszNotesServer,
                           char * pszFMWINotesDB,
                           char * pszWorkitemNID)
```

Figure 29. Delete a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work-item object that is to be deleted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.

Parameter	I/O	Description
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be deleted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the deletion of the specified work item in the Lotus Notes database failed.

Get support tools for a work item

Use the ExmnGetSupportTools function to get a list of the support tools that were defined for a work item.

```
NAPIRET ExmnGetSupportTools(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            long   lSuppToolsBufLen,
                            long * plSuppToolsLen,
                            char * pszSupportToolList)
```

Figure 30. Get support tools for a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work-item object for which the list of support tools is requested. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
lSuppToolsBufLen	O	Provide the length of the buffer allocated for the list of support tools that is provided in *pszSupportToolList*.
plSuppToolsLen	O	This parameter contains the actual size of the string returned in *pszSupportToolList*.
pszSupportToolList	O	Returns the list of support tools that are defined for a work item. The list is returned as a string where the tool names are separated with a period (.). For example, if the list contains the items `Editor` and `Calculator`, the string is `Editor.Calculator`.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_BUFFER_TOO_SMALL
The buffer allocated for *pszSupportToolList* is too small. Allocate a buffer with the size returned in *plSuppToolsLen* and call this function again.

Manual exit from a work item

Use the ExmnManualExitWorkitem function to successfully complete a specific type of program activity, for example, a checklist of manual tasks.

```
NAPIRET ExmnManualExitWorkitem(char * pszFMUserID,
                               char * pszFMDatabase,
                               char * pszWorkItemOID,
                               char * pszNotesServer,
                               char * pszFMWINotesDB,
                               char * pszWorkitemNID)
```

Figure 31. Manual exit from a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work-item object that is to be exited manually. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be completed.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified work item in the Lotus Notes database failed.

Restart a work item

Use the ExmnRestartWorkitem function to set a running work item back to the ready state so that the ExmnStartWorkitem function can be called again.

```
NAPIRET ExmnRestartWorkitem(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            char * pszNotesServer,
                            char * pszFMWINotesDB,
                            char * pszWorkitemNID)
```

Figure 32. Restart a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work item that is to be restarted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be restarted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified work item in the Lotus Notes database failed.

Start a support tool

Use the ExmnStartSupportTool function to start a support tool that is associated with the work item.

```
NAPIRET ExmnStartSupportTool(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszSupportTool)
```

Figure 33. Start a support tool

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work item for which the support tool is to be started. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszSupportTool	I	Provide the name of the support tool that is to be started.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Start a work item

Use the ExmnStartWorkitem function to start the implementation tool or process assigned to the activity that is represented by the work item. The tool is started by MQ Workflow's tool-invocation mechanism. Specify a Lotus Notes database to replicate the new state of the work item to the Lotus Notes database.

```
NAPIRET ExmnStartWorkitem(char * pszFMUserID,
                          char * pszFMDatabase,
                          char * pszWorkItemOID,
                          char * pszNotesServer,
                          char * pszFMWINotesDB,
                          char * pszWorkitemNID)
```

Figure 34. Start a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work item that is to be started. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.

Parameter	I/O	Description
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be started.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified work item in the Lotus Notes database failed.

Terminate a work item

Use the ExmnTerminateWorkitem function to terminate an MQ Workflow work item.

```
NAPIRET ExmnTerminateWorkitem(char * pszFMUserID,
                              char * pszFMDatabase,
                              char * pszWorkItemOID,
                              char * pszNotesServer,
                              char * pszFMWINotesDB,
                              char * pszWorkitemNID)
```

Figure 35. Terminate a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work item that is to be terminated. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be terminated.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified work item in the Lotus Notes database failed.

Transfer a work item

Use the ExmnTransferWorkitem function to transfer a work item to another MQ Workflow user.

```
NAPIRET ExmnTransferWorkitem(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszUserID,
                             char * pszNotesServer,
                             char * pszFMWINotesDB,
                             char * pszWorkitemNID)
```

Figure 36. Transfer a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
pszWorkItemOID	I	Provide the identifier of the work item that is to be transferred. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszUserID	I	Provide the user ID of the user to whom the work item is to be transferred.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be transferred.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_USERID_UNKNOWN
Unknown user ID.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
A FlowMark internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

EXMN_API_ERROR_REPLICATION_INTERNAL
The method completed successfully but the replication of the specified work item in the Lotus Notes database failed.

Update a work item

Use the ExmnUpdateWorkitem function to update the MQ Workflow work item with the values of the fields specified in the Lotus Notes database.

```
NAPIRET ExmnUpdateWorkitem(char * pszFMUserId,
                           char * pszFMDatabase,
                           char * pszWorkitemOId,
                           char * pszNotesServer,
                           char * pszFMWINotesDB,
                           char * pszWorkitemNID)
```

Figure 37. Update a work item in MQ Workflow

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWorkItemOID	I	Provide the identifier of the work item that is to be updated. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database where the MQ Workflow work item is stored.

Parameter	I/O	Description
pszWorkitemNID	I	Provide the identifier of the work-item object (in the Lotus Notes database) that is to be updated.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Chapter 79. Work-item notification actions

Work-item notification action:

ExmnDeleteWINotification
Delete a work-item notification.

Delete a notification for a work item

Use the ExmnDeleteWINotification function to delete a finished work-item notification object from MQ Workflow.

```
NAPIRET ExmnDeleteWINotification(char * pszFMUserID,
                                 char * pszFMDatabase,
                                 char * pszWINotificationOID,
                                 char * pszNotesServer,
                                 char * pszFMWINotesDB,
                                 char * pszWINotificationNID)
```

Figure 38. Delete a notification for a work item

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
pszWINotificationOID	I	Provide the identifier of the work-item notification object that is to be deleted. **Note:** You can obtain this value from the _ExmDocObjectId item in the Lotus Notes document.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items. If the Lotus Notes database is not to be updated, specify a null pointer or an empty string.
pszWINotificationNID	I	Provide the identifier of the work-item notification object (in the Lotus Notes database) that is to be deleted.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

EXMN_API_ERROR_DOES_NOT_EXIST
The work item no longer exists.

EXMN_API_ERROR_NOT_AUTHORIZED
Not authorized to use the method.

EXMN_API_ERROR_WRONG_STATE
Not possible in this state.

EXMN_API_ERROR_TIMEOUT
Timeout has occurred.

EXMN_API_ERROR_SERVER
The server is not available, or is unknown.

EXMN_API_ERROR_INTERNAL
An MQ Workflow internal error has occurred. Contact your IBM representative.

EXMN_API_ERROR_RESOURCE
An internal resource problem.

EXMN_API_ERROR_MESSAGE_FORMAT
An internal message format error.

EXMN_API_ERROR_EMPTY
The work-item object is still empty.

Chapter 80. Replication actions

The replication functions read the specified MQ Workflow objects from the MQ Workflow database and synchronize the Lotus Notes database. If return code EXMN_API_OK is issued, this means that the databases are synchronized. This does not always imply an update of the Lotus Notes database.

Replication actions are:

ExmnReplicateFMUserSettings
Replicate MQ Workflow user settings per session.

ExmnReplicateInstances
Replicate process instances per session.

ExmnReplicatePINotification
Replicate process-instance notifications per session.

ExmnReplicateTemplates
Replicate process templates per session.

ExmnReplicateWINotification
Replicate work-item notifications per session.

ExmnReplicateWorkitems
Replicate work items per session.

Replicate MQ Workflow user settings per session

Use the ExmnReplicateFMUserSettings function to replicate the user settings to the Lotus Notes database.

```
ExmnReplicateFMUserSettings(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszNotesServer,
                            char * pszFMUSNotesDB)
```

Figure 39. Replicate FMUserSettings per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMUSNotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMUSNotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow user settings.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Replicate process instances per session

Use the ExmnReplicateInstances function to replicate to the Lotus Notes database all MQ Workflow process-instance objects that meet the specified filter criteria.

```
NAPIRET ExmnReplicateInstances(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszInstanceOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMPINotesDB,
                               char *       pszInstanceNIDList)
```

Figure 40. Replicate process instances per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
fReplicateFlag	I	Set the filter flag. This can be: **ReplAll** Replicate all process instances to the specified Lotus Notes database. *pszInstanceOIDList* and *pszLastReplicationDate* are ignored. **ReplObjectList** Replicate only those process instances of which the object IDs are listed in *pszInstanceOIDList* and *pszInstanceNIDList*. **ReplLastDate** Reserved.
pszInstanceOIDList	I	Provide a list of the MQ Workflow identifiers (separated by periods) of the objects to be refreshed if *fReplicateFlag* is set to ReplObjectList.
pszLastReplicationDate	I	Specify NULL or an empty string.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process instances.
pszInstanceNIDList	I	Provide a list of the Lotus Notes identifiers of the objects that are to be refreshed if *fReplicateFlag* is set to ReplObjectList.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR
pszWorkItemOID references an undefined location or contains an incorrect value.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Replicate process-instance notifications per session

Use the ExmnReplicatePINotification function to replicate to the Lotus Notes database all MQ Workflow process-instance notification objects that meet the specified filter criteria.

```
NAPIRET ExmnReplicatePINotification(char *       pszFMUserID,
                                    char *       pszFMDatabase,
                                    ExmnReplFlag fReplicateFlag,
                                    char *       pszPINotificationOIDList,
                                    char *       pszLastReplicationDate,
                                    char *       pszNotesServer,
                                    char *       pszFMWINotesDB,
                                    char *       pszPINotificationNIDList)
```

Figure 41. Replicate process-instance notifications per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
fReplicateFlag	I	Set the filter flag. This can be: **ReplAll** Replicate all process instances notifications to the specified Lotus Notes database. *pszPINotificationOIDList* and *pszLastReplicationDate* are ignored. **ReplObjectList** Replicate only those process instance notifications of which the object IDs are listed in *pszPINotificationOIDList* and *pszPINotificationNIDList*. **ReplLastDate** Reserved.
pszPINotificationOIDList	I	Provide a list of the identifiers (separated by periods) of the objects to be refreshed if *fReplicateFlag* is set to ReplObjectList.
pszLastReplicationDate	I	Specify NULL or an empty string.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.

Parameter	I/O	Description
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process-instance notifications.
pszInstanceNIDList	I	Provide a list of the Lotus Notes identifiers of the objects that are to be refreshed if *fReplicateFlag* is set to ReplObjectList.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Replicate process templates per session

Use the ExmnReplicateTemplates function to replicate to the Lotus Notes database all process templates that meet the specified filter criteria.

```
NAPIRET ExmnReplicateTemplates(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszTemplateOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMPTNotesDB,
                               char *       pszTemplateNIDList)
```

Figure 42. Replicate process templates per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.

Parameter	I/O	Description
fReplicateFlag	I	Set the filter flag. This can be: **ReplAll** Replicate all process templates to the specified Lotus Notes database. *pszTemplateOIDList* and *pszLastReplicationDate* are ignored. **ReplObjectList** Replicate only those process templates of which the object IDs are listed in *pszTemplateOIDList* and *pszTemplateNIDList*. **ReplLastDate** Reserved.
pszTemplateOIDList	I	Provide a list of identifiers (separated by periods) of the objects to be refreshed if *fReplicateFlag* is set to ReplObjectList.
pszLastReplicationDate	I	Specify NULL or an empty string.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMPTNotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMPTNotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow process templates.
pszTemplateNIDList	I	Provide a list of the Lotus Notes identifiers of the objects that are to be refreshed if *fReplicateFlag* is set to ReplObjectList.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Replicate work-item notifications per session

Use the ExmnReplicateWINotification function to replicate to the Lotus Notes database all work-item notification objects that meet the specified filter criteria.

```
NAPIRET ExmnReplicateWINotification(char *       pszFMUserID,
                                   char *       pszFMDatabase,
                                   ExmnReplFlag fReplicateFlag,
                                   char *       pszWINotificationOIDList,
                                   char *       pszLastReplicationDate,
                                   char *       pszNotesServer,
                                   char *       pszFMWINotesDB,
                                   char *       pszWINotificationNIDList)
```

Figure 43. Replicate work-item notifications per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
fReplicateFlag	I	Set the filter flag. This can be: **ReplAll** Replicate all work-item notifications to the specified Lotus Notes database. *pszWINotificationOIDList* and *pszLastReplicationDate* are ignored. **ReplObjectList** Replicate only those work-item notifications of which the object IDs are listed in *pszWINotificationOIDList* and *pszWINotificationNIDList*. **ReplLastDate** Reserved.
pszWINotificationOIDList	I	Provide a list of the objects (separated by periods) to be refreshed if *fReplicateFlag* is set to ReplObjectList.
pszLastReplicationDate	I	Specify NULL or an empty string.
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items.
pszWINotificationNIDList	I	Provide a list of the Lotus Notes identifiers of the objects that are to be refreshed if *fReplicateFlag* is set to ReplObjectList.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Replicate work items per session

Use the ExmnReplicateWorkitems function to replicate to the Lotus Notes database all work-item objects that meet the specified filter criteria.

```
NAPIRET ExmnReplicateWorkitems(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszWorkitemOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMWINotesDB,
                               char *       pszWorkitemNIDList)
```

Figure 44. Replicate work items per session

Parameter	I/O	Description
pszFMUserID	I	Provide the MQ Workflow user ID.
pszFMDatabase	I	Provide the name of the MQ Workflow system group.
fReplicateFlag	I	Set the filter flag. This can be: **ReplAll** Replicate all work items to the specified Lotus Notes database. *pszWorkitemOIDList* and *pszLastReplicationDate* are ignored. **ReplObjectList** Replicate only those work items of which the object IDs are listed in *pszWorkitemOIDList* and *pszWorkitemNIDList*. **ReplLastDate** Reserved.
pszWorkitemOIDList	I	Provide a list of the objects (separated by periods) to be refreshed if *fReplicateFlag* is set to ReplObjectList.
pszLastReplicationDate	I	Specify NULL or an empty string.

Parameter	I/O	Description
pszNotesServer	I	Provide the name of the Lotus Notes server where the database specified in *pszFMWINotesDB* is located. If the database is local, specify a null pointer or an empty string.
pszFMWINotesDB	I	Provide the name of the Lotus Notes database for the MQ Workflow work items.
pszWorkitemNIDList	I	Provide a list of the Lotus Notes identifiers of the objects that are to be refreshed if *fReplicateFlag* is set to ReplObjectList.

Return Codes:

EXMN_API_OK
The method completed successfully.

EXMN_API_ERROR_NOT_LOGGED_ON
There is no session established for the specified user.

Chapter 81. Fields used by the Client for Lotus Notes

The tables in the following sections describe the fields used in the forms supplied with the MQ Workflow Lotus Notes client. The value in the **Set** column in those tables shows if this item is set immediately (I) when queried or only after a refresh (R).

Application settings

Table 4. Application settings

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: "ExmAs"
_ExmDocType	I	Text	The field is set to: "ExmAs"
Implementation-specific fields			
_ImplSuppressNotLoggedOn		Text	Possible values for this field are: **Value** **Explanation** **0** No **1** Yes
_ImplOriginalForm		Text	

User settings

Table 5. User settings

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: "ExmUs"
_ExmDocType	I	Text	The field is set to: "ExmUs"
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the logged-on user. The length of the user ID can be 1–32 characters.

Table 5. User settings (continued)

Field name	Set	Data type	Description and contents of the field
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–32 characters.
_ExmPersonId	I	Text	This field contains the person ID of the logged-on user. The length of the person ID can be 0–32 characters.
_ExmFirstName	I	Text	This field contains the first name of the logged-on user. The length of the name can be 0–32 characters.
_ExmMiddleName	I	Text	This field contains the middle name of the logged-on user. The length of the name can be 0–32 characters.
_ExmLastName	I	Text	This field contains the last name of the logged-on user. The length of the name can be 0–32 characters.
_ExmPhone1	I	Text	This field contains the telephone number of the logged-on user. The field can contain 0–32 characters.
_ExmPhone2	I	Text	The alternative telephone number of the logged-on user. This field can contain 0–32 characters.
_ExmLevel	I	Text	This field contains the level for the logged-on user. This can be a number from 0 to 9.

Table 5. User settings (continued)

Field name	Set	Data type	Description and contents of the field
_ExmAbsent	I	Text	This field indicates whether the logged-on user is declared absent. It can be modified in MQ Workflow via the update API function. Possible values are: **Value** **Explanation** **0** Not absent **1** Absent
_ExmDeleteFinished	I	Text	This field indicates whether finished items are to be deleted from the MQ Workflow database. It can be modified in MQ Workflow via the update API function. Possible values are: **Value** **Explanation** **0** No **1** Yes
_ExmManager	I	Text	This field contains the manager information for the logged-on user. It can contain 0–32 characters.
_ExmOrganization	I	Text	This field shows the organization to which the logged-on user belongs. The field can contain 0–32 characters.
_ExmOrgManager	I	Text	This field indicates whether the logged-on user has the role manager. Possible values are: **Value** **Explanation** **0** No **1** Yes
_ExmRolesAssigned	I	Text	This field contains the roles the logged-on user has. The field is a multi-value field. It can contain 0–32 characters.

Table 5. User settings (continued)

Field name	Set	Data type	Description and contents of the field
_ExmRolesCoordinating	I	Text	This field contains the roles the logged-on user coordinates. The field is a multi-value field. It can contain 0–32 characters.
_ExmAuthUsers	I	Text	This field contains a list of persons to whose work items the logged-on user has access. This field is a multi-value field. It can contain 0–32 characters.
_ExmAuthAllUsers	I	Text	This field indicates whether the logged-on user is authorized to access other people's activities. Possible values are: **Value** **Explanation** **0** No **1** Yes
_ExmAuthAccessPermitted	I	Text	This field contains a list of persons who have access to the logged-on user's work items. This field is a multi-value field. It can contain 0–32 characters.
_ExmAuthStaff	I	Text	This field indicates whether the logged-on user is authorized to define staff. Possible values are: **Value** **Explanation** **0** No **1** Yes

Table 5. User settings (continued)

Field name	Set	Data type	Description and contents of the field
_ExmAuthOther	I	Text	This field indicates whether the logged-on user is authorized to model processes. Possible values are: **Value** **Explanation** **0** No **1** Yes
_ExmAuthCategories	I	Text	This field contains the categories for which the logged-on user has basic authorization. The field is a multi-value field. It can contain 0–32 characters.
_ExmAuthCategoriesAdmin	I	Text	This field contains the categories for which the logged-on user has administrator authorization. The field is a multi-value field. It can contain 0–32 characters.
_ExmAuthAllCategoriesAdmin	I	Text	This field indicates whether the logged-on user is authorized to manage process instances in all categories. Possible values are: **Value** **Explanation** **0** No **1** Yes
_ExmOwnSubstitute	I	Text	This field contains information about the user who substitutes for the logged-on user. The field can contain 0–32 characters.

Table 5. User settings (continued)

Field name	Set	Data type	Description and contents of the field
_ExmUsersSubstituted	I	Text	This field shows the persons for whom the logged-on user substitutes. The field is a multi-value field and can be modified in MQ Workflow via the update API function. It can contain 0–32 characters.
_ExmDescription	I	Text	This field contains a description of the logged-on user. The field can contain 0–1 024 characters.
Implementation-specific fields			
_ImplConnectionState		Text	Possible values are: **Value** **Explanation** **'0'** Disconnected **'1'** Connected
_ImplPassword		Text	This field is encrypted. It can contain 0–32 characters.
_ImplPasswordVerify		Text	This field is encrypted. It can contain 0–32 characters.
_ImplOriginalForm		Text	
_ImplDatabaseList		Text	This is a multi-value field.
_ImplAbsentDisp		Keyword	
_ImplDeleteFinishedDisp		Keyword	

Process instance

Table 6. Process template

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: “ExmPi”
_ExmDocType	I	Text	The field is set to: “ExmPi”
_ExmDocObjectId	I	Text	This field can contain 1–n characters.

Table 6. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–8 characters.
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the person. The length of the user ID can be 1–32 characters.
_ExmName	I	Text	This field contains the name of the process instance. The field can contain 1–63 characters.
_ExmDescription	I	Text	This field contains the description of the process instance. The field can contain 0–254 characters.
_ExmCategory	I	Text	This field contains the category of the process instance. The field can contain 0–32 characters.
_ExmState	I	Text	This field indicates the status of the process instance. Possible values are: **Value** **Explanation** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended
_ExmParentName	I	Text	This field contains the name of the parent process of the process instance associated with this notification. This field can contain 0–63 characters.

Table 6. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmTopLevelName	I	Text	This field contains the name of the top-level process of the process instance associated with this notification. This field can contain 1–63 characters.
_ExmAudit	R	Text	This field indicates whether an audit log is to be written for this process instance. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmTerminateOnError	R	Text	This field indicates whether the process instance is to be terminated if an error occurs during the evaluation of an exit or transition condition. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmInContainerNeeded	I	Text	This field indicates whether this process instance requires input data. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 6. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmInContainerName	R	Text	This field contains the name of the input-container data structure of the process instance. The field can contain 1–32 characters.
_ExmOutContainerName	R	Text	This field contains the name of the output-container data structure of the process instance. The field can contain 1–32 characters.
_ExmProcessAdmin	R	Text	This field contains the user ID of the process administrator of this process instance. The field can contain 0–32 characters.
_ExmStarter	R	Text	This field contains the user ID of the user who started this process instance. The field can contain 0–32 characters.
_ExmRole	R	Text	This field contains the role that can be used for staff assignment. The field can contain 0–32 characters.
_ExmOrganization	R	Text	This field contains the organization that can be used for staff assignment. The field can contain 0–32 characters.
_ExmDocumentation	R	Text	This field contains the documentation of the process instance. The field can contain 0–4 096 characters.
_ExmStartTime	R	Time	This field contains the time and date when the process instance was started.

Table 6. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmNotificationTime	R	Time	This field contains the time and date when a notification for the process instance is to occur.
_ExmEndTime	I	Time	This field contains the time and date when the process instance was completed.
Internal fields			
_ExmReplicationDate	I	Text	
Implementation-specific fields			
_ImplOriginalForm		Text	

Process-instance notification

Table 7. Process-instance notification

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: "ExmWiPin"
_ExmDocType	I	Text	The field is set to: "ExmWiPin"
_ExmDocObjectId	I	Text	This field can contain 1–n characters.
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–8 characters.
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the person. The length of the user ID can be 1–32 characters.

Table 7. Process-instance notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmExpired	I	Text	This field indicates whether the maximum duration time of the process instance has expired. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmName	I	Text	This field contains the name of the process instance. The field can contain 1–63 characters.
_ExmDescription	I	Text	This field contains the description of the process instance. The field can contain 0–254 characters.
_ExmCategory	I	Text	This field contains the category of the process the process-instance notification is a part of. The field can contain 0–32 characters.
_ExmState	I	Text	This field indicates the status of the process instance. Possible values are: **Value** **Explanation** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended

Table 7. Process-instance notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmParentName	I	Text	This field contains the name of the parent process of the process instance associated with this notification. This field can contain 0–63 characters.
_ExmTopLevelName	I	Text	This field contains the name of the top-level process of the process instance associated with this notification. This field can contain 1–63 characters.
_ExmAudit	I	Text	This field indicates whether an audit log is to be written for the process instance associated with this notification. This field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmTerminateOnError	I	Text	This field indicates whether the process instance associated with this notification is terminated if an error occurs during the evaluation of an exit or transition condition. This field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 7. Process-instance notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmInContainerNeeded	I	Text	This field indicates whether the process instance associated with this notification requires input data. This field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmInContainerName	I	Text	This field contains the name of the input-container data structure of the process instance. The field can contain 1–32 characters.
_ExmOutContainerName	I	Text	This field contains the name of the output-container data structure of the process instance. The field can contain 1–32 characters.
_ExmProcessAdmin	I	Text	This field contains the user ID of the process administrator of the process instance. The field can contain 1–32 characters.
_ExmStarter	I	Text	This field contains the user ID of the user who started the process instance associated with this notification. This field can contain 1–32 characters.
_ExmRole	I	Text	This field contains the role that can be used for the staff assignment of the process instance associated with this notification. This field can contain 0–32 characters.

Table 7. Process-instance notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmOrganization	I	Text	This field contains the organization that can be used for the staff assignment of the process instance associated with this notification. This field can contain 0–32 characters.
_ExmDocumentation	I	Text	This field contains the documentation of the process instance. The field can contain 0–4 096 characters.
_ExmStartTime	I	Time	This field contains the time and date when the process instance was started.
_ExmNotificationTime	I	Time	This field contains the time and date when a notification is to occur for the process instance associated with this notification.
_ExmEndTime	I	Time	This field contains the time and date when the item was completed.
_ExmProcessInstanceName	I	Text	This field contains the name of the process instance associated with this notification. The field can contain 1–54 characters.
Internal fields			
_ExmReplicationDate	I	Text	
Implementation-specific fields			
_ImplOriginalForm		Text	

Process template

Table 8. Process template

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			

Table 8. Process template (continued)

Field name	Set	Data type	Description and contents of the field
FORM		Text	The field is set to: "ExmPt"
_ExmDocType	I	Text	The field is set to: "ExmPt"
_ExmDocObjectId	I	Text	This field can contain 1–n characters.
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–8 characters.
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the person. The length of the user ID can be 1–32 characters.
_ExmName	I	Text	This field contains the name of the process template. The field can contain 1–32 characters.
_ExmDescription	I	Text	This field contains the description of the process template. The field can contain 0–1 024 characters.
_ExmCategory	I	Text	This field contains the category of the process template. The field can contain 0–32 characters.
_ExmAudit	R	Text	This field indicates whether an audit log is to be written for process instances created from this process template. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 8. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmTerminateOnError	R	Text	This field indicates whether process instances created from this template are to be terminated if an error occurs during the evaluation of an exit or transition condition. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmInContainerNeeded	I	Text	This field indicates whether process instances created from this template require input data. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmInContainerName	R	Text	This field contains the name of the input-container data structure of the process. The field can contain 1–32 characters.
_ExmOutContainerName	R	Text	This field contains the name of the output-container data structure of the process. The field can contain 1–32 characters.
_ExmProcessAdmin	R	Text	This field contains the user ID of the process administrator of the process instances created from this template. The field can contain 0–32 characters.

Table 8. Process template (continued)

Field name	Set	Data type	Description and contents of the field
_ExmRole	R	Text	This field contains the role that can be used for staff assignment. The field can contain 0–32 characters.
_ExmOrganization	R	Text	This field contains the organization that can be used for staff assignment. The field can contain 0–32 characters.
_ExmDocumentation	R	Text	This field contains the documentation of the process template. The field can contain 0–4 096 characters.
Internal fields			
_ExmReplicationDate	I	Text	
Implementation-specific fields			
_ImplOriginalForm		Text	

Work item

Table 9. Work item

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: “ExmWi”
_ExmDocType	I	Text	The field is set to: “ExmWi”
_ExmDocObjectId	I	Text	This field can contain 1–n characters.
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–8 characters.

Table 9. Work item (continued)

Field name	Set	Data type	Description and contents of the field
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the person. The length of the user ID can be 1–32 characters.
_ExmName	I	Text	This field contains the name of the work item. The field can contain 1–32 characters.
_ExmDescription	I	Text	This field contains the description of the work item. It can be modified in MQ Workflow via the update API function and can contain 0–1 024 characters.
_ExmState	I	Text	This field indicates the status of the work item. Possible values are: **Value** **Explanation** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended **'32'** Disabled
_ExmEscalated	I	Text	This field indicates whether the work item is escalated (that is, whether there is a notification for the work item). This field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 9. Work item (continued)

Field name	Set	Data type	Description and contents of the field
_ExmImplementation	I	Text	This field contains the name of the tool or process implementing the work item. The field can contain 1–32 characters.
_ExmReceivedAs	I	Text	This field indicates why the work item was placed on the worklist. The field can have the following values: **Value** **Explanation** **'0'** Undefined **'1'** Normal **'2'** Substitute **'3'** ProcessAdministrator **'4'** SystemAdministrator **'5'** ByTransfer
_ExmProcessInstanceName	I	Text	This field contains the name of the process instance the work item is a part of. The field can contain 1–54 characters.
_ExmProcessCategory	I	Text	This field contains the category of the process the work item is a part of. The field can contain 0–32 characters.
_ExmManualStartMode	R	Text	This field indicates whether the start mode of the item is set to manual. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 9. Work item (continued)

Field name	Set	Data type	Description and contents of the field
_ExmStartCondition	R	Text	This field contains the start condition associated with the work item. The field can have the following values: **Value** **Explanation** **'ANY'** One of the incoming connectors must evaluate to true. **'ALL'** All incoming connectors must evaluate to true.
_ExmManualExitMode	R	Text	This field indicates whether the exit mode of the work item is set to manual. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmExitCondition	R	Text	This field contains the end condition of the work item. The field can contain up to 4 KB of data.
_ExmStaff	R	Text	This field contains the user IDs of all persons associated with the activity that is represented by the work item. This is a multi-value field and can contain 0–32 characters.
_ExmProcessAdmin	R	Text	This field contains the user ID of the process administrator of the process of which the work item is a part. The field can contain 1–32 characters.

Table 9. Work item (continued)

Field name	Set	Data type	Description and contents of the field
_ExmPriority	I	Text	This field indicates the priority of the work item. The field can contain a number from 0 to 9.
_ExmDocumentation	R	Text	This field contains the documentation of the work item. This field can contain 0–4 096 characters.
_ExmStartTime	I	Time	This field contains the time and date when the activity was started (that is, when the work item became ready).
_ExmNotificationTime	R	Time	This field contains the time and date when a notification for the work item is to occur.
_ExmEndTime	R	Time	This field contains the time and date when the work item was completed.
Internal fields			
_ExmReplicationDate	I	Text	
Implementation-specific fields			
_ImplOriginalForm		Text	
_ImplNewOwner		Text	
_ImplTool		Text	
_ImplToolsList		Text	This is a multi-value field.
_ImplRc		Text	

Work-item notification

Table 10. Work-item notification

Field name	Set	Data type	Description and contents of the field
Lotus Notes API fields			
FORM		Text	The field is set to: “ExmWiWin”
_ExmDocType	I	Text	The field is set to: “ExmWiWin”

Table 10. Work-item notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmDocObjectId	I	Text	This field can contain 1–n characters.
_ExmDatabase	I	Text	This field contains the name of the MQ Workflow system group. The length of the name can be 1–8 characters.
_ExmUserId	I	Text	This field contains the MQ Workflow user ID of the person. The length of the user ID can be 1–32 characters.
_ExmExpired	I	Text	This field indicates whether the maximum duration time for the work item has expired. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmOverdue	I	Text	This field indicates whether starting the work item is overdue. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmEscalation	I	Text	This field indicates if the escalation is the first or the second escalation. The field can have the following values: **Value** **Explanation** **'1'** First escalation **'2'** Second escalation

Table 10. Work-item notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmName	I	Text	This field contains the name of the work item. The field can contain 1–32 characters.
_ExmDescription	I	Text	This field contains the description of the work item. The field can contain 0–1 024 characters.
_ExmState	I	Text	This field indicates the status of the work item. Possible values are: **Value** **Explanation** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended **'32'** Disabled
_ExmEscalated	I	Text	This field indicates whether the work item is escalated. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmImplementation	I	Text	This field contains the name of the tool or process implementing the work item. The field can contain 1–32 characters.

Table 10. Work-item notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmReceivedAs	I	Text	This field indicates why the work-item notification was placed on the worklist. This field can have the following values: **Value** **Explanation** **'0'** Undefined **'1'** Normal **'2'** Substitute **'3'** ProcessAdministrator **'4'** SystemAdministrator **'5'** ByTransfer
_ExmProcessInstanceName	I	Text	This field contains the name of the process instance the work-item notification is a part of. The field can contain 1–54 characters.
_ExmProcessCategory	I	Text	This field contains the category of the process the work-item notification is a part of. This field can contain 0–32 characters.
_ExmManualStartMode	R	Text	This field indicates whether the start mode of the item is set to manual. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes

Table 10. Work-item notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmStartCondition	R	Text	This field contains the start condition associated with the work item. The field can have the following values: **Value** **Explanation** **'ANY'** One of the incoming connectors must evaluate to true. **'ALL'** All incoming connectors must evaluate to true.
_ExmManualExitMode	R	Text	This field indicates whether the exit mode of the work item is set to manual. The field can have the following values: **Value** **Explanation** **'0'** No **'1'** Yes
_ExmExitCondition	R	Text	This field contains the end condition of the work item. The field can contain up to 4 KB of data.
_ExmStaff	R	Text	This field contains the user IDs of all persons associated with the activity that is represented by the work item. This is a multi-value field and can contain 0–32 characters.
_ExmPriority	I	Text	This field indicates the priority of the work item. The field can contain a number from 0 to 9.

Table 10. Work-item notification (continued)

Field name	Set	Data type	Description and contents of the field
_ExmDocumentation	R	Text	This field contains the documentation of the work item. The field can contain 0–4 096 characters.
_ExmStartTime	R	Time	This field contains the time and date when the activity was started (that is, when the work item became ready).
_ExmNotificationTime	R	Time	This field contains the time and date when a second notification for the work item is to occur.
_ExmEndTime	R	Time	This field contains the time and date when the work-item notification was completed.
Implementation-specific fields			
_ImplOriginalForm		Text	

Part 10. Appendixes

Appendix A. 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.

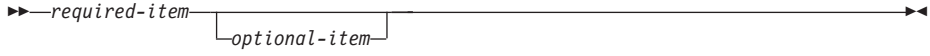
  The —▸ symbol indicates that the statement syntax is continued on the next line.

  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.

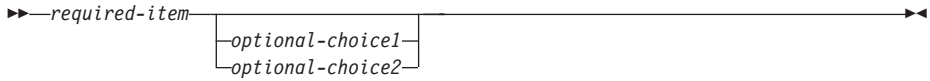
- Optional items appear below (or above) the main path.



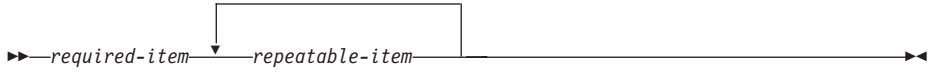
- If you can choose from one or more items, they appear vertically, in a stack.

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

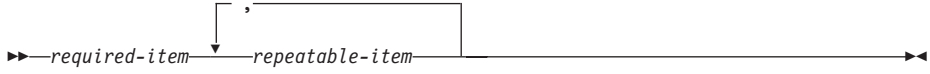
  If choosing one of the items is optional, the entire stack appears below the main path.



- 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.

Appendix B. FlowMark Version 2 compatibility mode

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 functions/methods with the MQ Workflow Version 3 functions/methods in all your applications.

The following languages and compilers are supported in compatibility mode:

- The C-language API
  - For AIX and the IBM C Set++ Version 3.1.4
  - For OS/2 and IBM VisualAge for C++ 3.0
  - For Windows NT or Windows 95 and IBM VisualAge for C++ 3.5 or Microsoft Visual C++ 5.0
- The C++ language API
  - For AIX and the IBM C Set++ Version 3.1.4
  - For HP-UX and the HP aC++ Compiler S700 Version A.01.15.01
  - For OS/2 and IBM VisualAge for C++ 3.0
  - For Windows NT or Windows 95 and IBM VisualAge for C++ 3.5 or Microsoft Visual C++ 5.0
- The REXX API
  - For OS/2 and the compiler provided with the operating system
- The VisualBasic API
  - For Windows NT or Windows 95 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 11. FlowMark Version 2 Compatibility APIs on AIX

Language	**AIX**		
	header	**lib**	**DLL**
C	exmajapc.h	fmcjdapc.lib	libfmcjdapc.a

Table 11. FlowMark Version 2 Compatibility APIs on AIX (continued)

Language	AIX		
	header	**lib**	**DLL**
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	libfmcjdcom.a libfmcjdcbr.a libfmcjdrun.a libfmcjdcon.a

Table 12. 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 13. FlowMark Version 2 Compatibility APIs on OS/2

Language	OS/2		
	header	**lib**	**DLL**
C	exmpjapc.h	fmcjdapc.lib	fmcjdapc.dll
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	fmcjdcom.dll fmcjdcbr.dll fmcjdrun.dll fmcjdcon.dll
Rexx			fmcjdrex.dll

Table 14. FlowMark Version 2 Compatibility APIs on Windows NT/95

Language	Windows NT/95		
	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 321).

- 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 here* is used. Present here forces any other present session for the user logoff.

  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.

  Present here especially forces any pending present session logoff. If you rely on the *already logged on* return code, set the *present* mode in the profile. *Default* allows for multiple parallel sessions.

  You can use the configuration utility fmczutil to set or erase profile values; the key is V2_SESSION_MODE; the values are DEFAULT, PRESENT, or PRESENTHERE.
- 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.
- 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.
- 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.
- 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.
- 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 785 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 785 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 REXX 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 785 for possible increased authorization requirements.

If you want to run an existing FlowMark Version 2 REXX application program in an MQ Workflow Version 3 environment:

- Load the MQ Workflow REXX functions contained in fmcjdrex.dll
- Ensure that the MQ Workflow Version 3 dynamic link library fmcjdrex.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 785 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 14. Compiling and linking" on page 137
- 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 error codes:

e. Change all EXM_API_ERROR occurrences to FMC_ERROR.

   After this step, 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 Role() method calls to RoleName().
   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.

- **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.

- **ExmProcessInstanceNotification**

  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 C. 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, or 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 registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

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

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

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 translated the process model into a process template, the process template can be executed over and over again. *Workflow model* and *process definition* are synonyms.

process monitor API. An application programming interface that allows applications to implement the functions of a process instance monitor.

process-relevant data. Data that is used to control the sequence of activities in a process instance.

process status. The status of a process instance.

process template. A fixed form of a process model from which process instances can be created. It is the translated form in MQ Workflow Runtime. See also *process instance*.

process template list. A set of process templates that have been selected and sorted according to user-defined criteria.

program. A computer-based application that serves as the implementation of a program activity or as a support tool. Program activities reference executable programs using the logical

names associated with the programs in MQ Workflow program registrations. See also *program registration*.

program activity. An activity that is executed by a registered program. Starting this activity invokes the program. Contrast with *process activity*.

program execution agent. 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 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 and notifications assigned to a user and retrieved from a workflow management system.

worklist view. List of work items selected from a work item set of a user according to filter criteria which are an attribute of a worklist. It can be sorted according to sort criteria if specified for this worklist.

Bibliography

To order any of the following publications, contact your IBM representative or IBM branch office.

MQSeries Workflow 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.
- *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.

Index

D

E

F

G

H

I

Q

R

S

T

U

V

W

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

IBM MQSeries Workflow
Programming Guide
Version 3.2

Publication No. SH12-6291-03

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
SH12-6291-03

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
Postfach 1380
71003 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SH12-6291-03

Cut or Fold
Along Line

Part Number: 22L4285
Program Number: 5697-FM3

Printed in Denmark by IBM Danmark A/S

22L4285

SH12-6291-03

Spine information:

IBM MQSeries Workflow Programming Guide Version 3.2