IBM MQ Workflow for z/OS

Programming Guide

Version 3.3

SC33-7031-05

IBM MQ Workflow for z/OS

Programming Guide

Version 3.3

SC33-7031-05

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 647.

Fifth Edition (March 2001)

This edition applies to version 3, release 3 of IBM MQSeries Workflow for z/OS (product number 5655-BPM) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC33-7031-04.

Contents

Figures

Tables

About this book

This book describes how to use the IBM MQSeries Workflow for OS/390 for OS/390 Runtime Application Programming Interfaces (hereafter called MQ Workflow APIs) and also how to invoke API requests by passing messages in Extensible Markup Language (XML). The first part of the book describes the concepts underlying the APIs while the remainder of the book provides a reference for the API action calls. The book also describes the MQ Workflow predefined data structures.

Note: All references to the operating system OS/390 (R) in this book also apply to z/OS.

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 for OS/390. It assumes that readers are experienced programmers and that they understand the concepts of modeling processes.

How to get additional information

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

For a list of additional publications, refer to "MQ Workflow publications" on page 657.

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

"Notices" on page 647 describes some notices and trademarks.

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

"Chapter 2. Language interfaces" on page 137 discusses the API from the perspective of the language used: C, C++, Java, or COBOL.

"Chapter 3. Interfacing with the Program Execution Server" on page 189 describes the interface with the Program Execution Server, including the use of program mappings to bring Workflow API containers into a format acceptable by legacy applications and how to use exits.

"Chapter 4. Using the MQ Workflow Runtime API" on page 249 provides an overview of the classes supported by the API.

"Chapter 5. API action and activity implementation calls" on page 317 describes the API calls that enable applications to manipulate worklists and work items, to work with process instances and container data, and to log on to and log off from an MQ Workflow server.

"Chapter 6. Examples" on page 575 provides some examples that show how to use the API.

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

How to read the syntax diagrams

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

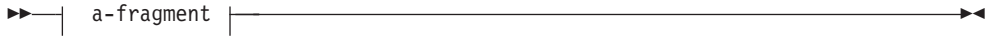
- Read the syntax diagrams from left to right, from top to bottom, following the main path of the line.

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

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

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

  The —►◄ symbol indicates the end of a statement.
- Diagrams can be broken into fragments. A fragment is indicated by vertical bars with the name of the fragment between the bars. The fragment itself follows the same syntactical rules as the main diagram.

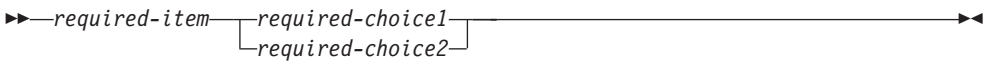


- Required items appear on the horizontal line, the main path.

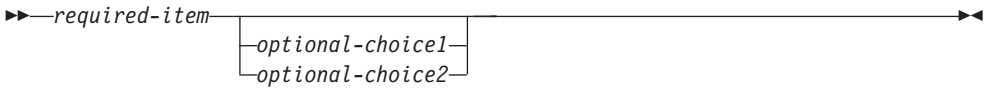
- Optional items appear below (or above) the main path.

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

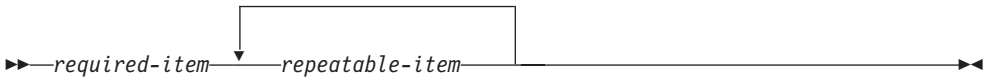
  If you must choose one of the items, one item of the stack appears on the main path.



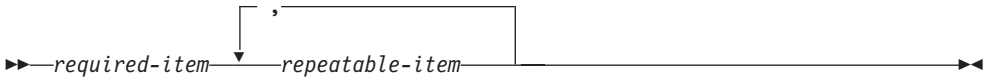
If choosing one of the items is optional, the entire stack appears below the main path.



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



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



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

Summary of Changes

Changes to this document for IBM MQSeries Workflow for OS/390 Version 3.3 are:

- API extensions are added to support stateless server implementations, that is:
  - The execution service exposes new API calls to retrieve a session representation and to recreate sessions, that is, SessionID() and SetSessionContext() are added.
  - New API calls are added so that all major objects can be externally represented by some string, that is, ProcessInstanceListPersistentOid(), ProcessTemplateListPersistentOid(), WorklistPersistentOid(), InstanceMonitorPersistentOid(), ReadOnlyContainerAsStream(), ReadWriteContainerAsStream(), ProgramDataAsStream(), and ProgramTemplateAsStream() are added.
  - The execution service exposes new API calls to recreate objects from their OID or from a stream, that is, PersistentActivityInstance(), PersistentActivityInstanceNotification(), PersistentInstanceMonitor(), PersistentPerson(), PersistentProcessInstance(), PersistentProcessInstanceList(), PersistentProcessInstanceNotification(), PersistentProcessTemplate(), PersistentProcessTemplateList(), PersistentWorkitem(), PersistentWorklist(), ProgramDataFromStream(), ProgramTemplateFromStream(), ReadOnlyContainerFromStream(), and ReadWriteContainerFromStream() are added.
- A new instance monitor is added. The instance monitor state can be passed between applications - see above - and nested monitors can be deleted without caring for the nesting level. Appropriate API calls are added to access the new monitor. Using the existing monitor is deprecated, that is, support will be removed in a later release or version.
- API extensions are added to support process repair actions, that is, ActivityInstanceForceFinish() and ActivityInstanceForceRestart() are added. These API calls allow to pass containers. The appropriate work item actions now allow to pass containers.
- The execution service supports a new way to log on with user credentials, which are passed to a user-provided authentication exit.
- A new activity instance and work item state *Expired* is supported, which can be used in queries for work items. The ExpirationTime() can be queried.
- The activity instance exposes a new API call Refresh().
- The process instance exposes a new API call to retrieve the OID of the associated process template, PersistentOidOfProcessTemplate().
- The container size can be 4 MB. CICS COMMAREA and IMS IOAREA restrictions must, however, be considered.
- Conversion between read/write and read-only containers is added.
- The Java API supports the ExecuteProcessInstance() method.
- Beside performance, the distinction between primary and secondary attributes is removed. That is, an object is automatically refreshed from the server when an attribute not yet available in the API is read.
- The audit mode can be more detailed, which is expressed by the *Filter* enumeration.
- An audit record can be sent to a user-defined MQSeries application. The XML DTD is extended appropriately.

- When a work item is terminated or expires, an XML message is sent to a UPES. The UPES must at least have version 3.3.0.

Changes to this document for IBM MQSeries Workflow for OS/390 Version 3.2.2 are:

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

Summary of deprecated API calls

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

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

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

Chapter 1. MQSeries Workflow programming concepts

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

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.

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

you would have to change them all. This mapper enables you to use all legacy applications with your Workflow applications via the mapping tool.

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

Programming interfaces

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

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

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

API interfaces in the following languages are described in this book:
- MQ Workflow C-language API
- MQ Workflow C++ language API
- MQ Workflow Java API
- MQ Workflow COBOL API
- MQ Workflow XML message interface

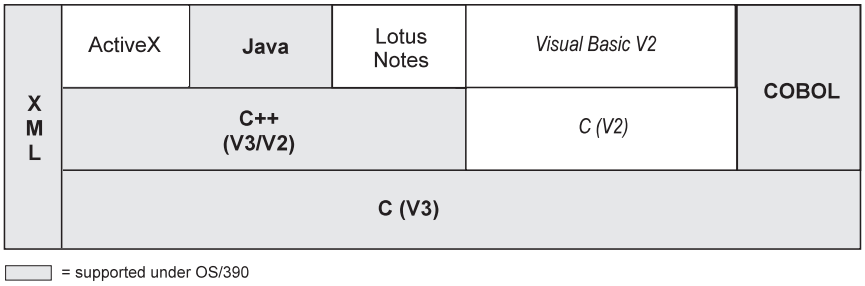


Figure 1. MQ Workflow Client APIs

The basic interfaces for requesting Runtime services from MQ Workflow are a C-language API and an XML message interface. Access can be gained to the C-language functions from all languages that support C calls - see “Compiling and linking” on page 139 for more information. A C++ language API and the COBOL API are provided on top of the C-language API. Since the C++ API is a small layer of inline methods, that is, delivered as source code, access can be gained from all popular C++ compilers. The Java API is implemented on top of the C++ layer. MQ Workflow uses the XML 1.0 standard as document description language.

The MQ Workflow APIs provide API calls:
- To execute process models, that is, to work with process instances and container data and to manipulate worklists and work items
- To monitor the progress of execution

- To issue process administrator functions
- To receive information sent by an MQ Workflow server
- To process container data associated with an activity implementation

Prerequisites for programming language API

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

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

Refer "Chapter 2. Language interfaces" on page 137 for more information.

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. 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 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". See the respective chapters per language.

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

Concepts of the programming language API

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

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

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

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

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

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

Concepts of the XML message interface

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

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

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

Handling errors

All action, activity implementation, program execution management API calls, or messages show whether or not the call has been successfully executed by returning a so-called *return code* as their return value. Java throws an appropriate FmcException when the method has not been executed successfully. The XML message interface provides the return code in the response message. The return code is one of a set of predefined codes (see "List of return codes" on page 9). The exact return codes or exceptions for each of those API calls are listed with the description of each call.

You should design your programs to handle all return codes or exceptions that can arise now or in future. That is, if you are not only asking whether the return code is different from FMC_OK, but, if you are checking return codes explicitly, then you should always care for unexpected errors. For example, if you are coding an *if* statement, then you should also code an *else* statement. If you are coding a *switch* statement, then you should also code the *default* case.

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

Although the result object is set by each API call, querying its contents can be of special importance for basic and accessor API calls. Basic and accessor API calls do not return any value or return the value queried as their return value. It can happen during application development that a wrong handle or a buffer too small to hold a character value is specified. Additional errors can occur when a not yet available attribute is automatically read from the server. To look for such erroneous situations, the *result object* can be queried (besides checking the trace).

The result object

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

- The return code.

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

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

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

All results of API calls are written into the result object associated with the thread the request executes in. It is sufficient to access the result object just once per-thread using the FmcjResultObjectOfCurrentThread() function respectively the FmcjResult::ObjectOfCurrentThread() method. The result object is automatically updated with each request.

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

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

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

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

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

  return 0;
}
```

Figure 2. Accessing a result object in the C-language

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

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

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

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

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

Figure 3. Accessing a result object in C++

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

For example, in COBOL, you can access and use a result object in the following ways:

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

DATA DIVISION.

  WORKING-STORAGE SECTION.

  COPY fmcvars.

  01 buffer PIC X(2000) VALUE SPACES.

PROCEDURE DIVISION.

     PERFORM FmcjResultObjOfCurrentThread.
     DISPLAY "Accessed result object of current thread".

     SET hdlResult TO FmcjResultHandleReturnValue.
     PERFORM FmcjResultRc.
     DISPLAY "Return code: " intReturnValue.
     MOVE 2000 TO bufferlength.
     CALL "SETADDR" USING buffer messageBuffer.
     PERFORM FmcjResultMessageText.
     DISPLAY "Text       : " buffer.
     CALL "SETADDR" USING buffer originBuffer.
     PERFORM FmcjResultOrigin.
     DISPLAY "Origin     : " buffer.
     PERFORM FmcjResultParms.
     SET hdlVector TO FmcjStrVHandleReturnValue.

     CALL "SETADDR" USING buffer elementBuffer.
     PERFORM FmcjStrVNextResultParmElement.

     PERFORM UNTIL pointerReturnValue = NULL
        DISPLAY "Parameter  : " buffer
        PERFORM FmcjStrVNextResultParmElement
     END-PERFORM.

     STOP RUN.

     COPY fmcperf.
```

Figure 4. Accessing a result object in COBOL (via PERFORM)

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

DATA DIVISION.

  WORKING-STORAGE SECTION.

  COPY fmcvars.

  01 buffer PIC X(2000) VALUE SPACES.

PROCEDURE DIVISION.


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

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

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

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

     STOP RUN.
```

Figure 5. Accessing a result object in COBOL (via CALL)

Note: The SETADDR routine is shown in "Example of the use of strings" on page 162 .

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. For COBOL, the return code identifiers have a maximum length of 30 characters. Additional words in the return codes are separated by hyphens and not by underscores (as is common for C). In order to avoid misunderstandings, the C version of the return codes is used in this book, especially in descriptions of the API calls ("Chapter 5. API action and activity implementation calls" on page 317).

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

Table 1. List of return codes

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

Table 1. List of return codes (continued)

Numeric value	Symbolic value (C/C++)	Symbolic value (COBOL)
130	FMC_ERROR_INVALID_HANDLE	FMC-ERROR-INVAL-HANDLE
131	FMC_ERROR_NOT_EMPTY	FMC-ERROR-NOT-EMPTY
132	FMC_ERROR_INVALID_USER	FMC-ERROR-INVAL-USER
133	FMC_ERROR_OWNER_ALREADY_ASSIGNED	FMC-ERROR-OWNER-ALR-ASSIGNED
134	FMC_ERROR_INVALID_NAME	FMC-ERROR-INVAL-NAME
135	FMC_ERROR_INVALID_PROGRAMID	FMC-ERROR-INVAL-PROGRAMID
136	FMC_ERROR_SIZE_EXCEEDED	FMC-ERROR-SIZE-EXCEEDED
137	FMC_ERROR_INVALID_TEMPLATE_NAME	FMC-ERROR-INVAL-TEMPL-NAME
138	FMC_ERROR_INFINITE_RECURSION	FMC-ERROR-INFINITE-RECURSION
139	FMC_ERROR_SUB_PROC_MEMBER_NOT_SET	FMC-ERROR-SUBPROC-M-NOT-SET
140	FMC_ERROR_PROCESS_TEMPLATE_NOT_FOUND	FMC-ERROR-PROC-TEMPL-NOT-FOUND
406	FMC_ERROR_WRONG_ACT_IMPL_KIND	FMC-ERROR-WRONG-ACT-IMPL-KIND
500	FMC_ERROR_NON_LOCAL_USER	FMC-ERROR-NON-LOCAL-USER
501	FMC_ERROR_WRONG_KIND	FMC-ERROR-WRONG-KIND
502	FMC_ERROR_INVALID_ACTIVITY	FMC-ERROR-INVAL-ACT
503	FMC_ERROR_CHECKOUT_NOT_POSSIBLE	FMC-ERROR-CHKOUT-NOT-POSSIBLE
504	FMC_BACK_LEVEL_VERSION	FMC-BACK-LEVEL-VERSION
505	FMC_ERROR_NEWER_VERSION	FMC-ERROR-NEWER-VERSION
506	FMC_ERROR_INVALID_CORRELATION_ID	FMC-ERROR-INVAL-CORRELATION-ID
507	FMC_ERROR_NOT_ALLOWED	FMC-ERROR-NOT-ALLOWED
508	FMC_ERROR_BACK_LEVEL_OBJECT	FMC-ERROR-BACK-LEVEL-OBJ
509	FMC_ERROR_INVALID_CONTAINER	FMC-ERROR-INVAL-CNTR
510	FMC_ERROR_UNEXPECTED_CONTAINER	FMC-ERROR-UNEXPECTED-CNTR
511	FMC_ERROR_NO_PROGRAM_FOR_PLATFORM	FMC-ERROR-NO-PROG-FOR-PLATF
512	FMC_ERROR_LOGON_DENIED	FMC-ERROR-LOGON-DENIED
513	FMC_ERROR_AUTHENTICATION	FMC-ERROR-AUTHENTICATION
800	FMC_ERROR_BUFFER	FMC-ERROR-BUFFER
801	FMC_ERROR_INVALID_SESSION	FMC-ERROR-INVAL-SESSION
802	FMC_ERROR_INVALID_TIME	FMC-ERROR-INVAL-TIME
804	FMC_ERROR_NO_MORE_DATA	FMC-ERROR-NO-MORE-DATA
805	FMC_ERROR_INVALID_OID	FMC-ERROR-INVAL-OID
807	FMC_ERROR_INVALID_THRESHOLD	FMC-ERROR-INVAL-THRESHOLD
808	FMC_ERROR_INVALID_SORT	FMC-ERROR-INVAL-SORT
809	FMC_ERROR_OBJECT_IN_USE	FMC-ERROR-OBJ-IN-USE
810	FMC_ERROR_INVALID_DESCRIPTION	FMC-ERROR-INVAL-DESCRIPTION
811	FMC_ERROR_INVALID_INVOCATION_TYPE	FMC-ERROR-INVAL-INV-TYPE
812	FMC_ERROR_OWNER_NOT_FOUND	FMC-ERROR-OWNER-NOT-FOUND
813	FMC_ERROR_INVALID_LIST_TYPE	FMC-ERROR-INVAL-LIST-TYPE
814	FMC_ERROR_INVALID_RESULT_HANDLE	FMC-ERROR-INVAL-RESULT-HANDLE
815	FMC_ERROR_MESSAGE_CATALOG	FMC-ERROR-MESSAGE-CATALOG
816	FMC_ERROR_INVALID_SPECIFICATION	FMC-ERROR-INVAL-SPECIFICATION
817	FMC_ERROR_QRY_RESULT_TOO_LARGE	FMC-ERROR-QRY-RESULT-TOO-LARGE
818	FMC_ERROR_NO_VERSION_2_FILTER	FMC-ERROR-NO-VERSION-2-FILTER
819	FMC_ERROR_INVALID_USER_CONTEXT	FMC-ERROR-INVAL-USER-CONTEXT
820	FMC_ERROR_MESSAGE_STRING	FMC-ERROR-MESSAGE-STRING
821	FMC_ERROR_MESSAGE_SIZE_EXCEEDED	FMC-ERROR-MSG-SIZE-EXCEEDED
822	FMC_ERROR_INVALID_STREAM	FMC-ERROR-INVAL-STREAM
900	FMC_ERROR_NO_SYS_ADMIN	FMC-ERROR-NO-SYS-ADMIN
901	FMC_ERROR_INVALID_SESSION_MODE	FMC-ERROR-INVAL-SESSION-MODE
902	FMC_ERROR_PROGRAM_UNDEFINED	FMC-ERROR-PROGRAM-UNDEFINED
903	FMC_ERROR_PEA_NOT_RUNNING	FMC-ERROR-PEA-NOT-RUNNING
904	FMC_ERROR_PEA_NOT_LOCAL	FMC-ERROR-PEA-NOT-LOCAL

Table 1. List of return codes (continued)

Numeric value	Symbolic value (C/C++)	Symbolic value (COBOL)
905	FMC_ERROR_INVALID_ABSENCE_SPEC	FMC-ERROR-INVAL-ABSENCE-SPEC
1000	FMC_ERROR_NOT_SUPPORTED	FMC-ERROR-NOT-SUPPORTED
1012	FMC_ERROR_PROGRAM_NOT_DEFINED	FMC-ERROR-PROGRAM-NOT-DEFINED
1014	FMC_ERROR_PEA_NOT_REACHABLE	FMC-ERROR-PEA-NOT-REACHABLE
1015	FMC_ERROR_INVALID_PEA_FROM_CTNR	FMC-ERROR-INVALID-PEA-FRM-CTNR
1016	FMC_ERROR_INVALID_PEA_FROM_MODEL	FMC-ERROR-INVAL-PEA-FRM-MODEL
1017	FMC_ERROR_INVALID_SYSTEM_FROM_CTNR	FMC-ERROR-INVAL-SYSTEM-FRM-CTNR
1018	FMC_ERROR_INVALID_SYSTEM_FROM_MODEL	FMC-ERROR-INVAL-SYSTEM-FRM-MODEL
1019	FMC_ERROR_SUB_PROC_TERMINATED_BY_ERROR	FMC-ERROR-SB-PRC-TERM-BY-ERROR
1020	FMC_ERROR_NO_PEA_FOUND_FOR_AUTO_START	FMC-ERROR-NO-PEA-FND-FR-AUT-ST
1021	FMC_ERROR_NO_CTNR_ACCESS	FMC-ERROR-NO-CTNR-ACCESS
1022	FMC_ERROR_INVALID_CONFIG_ID	FMC-ERROR-INVAL-CONFIG_ID
1023	FMC_ERROR_MIG_OF_RUNNING_PROG	FMC-ERROR-MIG-OF-RUNNING-PROG
1024	FMC_ERROR_MIG_OF_CHCKDOUT_SUSP	FMC-ERROR-MIG-OF-CHCKDOUT-SUSP
1025	FMC_ERROR_MIGRATION_NO_SUBPROC	FMC-ERROR-MIGRATION-NO-SUBPROC
1100	FMC_ERROR_XML_DOCUMENT_INVALID	FMC-ERROR-XML-DOCUMENT-INVAL
1101	FMC_ERROR_XML_NO_MQSWF_DOCUMENT	FMC-ERROR-XML-NO-MQSWF-DOC
1102	FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED	FMC-ERROR-XML-MSG-NOT-SUPP
1103	FMC_ERROR_XML_WRONG_DATA_STRUCTURE	FMC-ERROR-XML-WRONG-DATA-STR
1104	FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND	FMC-ERROR-XML-D-M-NOT-FOUND
1105	FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE	FMC-ERROR-XML-D-M-WRONG-TYPE
1106	FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED	FMC-ERROR-XML-BACKCNT-EXCEEDED
1107	FMC_ERROR_XML_DOCUMENT_FORMAT	FMC-ERROR-XML-DOCUMENT-FORMAT
1108	FMC_ERROR_XML_PARAMETER_INCORRECT	FMC-ERROR-XML-PARM-INCORRECT
1109	FMC_ERROR_XML_PARAMETER_SIGNATURE_ INCORRECT	FMC-ERROR-XML-PARMSG-INCORRECT
1110	FMC_ERROR_XML_INVALID_ELEMENT	FMC-ERROR-XML-INVAL-ELEMENT
1111	FMC_ERROR_XML_INCORRECT_PARAMETER	FMC-ERROR-XML-INCORRECT-PARM
1150	FMC_ERROR_XML_PARSER_NOT_INSTALLED	FMC-ERROR-XML-PARSER-NOT-INST
2000	FMC_ERROR_INVALID_QUEUE_SCOPE	FMC-ERROR-INVAL-QUEUE-SCOPE
32013	FMC_ERROR_USER_SUPPORT_ MISMATCH	FMC-ERROR-USER-SUPPORT-MISMAT
32014	FMC_ERROR_SUPPORT_MODE_MISMATCH	FMC-ERROR-SUPPORT-MODE-MISMAT
32015	FMC_ERROR_IMPLEMENTATION_SUPPORT_ MISMATCH	FMC-ERROR-IMPL-SUPPORT-MISMAT
32202	FMC_ERROR_USER_NOT_AUTHORIZED	FMC-ERROR-USER-NOT-AUTH
32203	FMC_ERROR_LOCAL_USER_REQUIRED	FMC-ERROR-LOCAL-USER-REQUIRED
32204	FMC_ERROR_EXIT_ERROR	FMC-ERROR-EXIT-ERROR

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, COBOL) or destructed (C++). Transient objects are, however, no longer available when your application program ends.

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

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

Although the life time of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQ Workflow API. The MQ Workflow API provides a handle (C-language, COBOL) to you so that you can issue requests against the object. In the C++ API, that handle is the only data member of your class. Therefore, you are independent of internal changes. It further allows 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.

Client/server communication and data access models

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

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

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

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

Synchronous client/server communication

Applying a synchronous protocol means that you issue a request to an MQ Workflow server and then wait until you receive a response. All action API calls

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

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

Asynchronous client/server communication

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

When you asynchronously issue an action, then you do, however, receive an acknowledgement telling whether MQ Workflow accepted the request or not. You can also receive a correlation identification which you can use in order to receive a specific response. You can specifiy a user context in order to correlate a response received.

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

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

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

The push data access model

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

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

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

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

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

1. On creation of the item.
2. On deletion of the item.

3. Whenever a primary property of the item changes - see "Accessor/mutator API calls" on page 92 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.

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

Receiving information

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

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

A timeout value of -1 specifies an indefinite wait time interval. Note that in this case you must ensure that you stop receiving data before your application ends. There is a TerminateReceive() API call which can be used to send a terminate indication to the receiving part of the application in order to inform that receiving data may end.

Notes:

1. A Receive() call survives a Logoff() call which ends your session with an execution server. The execution server stops, however, pushing information when logoff has been executed. When you did not send a TerminateReceive() to the receiving application thread, then you have to end that thread because of other knowledge. TerminateReceive() can only be called as long as a session exists.

2. If information is not received and therefore stays in the client input queue, the MQSeries(R) expiration mechanism applies in order to get rid of such "dead" messages. The expiration time of client messages can be configured for MQ Workflow.

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

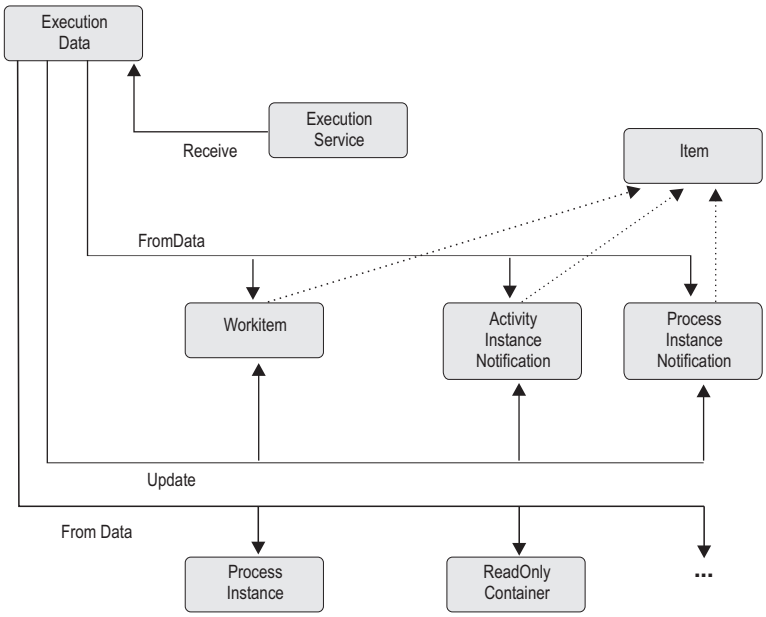


Figure 6. Handling data sent by an MQ Workflow server. Legend: -➤ Inheritance (C++); ⟶ provides for access

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

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

An MQ Workflow session

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

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

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

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

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

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

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

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

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

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

Using an authentication exit

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

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

use any authentication service like a simple file, a database, directory services, OS/390 Security Server (RACF) and so on.

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

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

Check SFMCSRC(FMCHSAXT) for an authentication exit programming example.

Coding an authentication exit

Authentication exits are supported in the C-language environment.

An authentication exit has a defined interface, to which every user-written exit must conform. For the C-language, the interface is described by the header file fmcaexit installed in the *SFMCH* library.

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

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

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

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

Notes:

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

C-language signatures

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

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

long FMC_APIENTRY Authenticate(
                          FmcjServerAuthExitAuthenticate * exitParms )
```

Parameters

All parameters but the *exitParms* are reserved for future use. The *exitParms* specify information exchanged between the MQ Workflow administration server and the authentication exit. See the fmch0xit header file for more detailed descriptions.

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

exitParameterListVersion
Input. The version of the parameter list.

exitParameterListLength
Input. The size of the parameter list structure.

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

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

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

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

userCredentialsSize
Input/Output. The size of the user credentials contained in the user credentials buffer.

userCredentialsBufferSize
Input. The size of the buffer containing the user credentials.

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

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

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

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

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

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

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

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

Activating an authentication exit

Set the *RTAuthenticationExitTypeServer* variable in the configuration profile of the server to "C". That is, add the specification to *CustHLQ*.SFMCDATA(FMCHEMPR).

Depending on the *RTAuthenticationExitTypeServer* setting, the authentication exit is loaded when the MQ Workflow administration server starts and the authentication exit Init() function is called.

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

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

To ensure adequate performance, the authentication exit must be a load library so that it can be loaded into the address space of the administration server. It has to be named *fmcaexit* and to be placed into a load library which is part of the STEPLIB concatenation.

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

Error handling

Logging on with user credentials can be rejected because:

- The authentication exit is not found; the required functions (entry points in the DLL) are not found; the called functions do not return FMC_EXIT_OK[1]. In all these situations, FMC_ERROR_NOT_AUTHORIZED is returned by the logon with credentials request.

1. FMC_EXIT_NONRECEOVERABLE_ERROR shuts down the administration server.

- The user ID returned by the authentication exit is not a registered MQ Workflow user ID; results in FMC_ERROR_USERID_UNKNOWN.
- The authentication exit denies authentication; results in FMC_ERROR_LOGON_DENIED.
- The authentication exit reports an error because of the stated reason; results in FMC_ERROR_AUTHENTICATION and the first parameter (of the result object) set to the reason code.

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.
- A query using a persistent list definition, which returns all objects qualifying through the list definition.
- A specific request, like the request for user settings or a refresh request for a specific object.

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

Persistent lists

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

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

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

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

Using filters, sort criteria, and thresholds

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

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

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

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

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

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

Handling collections

The result of a query for a set of objects is a so-called vector of objects in C, C++, or COBOL, or an array of objects in the 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 and COBOL.[2] If the vector handle is not initialized to 0, it **must** point to a vector of objects of the appropriate kind so that newly queried objects can be appended. In other words, any nonzero handle is used by the C-language or COBOL in order to access a vector assumed to already exist.

In the C-language or COBOL, 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 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 and COBOL 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.

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

Return codes

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

FMC_OK(0) The API call completed successfully.

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

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

FMC_ERROR_NO_MORE_DATA(804)
The vector contains no or no more element.

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

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

FmcjXxxVectorDeallocate

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

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

C-language signature

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

COBOL

```
        FmcjXxxVectorDeallocate.

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

Parameters

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

FmcjXxxVectorFirstElement

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

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

C-language signature

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

COBOL

```
        FmcjXxxVectorFirstElement.

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

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

Return type
FmcjXxxHandle
The handle of the first element of the vector or 0.

FmcjXxxVectorNextElement

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

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

C-language signature

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

COBOL

```
        FmcjXxxVectorNextElement.

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

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 )
```

COBOL

```
        FmcjXxxVectorSize.

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

Parameters

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

Return type

unsigned long

The number of elements in the vector.

C-language examples

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

Using First/NextElement() calls:

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

Figure 7. Reading a vector in C (using First/NextElement() calls) (Part 1 of 5)

```
  FmcjGlobalConnect();

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

Figure 7. Reading a vector in C (using First/NextElement() calls) (Part 2 of 5)

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

Figure 7. Reading a vector in C (using First/NextElement() calls) (Part 3 of 5)

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

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

  FmcjProcessInstanceVectorDeallocate(&hdlVecor);
```

Figure 7. Reading a vector in C (using First/NextElement() calls) (Part 4 of 5)

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

   FmcjGlobalDisconnect();
   return FMC_OK;
}
```

Figure 7. Reading a vector in C (using First/NextElement() calls) (Part 5 of 5)

Using NextElement() call only:

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

Figure 8. Reading a vector in C (using NextElement() only) (Part 1 of 5)

```
  FmcjGlobalConnect();

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

Figure 8. Reading a vector in C (using NextElement() only) (Part 2 of 5)

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

Figure 8. Reading a vector in C (using NextElement() only) (Part 3 of 5)

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

Figure 8. Reading a vector in C (using NextElement() only) (Part 4 of 5)

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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

Figure 8. Reading a vector in C (using NextElement() only) (Part 5 of 5)

COBOL examples

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

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

       DATA DIVISION.

         WORKING-STORAGE SECTION.

         COPY fmcvars.
         COPY fmcconst.
         COPY fmcrcs.

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

         LINKAGE SECTION.

         01 retCode            PIC S9(9) BINARY.
```

Figure 9. Reading a vector in COBOL (using First/NextElement calls) (Part 1 of 4)

```
PROCEDURE DIVISION USING retCode.

     PERFORM FmcjGlobalConnect.
     PERFORM FmcjESAllocate.

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

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

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

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

Figure 9. Reading a vector in COBOL (using First/NextElement calls) (Part 2 of 4)

```
     PERFORM FmcjPIVFirstElement.
     SET hdlInstance TO FmcjPIHandleReturnValue.
     PERFORM FmcjPIVSize.
     MOVE ulongReturnValue TO numElements.

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

Figure 9. Reading a vector in COBOL (using First/NextElement calls) (Part 3 of 4)

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

     COPY fmcperf.
```

Figure 9. Reading a vector in COBOL (using First/NextElement calls) (Part 4 of 4)

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

DATA DIVISION.

  WORKING-STORAGE SECTION.

  COPY fmcvars.
  COPY fmcconst.
  COPY fmcrcs.

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

  LINKAGE SECTION.

  01 retCode            PIC S9(9) BINARY.
```

Figure 10. Reading a vector in COBOL (using NextElement calls only) (Part 1 of 4)

```
PROCEDURE DIVISION USING retCode.

     PERFORM FmcjGlobalConnect.
     PERFORM FmcjESAllocate.

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

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

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

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

Figure 10. Reading a vector in COBOL (using NextElement calls only) (Part 2 of 4)

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

PERFORM FmcjPIVNextElement.

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

Figure 10. Reading a vector in COBOL (using NextElement calls only) (Part 3 of 4)

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

COPY fmcperf.
```

Figure 10. Reading a vector in COBOL (using NextElement calls only) (Part 4 of 4)

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.

Handling containers

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

Data structure/container type

A data structure is uniquely identified by its name and contains an ordered list of *data members*. At Runtime, it can become a stream of 4 MB passed between the client and the server. Activity implementations need, however, to consider the restrictions imposed by CICS or IMS.

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

Data member/container element

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

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

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

Data members are specified using their fully qualified name within the container. The fully qualified name[3] of a data member is a name in dot notation where the hierarchy of nested data members is presented from left to right, and their names are separated by a dot.

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)
```

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

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

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

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

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

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

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

For detailed examples see "Chapter 6. Examples" on page 575.

The XML message interface

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

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

For example:

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

For more information refer to "Container data" on page 165.

Predefined data members

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

Predefined data member values can be:

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

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

There are the following sets of predefined data members:
- Fixed data members
- Process information data members
- Activity information data members

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

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

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

There are no arrays of any predefined data member.

Fixed data members

Fixed data members _ACTIVITY, _PROCESS, and _PROCESS_MODEL *cannot be set* using API API calls. Their values *can be read* using API container API calls. Fixed data member _RC is available in output containers but should only be set when your compiler does not support a program exit code.

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

  Data type: STRING

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

  Data type: STRING

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

  Data type: STRING

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

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

Data type: LONG

Process information data members

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

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

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

_PROCESS_INFO.Role

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

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

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

Data type: STRING

_PROCESS_INFO.Organization

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

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

Data type: STRING

_PROCESS_INFO.ProcessAdministrator

The user ID of the person notified if:

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

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

Data type: STRING

_PROCESS_INFO.Duration

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

If not set, the default is "Endless".

Data type: LONG

Activity information data members

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

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

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

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

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

_ACTIVITY_INFO.Priority

The numeric value assigned as the priority of an activity instance. MQ Workflow does not deduce any meaning from this value; it is just used for client purposes. Any integer value between 0 and 999 can be specified. If the value specified is invalid or the data member is not set, a default of 0 (zero) is used.

Data type: LONG

_ACTIVITY_INFO.MembersOfRoles

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

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

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

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

Data type: STRING

_ACTIVITY_INFO.CoordinatorOfRole

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

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

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

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

Data type: STRING

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

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

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

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

Data type: STRING

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

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

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

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

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

Data type: long

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

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

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

Data type: LONG

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

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

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

Data type: LONG

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

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

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

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

Data type: STRING

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

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

Data type: STRING

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

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

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

Data type: LONG

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

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

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

Data type: LONG

Determining the structure of an unknown container

There are various API calls in order to determine the structure of an unknown container and/or its leaves. Applied to a container, they return a collection of

container elements. Once the collection of container elements is available, similar API calls can be recursively applied in order to step down through a nested structure.

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

Determining the leaves

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

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
```

COBOL

```
FmcjCLeafCount.

   CALL    "FmcjContainerLeafCount"
                  USING
                  BY VALUE
                     hdlContainer
                  RETURNING
                     ulongReturnValue.

FmcjCLeaves.

   CALL    "FmcjContainerLeaves"
                  USING
                  BY VALUE
                     hdlContainer
                  RETURNING
                     FmcjCEVHandleReturnValue.

FmcjCAllLeafCount.

   CALL    "FmcjContainerAllLeafCount"
                  USING
                  BY VALUE
                     hdlContainer
                  RETURNING
                     ulongReturnValue.

FmcjCAllLeaves.

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

Parameters

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

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

Return type

ContainerElement[]/FmcjContainerElementVectorHandle
The container elements which are leaves.

long/unsigned long/int
The number of user-defined leaves or the number of all leaves, user-defined and predefined.

Determining the structural members

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

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
```

COBOL

```
        FmcjCMemberCount.

           CALL    "FmcjContainerMemberCount"
                          USING
                          BY VALUE
                             hdlContainer
                          RETURNING
                             ulongReturnValue.

        FmcjCStructMembers.

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

Parameters

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

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

Return type

ContainerElement[]/FmcjContainerElementVectorHandle
The container elements which are part of the container.

long/unsigned long/int
The number of structural members in the container.

Determining the type

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

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
```

COBOL

```
      FmcjCType.

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

Parameters

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

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

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

Return type

BSTR/char*/string/String
The type of the container.

Analyzing a container element

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

Determining the name or type of a container element

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

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
```

COBOL

```
FmcjCEName.

   CALL    "FmcjContainerElementName"
                  USING
                  BY VALUE
                     hdlElement
                     elementNameBuffer
                     bufferLength
                  RETURNING
                     pointerReturnValue.


FmcjCEFullName.

   CALL    "FmcjContainerElementFullName"
                  USING
                  BY VALUE
                     hdlElement
                     elementNameBuffer
                     bufferLength
                  RETURNING
                     pointerReturnValue.


FmcjCEType.

   CALL    "FmcjContainerElementType"
                  USING
                  BY VALUE
                     hdlElement
                     containerTypeBuffer
                     bufferLength
                  RETURNING
                     pointerReturnValue.
```

Parameters

bufferLength Input. The length of the buffer to be filled.

buffer Input/Output. The buffer to contain the container element name or type.

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

Return type

BSTR/char*/string/String

The name or type of the container element.

Determining the structural properties of a container element

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

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
```

COBOL

```
        FmcjCEIsArray.

           CALL    "FmcjContainerElementIsArray"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             boolReturnValue.

        FmcjCEIsLeaf.

           CALL    "FmcjContainerElementIsLeaf"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             boolReturnValue.

        FmcjCEIsStruct.

           CALL    "FmcjContainerElementIsStruct"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             boolReturnValue.
```

Parameters

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

Return type

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

Determining the leaves of a container element

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

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

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
```

COBOL

```
        FmcjCELeafCount.

           CALL    "FmcjContainerElementLeafCount"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             ulongReturnValue.

       FmcjCELeaves.

           CALL    "FmcjContainerElementLeaves"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             FmcjCEVHandleReturnValue.
```

Parameters

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

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

Return type

ContainerElement[]/FmcjContainerElementVectorHandle
The container elements which are leaves.

long/unsigned long/int
The number of user-defined leaves.

Determining the structural members of a container element

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

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
```

COBOL

```
        FmcjCEMemberCount.

           CALL    "FmcjContainerElementMemberCount"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             ulongReturnValue.

        FmcjCEStructMembers.

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

Parameters

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

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

Return type

ContainerElement[]/FmcjContainerElementVectorHandle
The container elements which are structural members.

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

Determining the elements of an array

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

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
```

COBOL

```
        FmcjCECardinality.

           CALL    "FmcjContainerElementCardinality"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             ulongReturnValue.

        FmcjCEArrayElements.

           CALL    "FmcjContainerElementArrayElements"
                          USING
                          BY VALUE
                             hdlElement
                          RETURNING
                             FmcjCEVHandleReturnValue.
```

Parameters

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

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

Return type

ContainerElement[]/FmcjContainerElementVectorHandle

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

long/unsigned long
The cardinality of the array described by the container element.

Accessing a known container element

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

Note: A qualified name must start with a letter and cannot start with brackets or parentheses. In other words, if you want to access an element of a container element which is an array, then you need to call the ArrayElements() API call.

C-language signature

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

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

C++ language signature

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

Java signature

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

COBOL

```
        FmcjCGetElement.

           CALL    "FmcjContainerGetElement"
                          USING
                          BY VALUE
                             hdlContainer
                             qualifiedName
                          BY REFERENCE
                             element
                          RETURNING
                             intReturnValue.

     FmcjCEGetElement.

           CALL    "FmcjContainerElementGetElement"
                          USING
                          BY VALUE
                             hdlElement
                             qualifiedName
                          BY REFERENCE
                             element
                          RETURNING
                             intReturnValue.
```

Parameters

element Output. The container element.

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

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

Return type

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

Accessing a value of a container

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

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

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
```

COBOL

```
FmcjCArrayBinaryLength.

   CALL    "FmcjContainerArrayBinaryLength"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     indexValue
                  RETURNING
                     ulongReturnValue.


FmcjCArrayBinaryValue.

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

FmcjCBinaryLength.

   CALL    "FmcjContainerBinaryLength"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                  RETURNING
                     ulongReturnValue.


FmcjCBinaryValue.

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

COBOL

```
FmcjCArrayFloatValue.

   CALL    "FmcjContainerArrayFloatValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     indexValue
                  BY REFERENCE
                     doubleValue
                  RETURNING
                     intReturnValue.

FmcjCFloatValue.

   CALL    "FmcjContainerFloatValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                  BY REFERENCE
                     doubleValue
                  RETURNING
                     intReturnValue.
```

COBOL

```
FmcjCArrayLongValue.

   CALL    "FmcjContainerArrayLongValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     indexValue
                  BY REFERENCE
                     intValue
                  RETURNING
                     intReturnValue.

FmcjCLongValue.

   CALL    "FmcjContainerLongValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                  BY REFERENCE
                     intValue
                  RETURNING
                     intReturnValue.
```

COBOL

```
FmcjCArrayStringLength.

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


FmcjCArrayStringValue.

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

FmcjCStringLength.

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

FmcjCStringValue.

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

Parameters

bufferLength Input. The length of the buffer available for passing the value; must be greater than or equal to the actual length. Use the appropriate Length() API calls to determine the actual length.

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

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

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

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

value Output. The value of the leaf.

Return type

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

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

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

Accessing a value of a container element

The following API calls return the value of a container element leaf. When the leaf is an array of values, an index must be specified.
FMC_ERROR_MEMBER_NOT_SET is returned if no information is available. Note that, in contrast to querying container leaves, the name of the leaf need not be specified because the container element itself is the leaf queried.

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
```

COBOL

```
FmcjCEArrayBinaryLength.

   CALL    "FmcjContainerElementArrayBinaryLength"
                  USING
                  BY VALUE
                     hdlElement
                     indexValue
                  RETURNING
                     ulongReturnValue.


FmcjCEArrayBinaryValue.

   CALL    "FmcjContainerElementArrayBinaryValue"
                  USING
                  BY VALUE
                     hdlElement
                     indexValue
                     pointerValue
                     dataLength
                  RETURNING
                     intReturnValue

FmcjCEBinaryLength.

   CALL    "FmcjContainerElementBinaryLength"
                  USING
                  BY VALUE
                     hdlElement
                  RETURNING
                     ulongReturnValue.


FmcjCEBinaryValue.

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

COBOL

```
FmcjCEArrayFloatValue.

   CALL    "FmcjContainerElementArrayFloatValue"
                  USING
                  BY VALUE
                     hdlElement
                     indexValue
                  BY REFERENCE
                     doubleValue
                  RETURNING
                     intReturnValue.

FmcjCEFloatValue.

   CALL    "FmcjContainerElementFloatValue"
                  USING
                  BY VALUE
                     hdlElement
                  BY REFERENCE
                     doubleValue
                  RETURNING
                     intReturnValue.
```

COBOL

```
FmcjCEArrayLongValue.

   CALL    "FmcjContainerElementArrayLongValue"
                  USING
                  BY VALUE
                     hdlElement
                     indexValue
                  BY REFERENCE
                     intValue
                  RETURNING
                     intReturnValue.

FmcjCELongValue.

   CALL    "FmcjContainerElementLongValue"
                  USING
                  BY VALUE
                     hdlElement
                  BY REFERENCE
                     intValue
                  RETURNING
                     intReturnValue.
```

COBOL

```
FmcjCEArrayStringLength.

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

FmcjCEArrayStringValue.

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

FmcjCEStringLength.

   CALL    "FmcjContainerElementStringLength"
                  USING
                  BY VALUE
                     hdlElement
                  RETURNING
                     ulongReturnValue.


FmcjCEStringValue.

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

Parameters

bufferLength Input. The length of the buffer available for passing the value; must be greater than or equal to the actual length. Use the appropriate Length() API calls to determine the actual length.

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

index Input. When the leaf is an array, the index of the array element to be queried. In ActiveX, the index is ignored for a container element which is no array.

value Output. The value of the leaf.

Return type

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

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

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

Setting a value of a container

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

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

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

C-language signatures

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

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

C-language signatures

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

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

C-language signatures

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

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

C-language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

C++ language signatures

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

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

Java signatures

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

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

Java signatures

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

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

Java signatures

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

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

Java signatures

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

COBOL

```
FmcjRWCSetArrayBinaryValue.

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

FmcjRWCSetBinaryValue.

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

COBOL

```
FmcjRWCSetArrayFloatValue.

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

FmcjRWCSetFloatValue.

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

COBOL

```
FmcjRWCSetArrayLongValue.

   CALL    "FmcjReadWriteContainerSetArrayLongValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     indexValue
                     intValue
                  RETURNING
                     intReturnValue.

FmcjRWCSetLongValue.

   CALL    "FmcjReadWriteContainerSetLongValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     intValue
                  RETURNING
                     intReturnValue.
```

COBOL

```
FmcjRWCSetArrayStringValue.

   CALL    "FmcjReadWriteContainerSetArrayStringValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     indexValue
                     pointerValue
                  RETURNING
                     intReturnValue.

FmcjRWCSetStringValue.

   CALL    "FmcjReadWriteContainerSetStringValue"
                  USING
                  BY VALUE
                     hdlContainer
                     qualifiedName
                     pointerValue
                  RETURNING
                     intReturnValue.
```

Parameters

dataLength Input. The length of the binary value.

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

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

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

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

value Input. The value of the leaf. Note that values for leaves of type

BINARY must be specified as a sequence of two-digit hexadecimal numbers. For example, the string 'abc<cr><lf>'' would be represented as '6162630d0a' (where <cr> denotes the ASCII 'carriage return' character and <lf> denotes the ASCII line-feed character).

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

Return codes/FmcException

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

FMC_OK(0) The API call completed successfully.

FMC_ERROR_BUFFER(800)
The provided buffer is too small.

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

FMC_ERROR_EMPTY(122)
The object has not yet been read from the server.

FMC_ERROR_FORMAT(117)
The qualified name does not conform to the syntax rules.

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

FMC_ERROR_MEMBER_CANNOT_BE_SET(115)
The specified member is an MQ Workflow predefined fixed data member; it is for information only.

FMC_ERROR_MEMBER_NOT_FOUND(112)
The specified member is not part of the container or container element.

FMC_ERROR_MEMBER_NOT_SET(113)
The specified member has no value.

Monitoring a process instance

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

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

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

Obtaining an process instance monitor

Once a process instance[4] has been accessed, an **instance monitor** for a process instance can be obtained (ObtainProcessMonitor()). The transient instance monitor object then represents all information about activity instances directly contained in the described process instance as well as all information on control connector instances connecting those activity instances.

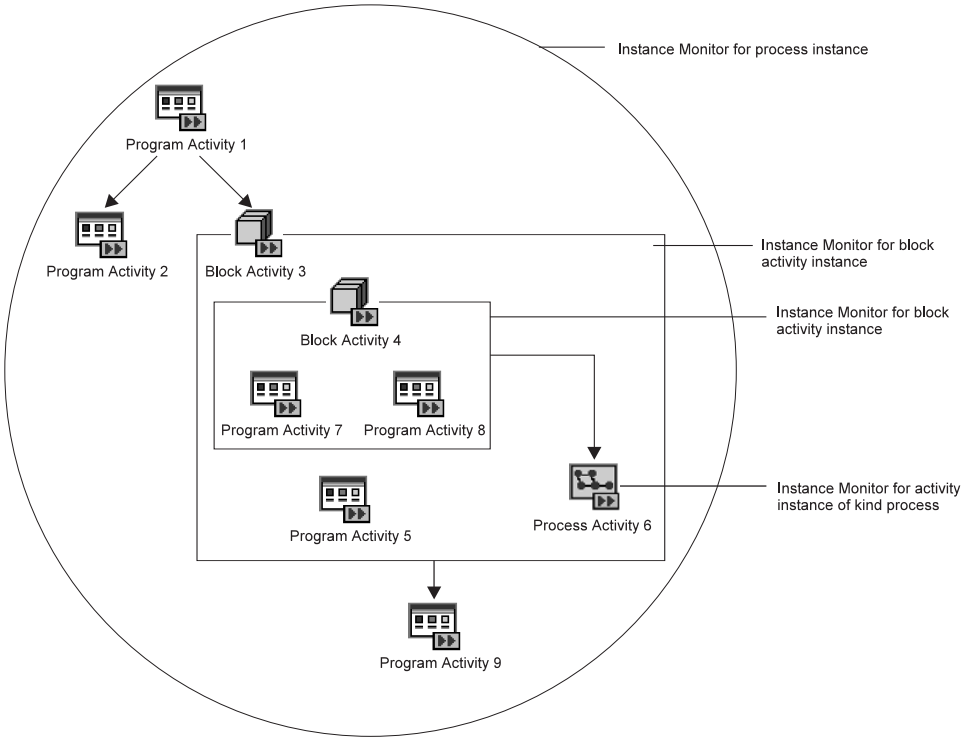


Figure 11. Instance monitors for process instances and activity instances of type Block

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

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

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

4. or activity instance or a (work) item

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

When obtaining an process instance monitor, it is possible to use the *deep option* in order to specify that *all* monitors for activities of type *Block* are to be returned from the MQ Workflow execution server in the same step. The instance monitors for the block activity instances then all show the state of the process instance at this retrieval time. This means, when an block 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 an 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, an instance monitor for a process instance or activity instance is owned by the caller of the API. When an instance monitor is no longer needed, you should delete/deallocate the object.

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 34 for details).

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

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

Table 2. Authorization for persons

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

Table 2. Authorization for persons (continued)

Name	E/I	Authorized Functions
Process authorization	E	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.
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

Table 2. Authorization for persons (continued)

Name	E/I	Authorized Functions
Work item authority	E	Can perform the following functions on (work) items for all persons if you are authorized for all persons or for selected persons: • 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

Stateless server support

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

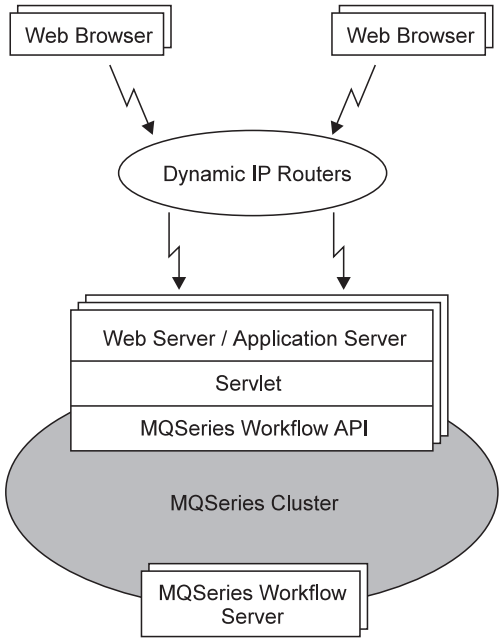


Figure 12. Stateless server support

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

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

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

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

Identity-based objects are:

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

These objects expose API calls:

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

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

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

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

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

Value-based objects are:

- Containers
- Program data
- Program templates

These objects expose API calls:

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

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

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

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

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

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

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

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

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

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

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

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

   and the stream format of the container.

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

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

   Since the process template and its container are the objects which are potentially acted on by the next client request, application B returns the process

template OID and the container stream to the caller so that the next request can be handled by a different application process; even on a different operating system platform.

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

   It allocates an execution service object and attaches itself to the specified session - see 2 on page 74.

   It recreates the process template from its OID

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

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

   and the container from the stream passed.

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

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

   Application process C issues the CreateAndStartProcessInstance() action method. On successful return from the MQ Workflow server, it retrieves the OID of the process instance created and returns that information to the caller - see 2 on page 74.

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

   It allocates an execution service object and attaches itself to the specified session - see 2 on page 74.

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

   It issues the Terminate() action method.

5. Depending on the client request, issue MQ Workflow API calls similar to steps 2 on page 74, 3, or 4.

Types of API calls

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

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

Basic and accessor API calls are described in more detail, but still generally, in the following paragraphs. This is because all these API calls are similar in appearance and have similar requirements, even for different objects. They are all handled locally by the API, that is, they do not communicate with the server. The API calls of the other categories are described separately in "Chapter 5. API action and activity implementation calls" on page 317. These API calls require client/server communication or communication with the program execution server.

Basic API calls

Basic API calls are essentially provided so that transient objects can be allocated or constructed and deallocated or destructed. They allow for the construction of supporting objects like service objects. They allow for the destruction of such objects as well as for the destruction of transient representations of persistent objects allocated implicitly by the MQ Workflow API. Refer also to "Memory management" on page 11.

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

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

Return codes

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

FMC_OK(0) The API call completed successfully.

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

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

FMC_ERROR_INVALID_NAME(134)
The name provided is invalid; it is a 0-pointer or it does not conform to the syntax rules.

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

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

Allocation

Following API calls allow the application to set up the respective object. This is needed for supporting objects like string vectors. 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.

C-language signatures

```
APIRET FMC_APIENTRY
FmcjExecutionServiceAllocate( FmcjExecutionServiceHandle * service )

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

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

APIRET FMC_APIENTRY
FmcjStringVectorAllocate( FmcjStringVectorHandle * hdlVector )
```

C++ language signature

```
FmcjXxx()

FmcjDateTime( bool initWithCurrentDateTime= false )

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

FmcjExecutionService( string const & systemGroup )

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

Java signatures

```
Agent()

ReadOnlyContainerHolder()
ReadOnlyContainerHolder( ReadOnlyContainer value )
```

COBOL

```
FmcjESAllocate.

   CALL    "FmcjExecutionServiceAllocate"
                  USING
                  BY REFERENCE
                     serviceValue
                  RETURNING
                     intReturnValue.

FmcjESAllocateForGroup.

   CALL    "FmcjExecutionServiceAllocateForGroup"
                  USING
                  BY VALUE
                     systemGroup
                  BY REFERENCE
                     serviceValue
                  RETURNING
                     intReturnValue.

FmcjESAllocateForSyst.

   CALL    "FmcjExecutionServiceAllocateForSystem"
                  USING
                  BY VALUE
                     system
                     systemGroup
                  BY REFERENCE
                     serviceValue
                  RETURNING
                     intReturnValue.

FmcjStrVAllocate.

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

Parameters

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

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

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

systemGroup
Input. The system group where the execution server resides. Only specifying the system group allows for exploiting the MQSeries clustering capabilities.

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

hour/minute/second
Input. The time part of the date/time.
value Input. A read-only container which initializes the holder object.

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

Assignment

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

C++ language signature

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

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

Comparison/equality

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

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

Exceptions:

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

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

C-language signature

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

C++ language signature

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

COBOL

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

Parameters

anObject Input. The object to be compared with this one.
handle1 Input. The first object to be compared.
handle2 Input. The other object to be compared.

Conversion

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

C-language signatures

```
FmcjReadWriteContainerHandle FMC_APIENTRY
    FmcjReadOnlyContainerAsReadWriteContainer(
                          FmcjReadOnlyContainerHandle handle )

FmcjReadOnlyContainerHandle FMC_APIENTRY
    FmcjReadWriteContainerAsReadOnlyContainer(
                           FmcjReadWriteContainerHandle handle )
```

C++ language signatures

```
operator FmcjReadWriteContainer();

operator FmcjReadOnlyContainer();
```

Java signatures

```
public abstract
ReadWriteContainer asReadWriteContainer() throws FmcException

public abstract
ReadOnlyContainer  asReadOnlyContainer () throws FmcException
```

COBOL

```
FmcjROCAsReadWriteContainer.
     CALL    "FmcjReadOnlyContainerAsReadWriteContainer"
                    USING
                    BY VALUE
                       handle
                    RETURNING
                       FmcjRWCHandleReturnValue.

FmcjRWCAsReadOnlyContainer.
     CALL    "FmcjReadWriteContainerAsReadOnlyContainer"
                    USING
                    BY VALUE
                       handle
                    RETURNING
                       FmcjROCHandleReturnValue.
```

Parameters

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

Copy

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

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

C-language signature

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

C++ language signature

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

COBOL

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

Parameters

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

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

Deallocation

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

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

C-language signature

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

C++ language signature

```
virtual ¬FmcjXxx()
```

COBOL

```
        FmcjXxxDeallocate.

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

Parameters

handle

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

IsComplete()

Returns true when the object has been completely read from an MQ Workflow server, that is, both primary and secondary properties are available (see also "Accessor/mutator API calls" on page 92).

C-language signature

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

C++ language signature

```
bool IsComplete()
```

Java signature

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

COBOL

```
FmcjXxxIsComplete.

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

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

Return type
bool/boolean
True if the object has been completely read from the server, otherwise false.

IsEmpty()

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

C++ 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.

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
```

COBOL

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

Parameters

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

Return type

FmcjXxxEnum/Enum

The kind of the object; some element of an enumeration - see also "Accessing an enumerated value" on page 96.

C-language Example: using basic functions

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

Figure 13. C example using basic functions (Part 1 of 8)

```
  FmcjGlobalConnect();

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

Figure 13. C example using basic functions (Part 2 of 8)

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

Figure 13. C example using basic functions (Part 3 of 8)

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

Figure 13. C example using basic functions (Part 4 of 8)

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

Figure 13. C example using basic functions (Part 5 of 8)

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

Figure 13. C example using basic functions (Part 6 of 8)

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

Figure 13. C example using basic functions (Part 7 of 8)

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

  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

Figure 13. C example using basic functions (Part 8 of 8)

C++ Example: using basic methods

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

Figure 14. C++ example using basic methods (Part 1 of 7)

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

Figure 14. C++ example using basic methods (Part 2 of 7)

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

Figure 14. C++ example using basic methods (Part 3 of 7)

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

Figure 14. C++ example using basic methods (Part 4 of 7)

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

Figure 14. C++ example using basic methods (Part 5 of 7)

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

Figure 14. C++ example using basic methods (Part 6 of 7)

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

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

Figure 14. C++ example using basic methods (Part 7 of 7)

COBOL example using basic calls

Note: The SETADDR routine, which sets a pointer item to the address of a given string, is listed in "Example of the use of strings" on page 162.

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

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

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

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      PERFORM FmcjGlobalConnect.
* logon
      PERFORM FmcjESAllocate.

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

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

Figure 15. COBOL example using basic calls (via PERFORM) (Part 1 of 2)

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

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

      PERFORM FmcjWIVDeallocate.

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

      COPY fmcperf.
```

Figure 15. COBOL example using basic calls (via PERFORM) (Part 2 of 2)

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

DATA DIVISION.

  WORKING-STORAGE SECTION.

  COPY fmcvars.
  COPY fmcconst.
  COPY fmcrcs.

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

  LINKAGE SECTION.

  01 retCode            PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

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

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

        IF ulongReturnValue >= 2

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

Figure 16. COBOL example using basic calls (via CALL) (Part 1 of 3)

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

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

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

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

Figure 16. COBOL example using basic calls (via CALL) (Part 2 of 3)

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

Figure 16. COBOL example using basic calls (via CALL) (Part 3 of 3)

Accessor/mutator API calls

Accessor/mutator API calls are provided so that properties of transient objects can be read or changed. If the transient object represents a persistent one, then the values that are returned reflect the state of the persistent object when it was retrieved and used to set the transient object or when it was created or updated. Retrieval is automatically done from an MQ Workflow server when the property is accessed and not yet available in the API cache or explicitly done by using the appropriate create, query, or refresh API calls. Creation or update can be done on the client when the MQ Workflow server sends new information (pushes information).

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

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

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

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

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

Primary/secondary properties

By default, the MQ Workflow API provides for two views on persistent objects. They divide the persistent object into so-called *primary* properties and so-called *secondary* properties. Primary properties are considered “more important” from an

access point of view. They are immediately provided by the server when objects are queried and can be used in filter expressions. Secondary properties, and a refresh of the primary properties, are only provided when the secondary property is accessed (the API automatically refreshes the object when a property is queried and not yet available) or on an explicit Refresh() request; on a per-object basis. You can use the *IsComplete()* API call to determine whether both primary and secondary object values have been read from the server.

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

Return codes

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

FMC_OK(0) The API call completed successfully.

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

FMC_ERROR_EMPTY(122)
The object has not yet been read from the database, that is, default values are returned.

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

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile; can be returned when the API automatically refreshes the object.

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

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

FMC_ERROR_INVALID_CONFIGURATION_ID(1022)
The configuration provided is invalid; it is 0 or it does not conform to its syntax rules.

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

FMC_ERROR_INVALID_RESULT_HANDLE(814)
The handle of the result object provided is invalid; it is 0 or it is not pointing to a result object.

FMC_ERROR_INVALID_TIME(802)
The time passed is invalid.

FMC_ERROR_MESSAGE_CATALOG(815)
The message catalog cannot be found.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error; can be returned when the API automatically refreshes the object. Contact your IBM representative.

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

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

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

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

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

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

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

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

Accessing a value of type bool

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

C-language signature

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

C++ language signature

```
bool Property() const
```

Java signature

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

COBOL

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

Parameters

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

Return type

bool/ boolean The property value.

Declaration examples

C-language bool FMC_APIENTRY FmcjWorkitemManualStartMode(FmcjWorkitemHandle handle);

C++ bool ManualStartMode() const;

Java public abstract boolean manualStartMode() throws FmcException;

Accessing a value of type date/time

Returns the value of a date/time property. A zero timestamp is returned if no information is available. Date/time values are expressed in local time.

C-language signature

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

C++ language signature

```
FmcjDateTime Property() const
```

Java signature

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

COBOL

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

Parameters

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

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

Return type
FmcjCDateTime/ FmcjDateTime/Calendar
The property value.

Declaration examples

C-language FmcjCDateTime FMC_APIENTRY FmcjWorkitemEndTime(FmcjWorkitemHandle handle);

C++ FmcjDateTime EndTime() const;

Java public abstract Calendar endTime() throws FmcException;

Accessing an enumerated value

Returns an enumerating value of a property. It is strongly advised to use the symbolic names in order to determine the actual value instead of the corresponding integer values. It is not guaranteed that integer values always stay the same.

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

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

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
```

COBOL

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

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

Return type
FmcjXxxEnum/Enum
The property value, some element of an enumeration.

Declaration examples

C-language FmcjItemAssignReason FMC_APIENTRY FmcjWorkitemReceivedAs(FmcjWorkitemHandle handle);

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

Java public abstract AssignReason receivedAs() throws FmcException;

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

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

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

- **C-language** Fmc_IR_NotSet
- **C++** FmcjItem::NotSpecified
- **Java** AssignReason.NOT_SPECIFIED
- **COBOL** Fmc-IR-NotSet

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

- **C-language** Fmc_IR_Normal
- **C++** FmcjItem::Normal
- **Java** AssignReason.NORMAL
- **COBOL** Fmc-IR-Normal

Substitute(2) Indicates that the work item or notification has been assigned because the user is the substitute for the person who should have received the item.

- **C-language** Fmc_IR_Substitute
- **C++** FmcjItem::Substitute
- **Java** AssignReason.Substitute
- **COBOL** Fmc-IR-Substitute

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

- **C-language** Fmc_IR_ProcessAdministrator
- **C++** FmcjItem::ProcessAdministrator
- **Java** AssignReason.PROCESS_ADMINISTRATOR
- **COBOL** Fmc-IR-ProcAdministrator

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

- **C-language** Fmc_IR_SystemAdministrator

	C++	FmcjItem::SystemAdministrator
	Java	AssignReason.SYSTEM_ADMINISTRATOR
	COBOL	Fmc-IR-SystAdministrator
ByTransfer(5)	Indicates that the work item or notification has been transferred to the user.	
	C-language	Fmc_IR_ByTransfer
	C++	FmcjItem::ByTransfer
	Java	AssignReason.BY_TRANSFER
	COBOL	Fmc-IR-ByTransfer

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

NotSet(0)	Indicates that nothing is known about the audit setting.	
	C-language	Fmc_TA_NotSet
	C++	FmcjProcessTemplate::NotSet
	Java	AuditSetting.NOT_SET
	COBOL	Fmc-TA-NotSet
NoAudit(1)	Indicates that auditing is not to be performed.	
	C-language	Fmc_TA_NoAudit
	C++	FmcjProcessTemplate::NoAudit
	Java	AuditSetting.NO_AUDIT
	COBOL	Fmc-TA-NoAudit
Condensed(2)	Indicates that condensed auditing is to be performed.	
	C-language	Fmc_TA_Condensed
	C++	FmcjProcessTemplate::Condensed
	Java	AuditSetting.CONDENSED
	COBOL	Fmc-TA-Condensed
Full(3)	Indicates that full auditing is to be performed.	
	C-language	Fmc_TA_Full
	C++	FmcjProcessTemplate::Full
	Java	AuditSetting.FULL
	COBOL	Fmc-TA-Full
Filter(4)	Indicates that fine-grained, filtered auditing is to be performed.	
	C-language	Fmc_TA_Filter
	C++	FmcjProcessTemplate::Filter
	Java	AuditSetting.FILTER
	COBOL	Fmc-TA-Filter

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

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

C-language	Fmc_AE_NotSet
C++	FmcjActivityInstance::NotSpecified
Java	Escalation.NOT_SPECIFIED
COBOL	Fmc-AE-NotSet

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

C-language	Fmc_AE_NoNotification
C++	FmcjActivityInstance::NoNotification
Java	Escalation.NO_NOTIFICATION
COBOL	Fmc-AE-NoNotif

FirstNotification
Indicates that the first notification occurred.

C-language(4)	Fmc_AE_FirstNotification
C++(4)	FmcjActivityInstance::FirstNotification
Java(2)	Escalation.FIRST_NOTIFICATION
COBOL(4)	Fmc-AE-FirstNotif

SecondNotification
Indicates that the second notification occurred.

C-language(5)	Fmc_AE_SecondNotification
C++(5)	FmcjActivityInstance::SecondNotification
Java(3)	Escalation.SECOND_NOTIFICATION
COBOL(5)	Fmc-AE-SecNotif

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

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

C-language	Fmc_AS_NotSet
C++	FmcjActivityInstance::undefined
Java	ExecutionState.UNDEFINED
COBOL	Fmc-AS-NotSet

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

C-language	Fmc_AS_Ready

	C++	FmcjActivityInstance::ready
	Java	ExecutionState.READY
	COBOL	Fmc-AS-Ready
Running(2)	Indicates that the activity instance is in the running state.	
	C-language	Fmc_AS_Running
	C++	FmcjActivityInstance::running
	Java	ExecutionState.RUNNING
	COBOL	Fmc-AS-Running
Finished	Indicates that the activity instance is in the finished state.	
	C-language(4)	Fmc_AS_Finished
	C++(4)	FmcjActivityInstance::finished
	Java(3)	ExecutionState.FINISHED
	COBOL(4)	Fmc-AS-Finished
Terminated	Indicates that the activity instance is in the terminated state.	
	C-language(8)	Fmc_AS_Terminated
	C++(8)	FmcjActivityInstance::terminated
	Java(4)	ExecutionState.TERMINATED
	COBOL(8)	Fmc-AS-Term
Suspended	Indicates that the activity instance is in the suspended state.	
	C-language(16)	Fmc_AS_Suspended
	C++(16)	FmcjActivityInstance::suspended
	Java(5)	ExecutionState.SUSPENDED
	COBOL(16)	Fmc-AS-Suspended
Inactive	Indicates that the activity instance is still inactive.	
	C-language(32)	Fmc_AS_Inactive
	C++(32)	FmcjActivityInstance::inactive
	Java(6)	ExecutionState.INACTIVE
	COBOL(32)	Fmc-AS-Inactive
CheckedOut	Indicates that the activity instance has been checked out.	
	C-language(64)	Fmc_AS_CheckedOut
	C++(64)	FmcjActivityInstance::checkedOut
	Java(7)	ExecutionState.CHECKED_OUT
	COBOL(64)	Fmc-AS-CheckedOut
InError	Indicates that the activity instance has not been executed successfully.	

C-language(128)
Fmc_AS_InError

C++(128) FmcjActivityInstance::inError

Java(8) ExecutionState.IN_ERROR

COBOL(128) Fmc-AS-InError

Executed Indicates that the activity instance has been executed.

C-language(256)
Fmc_AS_Executed

C++(256) FmcjActivityInstance::executed

Java(9) ExecutionState.EXECUTED

COBOL(256) Fmc-AS-Executed

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

C-language(512)
Fmc_AS_Planning

C++(512) FmcjActivityInstance::planning

Java(10) ExecutionState.PLANNING

COBOL(512) Fmc-AS-Planning

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

C-language(1024)
Fmc_AS_ForceFinished

C++(1024) FmcjActivityInstance::forceFinished

Java(11) ExecutionState.FORCE_FINISHED

COBOL(1024) Fmc-AS-ForceFinished

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

C-language(2048)
Fmc_AS_Skipped

C++(2048) FmcjActivityInstance::skipped

Java(12) ExecutionState.SKIPPED

COBOL(2048) Fmc-AS-Skipped

Deleted Indicates that the activity instance has been deleted.

C-language(4096)
Fmc_AS_Deleted

C++(4096) FmcjActivityInstance::deleted

Java(13) ExecutionState.DELETED

COBOL(4096) Fmc-AS-Deleted

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

C-language(8192)
Fmc_AS_Terminating

C++(8192) FmcjActivityInstance::terminating

Java(14) ExecutionState.TERMINATING

COBOL(8192) Fmc-AS-Terminating

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

C-language(16384)
Fmc_AS_Suspending

C++(16384) FmcjActivityInstance::suspending

Java(15) ExecutionState.SUSPENDING

COBOL(16384)
Fmc-AS-Suspending

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

C-language(32768)
Fmc_AS_Expired

C++(32768) FmcjActivityInstance::expired

Java(16) ExecutionState.EXPIRED

COBOL(32768)
Fmc-AS-Expired

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

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

C-language Fmc_AT_NotSet

C++ FmcjActivityInstance::NotSet

Java Type.NOT_SET

COBOL Fmc-AT-NotSet

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

C-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

COBOL Fmc-AT-Program

Block Indicates that the activity instance is implemented by a block.

C-language(16)
Fmc_AT_Block

C++(16) FmcjActivityInstance::Block

Java(3) Type.BLOCK

COBOL(16) Fmc-AT-Block

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

False(0) Indicates that evaluation of the control connector resulted in False.

C-language Fmc_CS_False

C++ FmcjControlConnectorInstance::False

Java EvaluationState.IS_FALSE

COBOL Fmc-CS-False

True(1) Indicates that evaluation of the control connector resulted in True.

C-language Fmc_CS_True

C++ FmcjControlConnectorInstance::True

Java EvaluationState.IS_TRUE

COBOL Fmc-CS-True

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

C-language Fmc_CS_NotEvaluated

C++ FmcjControlConnectorInstance::NotEvaluated

Java EvaluationState.NOT_EVALUATED

COBOL Fmc-CS-NotEvaluated

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

C-language Fmc_CS_NotSet

C++ FmcjControlConnectorInstance::NotSet

Java EvaluationState.NOT_SET

COBOL Fmc-CS-NotSet

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

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

C-language Fmc_CT_NotSet

C++ FmcjControlConnectorInstance::Undefined

Java Type.UNDEFINED

COBOL Fmc-CT-NotSet

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

C-language Fmc_CT_Condition

C++ FmcjControlConnectorInstance::Condition

Java Type.CONDITION

COBOL Fmc-CT-Condition

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

C-language Fmc_CT_Otherwise

C++ FmcjControlConnectorInstance::Otherwise

Java Type.OTHERWISE

COBOL Fmc-CT-Otherwise

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

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

C-language Fmc_DART_NotSet

C++ FmcjExecutionData::NotSet

Java not supported

COBOL Fmc-DART-NotSet

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

C-language Fmc_DART_Terminate

C++ FmcjExecutionData::Terminate

Java not supported

COBOL Fmc-DART-Terminate

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

C-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.

C-language Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

Java not supported

COBOL Fmc-DART-Workitem

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

C-language Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

Java not supported

COBOL Fmc-DART-ActInstNotif

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

C-language Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

Java not supported

COBOL Fmc-DART-ProcInstNotif

ExecuteInstanceResponse(1100)
Indicates that the execution data describes the answer to an asynchronous request which asked for the creation and execution of a process instance.

C-language Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

Java not supported

COBOL Fmc-DART-ExecuteInstResponse

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

C-language Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

Java not supported

COBOL Fmc-DART-ExecuteProgResponse

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

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

C-language Fmc_GM_NotSet

C++ FmcjProgramTemplate::NotSet

COBOL Fmc-GM-NotSet

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

C-language Fmc_GM_Normal

C++ FmcjProgramTemplate::Normal

COBOL Fmc-GM-Normal

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

C-language Fmc_GM_Safe

C++ FmcjProgramTemplate::Safe

COBOL Fmc-GM-Safe

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

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

C-language	Fmc_GU_NotSet
C++	FmcjProgramTemplate::NotSpecified
COBOL	Fmc-GU-NotSet

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

C-language	Fmc_GU_Agent
C++	FmcjProgramTemplate::Agent
COBOL	Fmc-GU-Agent

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

C-language	Fmc_GU_Starter
C++	FmcjProgramTemplate::Starter
COBOL	Fmc-GU-Starter

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

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

C-language	Fmc_EO_NotSet
C++	FmcjExeOptions::NotSet
Java	Style.NOT_SET
COBOL	Fmc-EO-NotSet

Visible(1) Indicates that the EXE should start visibly.

C-language	Fmc_EO_Visible
C++	FmcjExeOptions::Visible
Java	Style.VISIBLE
COBOL	Fmc-EO-Visible

Invisible(2) Indicates that the EXE should start invisibly.

C-language	Fmc_EO_Invisible
C++	FmcjExeOptions::Invisible
Java	Style.INVISIBLE
COBOL	Fmc-EO-Invisible

Minimized(3) Indicates that the EXE should start minimized.

C-language	Fmc_EO_Minimized
C++	FmcjExeOptions::Minimized
Java	Style.MINIMIZED
COBOL	Fmc-EO-Minimized

Maximized(4) Indicates that the EXE should start maximized.

C-language	Fmc_EO_Maximized
C++	FmcjExeOptions::Maximized
Java	Style.MAXIMIZED
COBOL	Fmc-EO-Maximized

External service options time period: This enumeration is named FmcjExternalOptionsTimePeriod in the C-language, FmcjExternalOptions::TimePeriod in C++ and com.ibm.workflow.api.ProgramDataPackage.TimePeriod in Java. Possible values are:

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

C-language	Fmc_EX_NotSet
C++	FmcjExternalOptions::NotSet
Java	TimePeriod.NOT_SET
COBOL	Fmc-EX-NotSet

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

C-language	Fmc_EX_TimeInterval
C++	FmcjExternalOptions::TimeInterval
Java	TimePeriod.TIME_INTERVAL
COBOL	Fmc-EX-TimeInterval

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.

C-language	Fmc_EX_Forever
C++	FmcjExternalOptions::Forever
Java	TimePeriod.FOREVER
COBOL	Fmc-EX-Forever

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

C-language	Fmc_EX_Never
C++	FmcjExternalOptions::Never
Java	TimePeriod.NEVER
COBOL	Fmc-EX-Never

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

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

	C-language	Fmc_DP_NotSet
	C++	FmcjImplementationData::NotSpecified
	Java	Basis.NOT_SET
	COBOL	Fmc-DP-NotSet
OS2(1)	Indicates that the program is an OS/2 program.	
	C-language	Fmc_DP_OS2
	C++	FmcjImplementationData::OS2
	Java	Basis.OS2
	COBOL	Fmc-DP-OS2
AIX(2)	Indicates that the program is an AIX program.	
	C-language	Fmc_DP_AIX
	C++	FmcjImplementationData::AIX
	Java	Basis.AIX
	COBOL	Fmc-DP-AIX
HPUX(3)	Indicates that the program is an HP-UX program.	
	C-language	Fmc_DP_HPUX
	C++	FmcjImplementationData::HPUX
	Java	Basis.HPUX
	COBOL	Fmc-DP-HPUX
Windows95(4)	Indicates that the program is a Windows 95, Windows 98, or Windows Me program.	
	C-language	Fmc_DP_Windows95
	C++	FmcjImplementationData::Windows95
	Java	Basis.WINDOWS_95
	COBOL	Fmc-DP-Windows95
WindowsNT(5)	Indicates that the program is a Windows NT or Windows 2000 program.	
	C-language	Fmc_DP_WindowsNT
	C++	FmcjImplementationData::WindowsNT
	Java	Basis.WINDOWS_NT
	COBOL	Fmc-DP-WindowsNT
OS/390(6)	Indicates that the program is an OS/390 program.	
	C-language	Fmc_DP_OS390
	C++	FmcjImplementationData::OS390
	Java	Basis.OS390
	COBOL	Fmc-DP-OS390
Solaris(7)	Indicates that the program is a Solaris program.	

C-language Fmc_DP_Solaris

C++ FmcjImplementationData::Solaris

Java Basis.SOLARIS

COBOL Fmc-DP-Solaris

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

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

C-language Fmc_DT_NotSet

C++ FmcjImplementationData::NotSet

Java ImplementationData.NOT_SET

COBOL Fmc-DT-NotSet

EXE(1) Indicates that the program is an executable.

C-language Fmc_DT_EXE

C++ FmcjImplementationData::EXE

Java ImplementationData.EXE

COBOL Fmc-DT-EXE

DLL(2) Indicates that the program is implemented by a dynamic link library.

C-language Fmc_DT_DLL

C++ FmcjImplementationData::DLL

Java ImplementationData.DLL

COBOL Fmc-DT-DLL

External Indicates that the program is some external service.

C-language(4) Fmc_DT_External

C++(4) FmcjImplementationData::External

Java(3) ImplementationData.EXTERNAL

COBOL(4) Fmc-DT-External

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

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

C-language Fmc_IS_NotSet

C++ FmcjItem::undefined

Java ExecutionState.UNDEFINED

COBOL Fmc-IS-NotSet

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

C-language Fmc_IS_Ready

	C++	FmcjItem::ready
	Java	ExecutionState.READY
	COBOL	Fmc-IS-Ready
Running(2)	Indicates that the item is in the running state.	
	C-language	Fmc_IS_Running
	C++	FmcjItem::running
	Java	ExecutionState.RUNNING
	COBOL	Fmc-IS-Running
Finished	Indicates that the item is in the finished state.	
	C-language(4)	Fmc_IS_Finished
	C++(4)	FmcjItem::finished
	Java(3)	ExecutionState.FINISHED
	COBOL(4)	Fmc-IS-Finished
Terminated	Indicates that the item is in the terminated state.	
	C-language(8)	Fmc_IS_Terminated
	C++(8)	FmcjItem::terminated
	Java(4)	ExecutionState.TERMINATED
	COBOL(8)	Fmc-IS-Term
Suspended	Indicates that the item is in the suspended state.	
	C-language(16)	Fmc_IS_Suspended
	C++(16)	FmcjItem::suspended
	Java(5)	ExecutionState.SUSPENDED
	COBOL(16)	Fmc-IS-Suspended
Disabled	Indicates that the item is disabled.	
	C-language(32)	Fmc_IS_Disabled
	C++(32)	FmcjItem::disabled
	Java(6)	ExecutionState.DISABLED
	COBOL(32)	Fmc-IS-Disabled
CheckedOut	Indicates that the item is checked out.	
	C-language(64)	Fmc_IS_CheckedOut
	C++(64)	FmcjItem::checkedOut
	Java(7)	ExecutionState.CHECKED_OUT
	COBOL(64)	Fmc-IS-CheckedOut
InError	Indicates that the item is in the InError state.	
	C-language(128)	Fmc_IS_InError

C++(128) FmcjItem::inError

Java(8) ExecutionState.IN_ERROR

COBOL(128) Fmc-IS-InError

Executed Indicates that the item has been executed.

C-language(256)
Fmc_IS_Executed

C++(256) FmcjItem::Executed

Java(9) ExecutionState.EXECUTED

COBOL(256) Fmc-IS-Executed

Planning Indicates that the item is in the planning state.

C-language(512)
Fmc_IS_Planning

C++(512) FmcjItem::Planning

Java(10) ExecutionState.PLANNING

COBOL(512) Fmc-IS-Planning

ForceFinished Indicates that the item has been force-finished.

C-language(1024)
Fmc_IS_ForceFinished

C++(1024) FmcjItem::ForceFinished

Java(11) ExecutionState.FORCE_FINISHED

COBOL(1024) Fmc-IS-ForceFinished

Deleted Indicates that the item has been deleted.

C-language(4096)
Fmc_IS_Deleted

C++(4096) FmcjItem::Deleted

Java(12) ExecutionState.DELETED

COBOL(4096) Fmc-IS-Deleted

Terminating Indicates that the item is in the terminating state.

C-language(8192)
Fmc_IS_Terminating

C++(8192) FmcjItem::Terminating

Java(13) ExecutionState.TERMINATING

COBOL(8192) Fmc-IS-Terminating

Suspending Indicates that the item is in the suspending state.

C-language(16384)
Fmc_IS_Suspending

C++(16384) FmcjItem::Suspending

Java(14) ExecutionState.SUSPENDING

COBOL(16384)
Fmc-IS-Suspending

Expired Indicates that the item is in the expired state.

C-language(32768)
Fmc_IS_Expired

C++(32768) FmcjItem::Expired

Java(15) ExecutionState.EXPIRED

COBOL(32768)
Fmc-IS-Expired

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

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

C-language Fmc_IT_NotSet

C++ FmcjItem::unknown

Java ItemType.UNKNOWN

COBOL Fmc-IT-NotSet

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

C-language Fmc_IT_Workitem

C++ FmcjItem::Workitem

Java ItemType.WORK_ITEM

COBOL Fmc-IT-Workitem

ProcessInstanceNotification
Indicates that the item is a process instance notification.

C-language(3) Fmc_IT_ProcessInstanceNotification

C++(3) FmcjItem::ProcessInstanceNotification

Java(2) ItemType.PROCESS_INSTANCE_NOTIFICATION

COBOL(3) Fmc-IT-ProcInstNotif

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

C-language(4) Fmc_IT_FirstActivityInstanceNotification

C++(4) FmcjItem::FirstActivityInstanceNotification

Java(3) ItemType.FIRST_ACTIVITY_INSTANCE_NOTIFICATION

COBOL(4) Fmc-IT-FirstActInstNotif

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

C-language(5) Fmc_IT_SecondActivityInstanceNotification

C++(45) FmcjItem::SecondActivityInstanceNotification

Java(4) ItemType.SECOND_ACTIVITY_INSTANCE_NOTIFICATION

COBOL(5) Fmc-IT-SecActInstNotif

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

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

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

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

C-language	Fmc_LT_Public
C++	FmcjPersistentList::Public
Java	TypeOfList.PUBLIC
COBOL	Fmc-LT-Public

Private Indicates that the list definition is for private usage.

C-language(3)	Fmc_LT_Private
C++(3)	FmcjPersistentList::Private
Java(2)	TypeOfList.PRIVATE
COBOL(3)	Fmc-LT-Private

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

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

C-language	Fmc_PE_NotSet
C++	FmcjProcessInstance::NotSet
Java	Escalation.NOT_SET
COBOL	Fmc-PE-NotSet

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

C-language	Fmc_PE_NoNotification
C++	FmcjProcessInstance::NoNotification
Java	Escalation.NO_NOTIFICATION
COBOL	Fmc-PE-NoNotif

ProcessInstanceNotification
Indicates that a process instance notification occurred.

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

COBOL(3) Fmc-PE-ProcNotif

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

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

C-language Fmc_PS_NotSet

C++ FmcjProcessInstance::undefined

Java ExecutionState.UNDEFINED

COBOL Fmc-PS-NotSet

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

C-language Fmc_PS_Ready

C++ FmcjProcessInstance::ready

Java ExecutionState.READY

COBOL Fmc-PS-Ready

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

C-language Fmc_PS_Running

C++ FmcjProcessInstance::running

Java ExecutionState.RUNNING

COBOL Fmc-PS-Running

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

C-language(4) Fmc_PS_Finished

C++(4) FmcjProcessInstance::finished

Java(3) ExecutionState.FINISHED

COBOL(4) Fmc-PS-Finished

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

C-language(8) Fmc_PS_Terminated

C++(8) FmcjProcessInstance::terminated

Java(4) ExecutionState.TERMINATED

COBOL(8) Fmc-PS-Term

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

C-language(16) Fmc_PS_Suspended

C++(16) FmcjProcessInstance::suspended

Java(5) ExecutionState.SUSPENDED

COBOL(16) Fmc-PS-Suspended

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

C-language(32)
Fmc_PS_Terminating

C++(32) FmcjProcessInstance::terminating

Java(6) ExecutionState.TERMINATING

COBOL(32) Fmc-PS-Terminating

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

C-language(64)
Fmc_PS_Suspending

C++(64) FmcjProcessInstance::suspending

Java(7) ExecutionState.SUSPENDING

COBOL(64) Fmc-PS-Suspending

Deleted Indicates that the process instance is in the deleted state.

C-language(128)
Fmc_PS_Deleted

C++(128) FmcjProcessInstance::deleted

Java(8) ExecutionState.DELETED

COBOL(128) Fmc-PS-Deleted

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

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

C-language Fmc_WS_NotSet

C++ FmcjWorkitem::NotSet

Java ProgramRetrieval.NOT_SET

COBOL Fmc-WS-NotSet

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

C-language Fmc_WS_CommonDataOnly

C++ FmcjWorkitem::CommonDataOnly

Java ProgramRetrieval.COMMON_DATA_ONLY

COBOL Fmc_WS_CommonDataOnly

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

C-language Fmc_WS_SpecifiedDefinitions

C++ FmcjWorkitem::SpecifiedDefinitions

Java ProgramRetrieval.SPECIFIED_DEFINITIONS

COBOL Fmc_WS_SpecifiedDefs

AllDefinitions

Indicates that all program definitions are to be retrieved.

C-language(4)	Fmc_WS_AllDefinitions
C++(4)	FmcjWorkitem::AllDefinitions
Java(3)	ProgramRetrieval.ALL_DEFINITIONS
COBOL(4)	Fmc_WS_AllDefs

Accessing a value of type integer

Returns the value of a property of type *long*, *unsigned long*, or *int*. Zero (0) is returned if no information is available. In the Java language, an Integer object is returned for optional integer values.

C-language signature

```
long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )

unsigned long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ language signature

```
long Property() const

unsigned long Property() const
```

Java signature

```
public abstract int property() throws FmcException

public abstract Integer property() throws FmcException
```

COBOL

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

Parameters

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

Return type

long/unsigned long/int

The property value.

Declaration examples

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

Java public int priority() throws FmcException;

public java.lang.Integer threshold() throws FmcException;

Accessing a value of type string

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

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
```

COBOL

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

Parameters

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

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

bufferLength Input. The length of the buffer; must be big enough to hold the largest possible value (see file fmcmxcon.h for the minimum required lengths). You can use a single buffer for retrieving all your character values.

Return type

BSTR/char*/string/String

The property value.

Declaration examples

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 C, C++, and COBOL, and as an array in Java. In C++, the collection object to be filled has to be provided by the caller. Use the appropriate accessor API calls to read a single value (refer to "C-language and COBOL vectors" on page 21).

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.

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
```

COBOL

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

Parameters

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

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

Return type

FmcjValueTypeVectorHandle/ValueType[]
The vector or array of values of the property.

Declaration examples

C-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.

C-language signature

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

C++ language signature

```
FmcjObject Property() const
```

Java signature

```
public abstract Object property() throws FmcException

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

public abstract
ExecutionAgent getExecutionAgent() throws FmcException

public
ReadOnlyContainer value() throws FmcException
```

COBOL

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

Parameters

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

system
Input. The system where the execution server runs.

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

Return type

ExecutionAgent
The program execution agent which provides for the context of an activity implementation.

ExecutionService
The execution service which provides for the interface to the execution server.

Object/Handle/FmcjObject
The property value.

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

Declaration examples

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.

C-language signature

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

C++ language signature

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

Java signature

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

COBOL

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

Parameters

handle

Input. The handle of the object to be queried.

Return type

Object*/Handle/FmcjObject*

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

Declaration examples

C-language FmcjReadOnlyContainerHandle FMC_APIENTRY FmcjProgramDataInContainer(FmcjProgramDataHandle handle);

FmcjProcessInstanceHandle FMC_APIENTRY FmcjExecutionServicePersistentProcessInstance(FmcjExecutionService service, char const * oid);

C++ FmcjReadOnlyContainer * InContainer() const;

FmcjProcessInstance * PersistentProcessInstance(string const & oid) const;

Java public abstract ReadOnlyContainer inContainer() throws FmcException;

public abstract ProcessInstance persistentProcessInstance(String oid) throws FmcException;

Determining whether an optional property is set

This API call states whether an optional property is set.

When the property is a secondary property and the object queried is not yet completely read, it is unknown whether the property is set or not so that a default value of true is returned.

Note: Java does not expose IsNull() methods since it is able to return null objects.

C-language signature

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

C++ language signature

```
bool PropertyIsNull() const
```

COBOL

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

Parameters

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

Return type

bool/boolean True if the property is not set, otherwise false.

Declaration examples

C-language bool FMC_APIENTRY FmcjWorkitemDescriptionIsNull(FmcjWorkitemHandle handle);

C++ bool DescriptionIsNull() const;

Setting a value of type integer

This API call sets the specified property to the specified value. In the Java language, an Integer object is to be passed for optional integer values.

C-language signature

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

C++ language signature

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

Java signature

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

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

COBOL

```
      FmcjXxxSetProperty.
           CALL    "FmcjXxxSetProperty"
                          USING
                          BY VALUE
                             handle
                             newValue.
```

Parameters

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

newValue Input. The new value of the property.

Declaration examples

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

C++ void SetTimeout(long newValue) const;

Java public void SetTimeout(int newValue) throws FmcException;

public void setThreshold(Integer threshold) throws FmcException;

An example is the FmcjService::SetTimeout API call which sets the timeout value for requests issued by the client to an MQ Workflow server via this FmcjService object. In other words, it sets the time the client is willing to wait for an answer.

When set, the new timeout value is used for all API calls requiring communication between the client and the server. It can be set (changed) as often as wanted. It is to be provided as milliseconds. A negative value is interpreted as -1, that is, an indefinite timeout.

The default timeout value is taken from the user's profile, from the APITimeOut value; if not found, from the configuration profile. If it is also not found there, the default is 180000 ms.

Note: It is possible that, even though FMC_ERROR_TIMEOUT is returned when you issue a client-server call, the MQ Workflow server has successfully processed the request. However, the server could not send back FMC_OK because communication reported a timeout in the meantime. If the request has not been processed, increase the value set for the timeout and retry the call.

Setting a value of type string

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

C-language signature

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

C++ language signature

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

Java signature

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

COBOL

```
      FmcjXxxSetProperty.
           CALL    "FmcjXxxSetProperty"
                         USING
                         BY VALUE
                            handle
                            newValue
                         RETURNING
                            intReturnValue.
```

Parameters

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

newValue Input. The new value of the property.

Declaration examples

C-language

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

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

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

Setting an object valued property

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

Java signature

```
public abstract void addProperty( Object value )

public abstract
void setContext( String args[], Properties properties )

public abstract
void setContext( Applet applet, Properties properties )
```

Parameters

applet Input. The applet which instantiated the agent. If IIOP is used as communication protocol, providing this information is necessary.

args Input. The command line arguments passed to the application which instantiated the agent bean.

properties Input. The environmental properties passed to the application or applet when it was instantiated.

value Input. The value of the property.

Declaration examples

Java public abstract void addPropertyChangeListener(PropertyChangeListener value);

Updating an object

This API call updates the specified object with information sent from an MQ Workflow server. The update information must have been provided for the specified object.

The server pushes update information for work items - as long as they are not disabled -, activity instance notifications, and process instance notifications. The process setting of the associated process instance must specify REFRESH_POLICY PUSH for that process instance itself or as a process default. Logon must have been performed with a present session mode.

C-language signature

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

C++ language signature

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

COBOL

```
      FmcjXxxUpdate.
           CALL    "FmcjXxxUpdate"
                          USING
                          BY VALUE
                             handle
                             dataValue
                          RETURNING
                             intReturnValue.
```

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

FMC_OK(0) The API call completed successfully.

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

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

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

FMC_ERROR_INVALID_OID(805)
The execution data is no data to update the specified object; it does not belong to the specified object.

FMC_ERROR_WRONG_KIND(501)
The execution data is no data to update the specified object; it is no update data.

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

C-language example: accessing values:

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
  APIRET                       rc;
  FmcjExecutionServiceHandle   service   = 0;
  FmcjWorkitemHandle           workitem  = 0;
  FmcjStringVectorHandle       sList     = 0;
  char                         category[FMC_CATEGORY_NAME_LENGTH+1];
  char                         generalBuffer[200];
  unsigned long                priority  = 0;
  int                          enumValue = 0;
  FmcjCDateTime                startTime;
  unsigned long                i         = 0;
```

Figure 17. Accessing values in C (Part 1 of 9)

```
 FmcjGlobalConnect();
```

Figure 17. Accessing values in C (Part 2 of 9)

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

Figure 17. Accessing values in C (Part 3 of 9)

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

Figure 17. Accessing values in C (Part 4 of 9)

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

Figure 17. Accessing values in C (Part 5 of 9)

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

Figure 17. Accessing values in C (Part 6 of 9)

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

Figure 17. Accessing values in C (Part 7 of 9)

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

Figure 17. Accessing values in C (Part 8 of 9)

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

Figure 17. Accessing values in C (Part 9 of 9)

C++ example: accessing values:

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

Figure 18. Accessing values in C++ (Part 1 of 6)

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

Figure 18. Accessing values in C++ (Part 2 of 6)

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

Figure 18. Accessing values in C++ (Part 3 of 6)

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

Figure 18. Accessing values in C++ (Part 4 of 6)

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

Figure 18. Accessing values in C++ (Part 5 of 6)

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

Figure 18. Accessing values in C++ (Part 6 of 6)

COBOL example: accessing values:

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

       DATA DIVISION.

         WORKING-STORAGE SECTION.

         COPY fmcvars.
         COPY fmcconst.
         COPY fmcrcs.

         01 localUserID   PIC X(30) VALUE z"USERID".
         01 localPassword PIC X(30) VALUE z"PASSWORD".
         01 categBuffer   PIC X(34).
         01 generalBuffer PIC X(200).
         01 i             PIC 9(9) BINARY VALUE 0.

         LINKAGE SECTION.

         01 retCode            PIC S9(9) BINARY.

       PROCEDURE DIVISION USING retCode.

            PERFORM FmcjGlobalConnect.
      * logon
            PERFORM FmcjESAllocate.

            CALL "SETADDR" USING localUserId userId.
            CALL "SETADDR" USING localPassword passwordValue.
            MOVE Fmc-SM-Default TO sessionMode.
            MOVE Fmc-SA-Reset TO absenceIndicator.
            PERFORM FmcjESLogon.
      * set the timeout for requests
            MOVE 60000 TO newTimeOutValue.
            PERFORM FmcjESSetTimeout.
```

Figure 19. Accessing values in COBOL (via PERFORM) (Part 1 of 3)

```
* assumption: workitem has been queried from the server
 * and hdlItem points to this workitem

 * access a value of type bool (PIC 9 BINARY)
       PERFORM FmcjWICategIsNull.
       IF boolReturnValue = 1
          DISPLAY "Category is not set"
       ELSE
 * access a value of type char (POINTER to a PIC X(n))
 *        use a buffer which fits
          CALL "SETADDR" USING categBuffer categoryNameBuffer
          MOVE FMC-CATEG-NAME-LENGTH TO bufferLength
          PERFORM FmcjWICateg
          DISPLAY "Category   : " categBuffer
       END-IF
 * access a date/time value
       PERFORM FmcjWIStartTime.
       MOVE dateTimeReturnValue TO timeValue.
       CALL "SETADDR" USING generalBuffer dateTimeBuffer.
       MOVE 200 TO bufferLength.
       PERFORM FmcjDateTimeAsString.
       DISPLAY "Start time : " generalBuffer.
 * access a value of type unsigned long (PIC 9(9) BINARY)
       SET hdlWorkitem TO hdlItem.
       PERFORM FmcjWIPriority.
       DISPLAY "Priority   : " ulongReturnValue.
```

Figure 19. Accessing values in COBOL (via PERFORM) (Part 2 of 3)

```
    * access an enumerated value (PIC S9(9) BINARY)
       PERFORM FmcjWIReceivedAs.
       IF intReturnValue = Fmc-IR-Normal
       DISPLAY "Received as: qualified user"
       END-IF
 * access a multi-valued field
       PERFORM FmcjWISupportTools.
       SET hdlVector TO FmcjStrVHandleReturnValue.
       PERFORM FmcjStrVSize.
       DISPLAY "Support tools: ".
 *     use a large buffer
       CALL "SETADDR" USING generalBuffer elementBuffer
       PERFORM VARYING i FROM 0 BY 1 UNTIL i >= ulongReturnValue
          PERFORM FmcjStrVNextElement
          DISPLAY generalBuffer
       END-PERFORM
 * logoff
       PERFORM FmcjESLogoff.
       DISPLAY "Logged off".
       PERFORM FmcjESDeallocate.
       PERFORM FmcjGlobalDisconnect.
       MOVE FMC-OK TO retCode.
       GOBACK.

       COPY fmcperf.
```

Figure 19. Accessing values in COBOL (via PERFORM) (Part 3 of 3)

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

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 categBuffer   PIC X(34).
   01 generalBuffer PIC X(200).
   01 i             PIC 9(9) BINARY VALUE 0.

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      CALL "FmcjGlobalConnect".
* logon
      CALL "FmcjExecutionServiceAllocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      CALL "FmcjExecutionServiceLogon"
         USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
         RETURNING intReturnValue.

* set the timeout for requests
      MOVE 60000 TO newTimeOutValue.
      CALL "FmcjServiceSetTimeout"
         USING BY VALUE serviceValue
                        newTimeoutValue.

* assumption: workitem has been queried from the server
* and hdlItem points to this workitem

* access a value of type bool (PIC 9 BINARY)
      CALL "FmcjItemCategoryIsNull"
         USING BY VALUE hdlItem
         RETURNING boolReturnValue.
      IF boolReturnValue = 1
         DISPLAY "Category is not set"
      ELSE
* access a value of type char (POINTER to a PIC X(n))
*        use a buffer which fits
         CALL "SETADDR" USING categBuffer categoryNameBuffer
         MOVE FMC-CATEG-NAME-LENGTH TO bufferLength
         CALL "FmcjItemCategory"
```

Figure 20. Accessing values in COBOL (via CALL) (Part 1 of 3)

```
            USING BY VALUE hdlItem
                           categoryNameBuffer
                           bufferLength
            RETURNING pointerReturnValue
         DISPLAY "Category   : " categBuffer
      END-IF

* access a date/time value
      CALL "FmcjItemStartTime"
         USING BY VALUE hdlItem
         RETURNING dateTimeReturnValue.
      CALL "SETADDR" USING generalBuffer dateTimeBuffer.
      MOVE 200 TO bufferLength.
      CALL "FmcjDateTimeAsString"
         USING BY REFERENCE dateTimeReturnValue
               BY VALUE     dateTimeBuffer
                            bufferLength
         RETURNING pointerReturnValue.
      DISPLAY "Start time : " generalBuffer.
* access a value of type unsigned long (PIC 9(9) BINARY)
      SET hdlWorkitem TO hdlItem.
      CALL "FmcjWorkitemPriority"
         USING BY VALUE hdlItem
         RETURNING ulongReturnValue.
      DISPLAY "Priority   : " ulongReturnValue.
* access an enumerated value (PIC S9(9) BINARY)
      CALL "FmcjItemReceivedAs"
         USING BY VALUE hdlItem
         RETURNING intReturnValue.
       IF intReturnValue = Fmc-IR-Normal
         DISPLAY "Received as: qualified user"
      END-IF
* access a multi-valued field
      CALL "FmcjWorkitemSupportTools"
         USING BY VALUE hdlItem
         RETURNING FmcjStrVHandleReturnValue.
      SET hdlVector TO FmcjStrVHandleReturnValue.
      CALL "FmcjStringVectorSize"
         USING BY VALUE hdlVector
         RETURNING ulongReturnValue.
      DISPLAY "Support tools: ".
*     use a large buffer
      CALL "SETADDR" USING generalBuffer elementBuffer
      PERFORM VARYING i FROM 0 BY 1 UNTIL i >= ulongReturnValue
      CALL "FmcjStringVectorNextElement"
         USING BY VALUE hdlVector
                        elementBuffer
                        bufferLength
         RETURNING pointerReturnValue
         DISPLAY generalBuffer
      END-PERFORM
* logoff
      CALL "FmcjExecutionServiceLogoff"
         USING BY VALUE serviceValue
         RETURNING intReturnValue.
      DISPLAY "Logged off".
      CALL "FmcjExecutionServiceDeallocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      CALL "FmcjGlobalDisconnect".
```

Figure 20. Accessing values in COBOL (via CALL) (Part 2 of 3)

```
MOVE FMC-OK TO retCode.
GOBACK.
```

Figure 20. Accessing values in COBOL (via CALL) (Part 3 of 3)

Action API calls

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

Action API calls can be issued on service objects and on transient objects representing persistent ones. These objects remember the context of a user session so that a communication path to an MQ Workflow server can be established. As a consequence, empty objects cannot be used in order to issue action calls.

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

All action API calls are described separately in "Chapter 5. API action and activity implementation calls" on page 317. You can find examples in "Chapter 6. Examples" on page 575.

Activity implementation API calls

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

A program implementing an activity is usually executed under the control of an MQ Workflow program execution server on request from some MQ Workflow execution server. When an MQ Workflow execution server receives a request to start a work item, it determines the implementing program to be started and sends an appropriate request together with the input and output containers, if needed, to the program execution server. Since containers are sent to the program execution server, input and output containers are requested from and returned to an MQ Workflow program execution server 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.

However, if you want to access not only containers, for example, if you want to query information about the process instance the work item is a part of, you have to log on to the MQ Workflow execution server that requested to start your program. You can use the Passthrough() API call of the execution service to begin a session with the execution server from within the activity implementation program. This way, you can use the environment of the work item, that is, you do not need any other user ID, password, system group, or system information.

An MQ Workflow program execution server 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.

Accessing general information

An activity implementation can query the user on whose behalf the activity implementation was started or the persistent object identification of the associated activity instance(class name FmcjPEA or ExecutionAgent in Java).

C-language signatures

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

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

C++ language signatures

```
static string PersistentOidOfActivityInstance()

sstatic string UserID()
```

Java signatures

```
public abstract
String persistentOidOfActivityInstance() throws FmcException

public abstract
String userID() throws FmcException
```

COBOL

```
      FmcjPEAPersOidOfActInst.
           CALL    "FmcjPEAPersistentOidOfActivityInstance"
                          USING
                          BY VALUE
                             oidBuffer,
                             bufferLength
                          RETURNING
                             pointerReturnValue.
      FmcjPEAUserID.
           CALL    "FmcjPEAUserID"
                          USING
                          BY VALUE
                             userIdBuffer,
                             bufferLength
                          RETURNING
                             pointerlReturnValue.
```

Parameters

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

bufferLength Input. The length of the buffer; must be big enough to hold the largest possible value (see file fmcmxcon.h for the minimum required lengths). You can use a single buffer for retrieving all your character values.

Return type
BSTR/char*/string/String
The value.

Return codes
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a buffer is expected, but 0 is passed.
FMC_ERROR_BUFFER(800)
The buffer provided is too small to hold the largest possible value. See file fmcmxcon.h for required lengths.
FMC_ERROR_COMMUNICATION(13)
The program execution agent cannot be contacted.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative. Also, check the MQ Workflow trace for any exceptions encountered.
FMC_ERROR_PROGRAM_EXECUTION(126)
The API call is not called from within an activity implementation.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call. For example, the activity implementation is not trusted and thus cannot retrieve its program ID.

See “An activity implementation” on page 622 or the ″API Programming Examples″ support pack for activity implementation examples.

Chapter 2. Language interfaces

This chapter discusses language-specific considerations for using the API. It also describes the XML message interface in detail.

C and C++ interface

This section provides an overview of the concepts which are specific to the MQ Workflow C and C++ APIs.

An MQ Workflow client application

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

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

Actions         {   MQ Workflow API calls

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

To set up your program, you typically declare the program variables or objects you are going to use and you include the MQ Workflow API header files you need for your actions. When using the C++ API, definitions of *bool*, *string*, and *vector* are needed. Include the respective files before the MQ Workflow API headers.

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

You then need to allocate a service object which represents the server you are going to ask services from. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your service object. All subsequent calls requiring client/server communication run through this session.

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

At the end of your program, you log off in order to close the session to the server and you deallocate any resources held by your program, especially the service object.

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

An MQ Workflow activity implementation

An MQ Workflow C or C++ activity 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() API call so that resources held by the API are allocated correctly. Connect() - and Disconnect() - are to be called at the begin respectively end of each thread.

An activity implementation can then retrieve the activity's input and output containers from the MQ Workflow program execution server that started this program.

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQ Workflow program execution agent. Any resources held by your program are deallocated. The return value of your program tells the program execution agent about the overall outcome of your program.

The output container as well as the return code of your program are passed back to the MQ Workflow server which requested the execution of the activity

implementation. The return code (_RC) can be used in exit or transition conditions in order to guide MQ Workflow navigation. [5]

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

Your activity implementation can as well behave like a client application (see "An MQ Workflow client application" on page 137) and request services from an MQ Workflow server, normally the server from where its execution had been triggered. The Passthrough() API call is then used instead of the Logon() API call in order to log on to the server which caused the program execution with the user identification and authority known to the server from the work item start request.

Note: Legacy applications in CICS or IMS used as activity implementations, get input and output containers passed in their COMMAREA or I/O Area when they are started by the PES.

Compiling and linking

C and C++ programs for MQ Workflow for OS/390 must be compiled with IBM C/C++ for OS/390 Version 2 Release 4 or later.

All C-language and C++ programs developed for use with MQ Workflow must include header files provided by MQ Workflow and link the corresponding library files.

When using the MQ Workflow C++ API, definitions for bool, string, and vector must be provided. MQ Workflow delivers some files to be included: *bool.h* which provides for the bool definition and must be included first, *fmcjstr.hxx* which provides for the string definition, and *vector.h* which provides for the vector definition.

Note that bool.h and vector.h are part of the Standard Template Library delivered with MQ Workflow and copyrighted[6] by the Hewlett-Packard Company. Documentation of this library is provided on the MQ Workflow CD-ROM (non-390 version) in a file named STLDOC.PS. It is installed in the stl subdirectory of the API.

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

5. For compilers which do not support an exit code of an application, it is possible to set the _RC data member of the output container.

6. Copyright (c) 1994

Hewlett-Packard Company

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

Feature	C-API Header	C++ Header
Runtime support tool		
- container access only	fmcjccon.h	fmcjpcon.hxx
- container and server access	fmcjcrun.h	fmcjprun.hxx

The following JCLs are provided as samples for the development and execution of IBM MQSeries Workflow for OS/390 applications. They are located in the SFMCCNTL library delivered with MQSeries Workflow.

Table 3. JCLs provided for C/C++ programs

Job	Sample
Native OS/390 C/C++ Full API compile job	FMCHJ1CF
Native OS/390 C/C++ API run job	FMCHJ1CR
CICS C/C++ Full API compile job	FMCHJ2CF
CICS C/C++ Container API compile job	FMCHJ2CC
IMS C/C++ Container API compile job	FMCHJ3CC

For more information about CICS/IMS specifics like stubs or precompiler, refer to the documentation for these components.

The compilers given as prerequisites or newer versions can be used to compile and link your applications accessing the MQ Workflow API. Your compile and link options must ensure that the MQ Workflow API is called with the calling convention that is defined in the FMC_APIENTRY macro (see file fmcjcglo.h). FMC_APIENTRY has been defined to the standard C calling convention and will automatically be applied when you use the header files provided by MQ Workflow.

Access can be gained to C-language functions using calls from all languages that support C calls.

Supplied sample programs

The following samples are provided in InstHLQ.SFMCSRC:

FMCHSCFA
C Full API sample (native OS/390)

FMCHSCCA
C Container API sample (CICS and IMS)

Java interface

Note: The following section describes the general Java environment supported by the LAN version of MQ Workflow. Only selected components of this environment, namely the local bindings, are currently implemented in the OS/390 version.

The MQ Workflow Java API consists of:

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

For a detailed overview of the Java API Classes and the Java CORBA Agent, refer to the *IBM MQSeries Workflow: Installation Guide*.

Threading considerations for the Java CORBA Agent

The MQ Workflow C and C++ programming languages offer API calls FmcjGlobalConnect()/FmcjGlobal::Connect() and FmcjGlobalDisconnect()/FmcjGlobal::Disconnect() which are to be called whenever the application starts or ends a thread. Framing a thread with these calls guarantees that communication with the MQ Workflow execution server can be handled properly by the API. There are, however, no such calls in the MQ Workflow Java API.

Communication between the MQ Workflow APIs and the MQ Workflow servers requires one MQSeries connection per thread. For performance reasons, these connections are cached in the C-language layer of the MQ Workflow API, which is part of the JNI code. MQSeries connections are managed through handles which are owned by the thread that opened the connection. That is, each thread has to acquire its own connection handle; it cannot use the handles of other threads. When a thread ends, the connection has to be closed for the MQSeries resources to be freed. If an application fails to close the connections, MQSeries has built-in mechanisms to nevertheless reclaim the resources after some time. But this mechanism is not always fast enough to free the resources in time. There are even configurations where this is not possible at all.

When the MQ Workflow Java API is used in thread-pooling environments (for example, when it is running as an RMI-IIOP Agent or in a Servlet container), not freeing resources can cause the MQSeries Listener to run out of resources and you will receive error messages like ″Maximum number of channels reached″ or ″Channel program terminated abnormally″. The problem encountered here is that the thread pool is managed outside of MQ Workflow (for example, by a Servlet or EJB container) and usually provides no hooks for thread creation or thread destruction. The MQ Workflow APIs can detect if they are invoked from a new worker thread. When the API is invoked from a new worker thread, there is no connection handle and a new handle is created. But there is no mechanism to get informed when a worker thread ends.

Although it is possible to periodically check if the thread associated with a connection handle ended, it is too late to issue a Disconnect() since MQSeries connection handles are thread specific and the thread already ended. Also, since none of the thread-pool owners maintains an affinity between a client and a worker thread, offering the Connect() and Disconnect() methods in the MQ Workflow Java API does not help to control communication from within threads.

Running out of channels can be avoided by using the MQSeries server interface (MQI) instead of the client interface (MQIC) in such configurations. This setup allows MQSeries to reclaim its resources without the application having to specifically call disconnect. Using MQI requires a local Queue Manager to be configured on the same machine which is a so-called *client concentrator* setup. The MQ Workflow Configuration automatically uses a client concentrator setup for these types of configurations. Details about the client concentrator setup can be found in the *IBM MQSeries Workflow: Installation Guide*.

JNDI locator policy

The JNDI locator policy normally uses the built in ORB of the Java 2 runtime environment. This ORB, as most other ORBs, manages a pool of worker threads and offers no mechanisms to hook the thread control methods. Thus, using the

JNDI locator policy requires an MQSeries server installation and a client concentrator setup on the machine where the Java CORBA Agent is running. Details about the client concentrator setup can be found in the *IBM MQSeries Workflow: Installation Guide*.

OSA, IOR, and COS locator policies

Note: Using the OSA, IOR, and COS locator policies is deprecated.

If you use the OSA, IOR, or COS locator policies, you can specify the number of threads in the thread pool.

The Visibroker ORB uses a thread pool as the default threading policy when accessing the Java CORBA Agent via one of the CORBA locator policies, that is, via a Visibroker Smart Agent (OSA), an Interoperable Object Reference (IOR), or a COS Naming Service (COS). To overcome running out of channels you should set the maximum number of threads, **OAthreadMax**, and the minimum number of threads, **OAthreadMin**, to the same value so that no worker threads are dynamically created or ended depending on the workload. This applies to all operating system platforms.

Note that this kind of setup may constantly require a lot of system resources. It can also be challenging to determine the optimum number of threads to handle the workload. Thus, the recommendation is to have a client concentrator setup.

To specify the number of threads opened, edit the Java CORBA Agent startup script (batch file) for a specific configuration and add the following statements to the end of the line containing "com.ibm.workflow.agent.Main":

```
-OAthreadMin xxx -OAthreadMax xxx    where xxx denotes the number of threads
```

For example, on Windows NT for configuration TTT:

```
java -Xbootclasspath: ... -classpath ... com.ibm.workflow.agent.Main -yTTT
                should be changed to
java -Xbootclasspath: ... -classpath ... com.ibm.workflow.agent.Main -yTTT
                          -OAthreadMin xxx -OAthreadMax xxx [7]
```

For further details on the Visibroker ORB command line parameters see the documentation that comes with the Visibroker for Java.

RMI locator policy

Using the RMI locator policy is adequate for applications that do not handle a large number of concurrent users, like test and prototyping environments. The RMI specification does not define a specific thread-dispatching policy so that each RMI implementation shows different runtime characteristics.

For example, the standard implementation suffers from several shortcomings:

- There is no configurable upper limit on the number of server (agent) threads to handle client requests. Thus the server (agent) could be overwhelmed with requests and thereby exhaust its resources. Developing a solution to this problem is difficult at best because RMI does not provide a hook for implementing custom threading policies as it does for custom sockets.
- Threads are not reused so that resources are wasted.

Note: Using the RMI locator policy is deprecated.

7. The line is split for printing purposes only.

Microsoft JVM/Internet Explorer V4/V5 and RMI

Although Microsoft does not officially support RMI, a file named rmi.zip' can be found on Microsoft's ftp site (ftp://ftp.microsoft.com/developr/MSDN/UnSup-ed/rmi.zip). Note that 'UnSup-ed' means 'unsupported'.

For Windows NT, this file must be extracted into the \Winnt\Java\Trustlib directory, for Windows 95 or Windows 98 into \Windows\Java\Trustlib. Add rmi.zip to the TrustedClasspath in the registry:

```
REGEDIT4 [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Java VM]
        "TrustedClasspath"="c:\winnt\java\trustlib\rmi.zip;..."
```

An MQ Workflow client application

An MQ Workflow Java client application typically contains the following parts, not necessarily divided that clearly.

```
          ⎧    import java packages
          ⎪    void main()
          ⎪    {
          ⎪      Declare objects
Setup     ⎨            :
          ⎪      Allocate agent
          ⎪      set Locator or Configuration / set Name
          ⎪      locate service object
          ⎩      Logon()

          ⎧
          ⎪
          ⎪
Actions   ⎨      MQ Workflow API calls
          ⎪
          ⎪
          ⎩
          ⎧      Logoff()
Cleanup   ⎩    }
```

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.

An MQ Workflow activity implementation

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

```
                 import java packages
                 void main()
                 {
                   Declare objects
                         :
Setup            Allocate agent
                 set Locator / set Name
                 get execution agent
                 InContainer()
                 OutContainer() if no support tool


Actions          read values
                 set values if no support tool


Cleanup          SetOutContainer() if no support tool
                 System.exit(rc)
                 }
```

To set up your program, you typically declare the program variables or objects you are going to use and you import the MQ Workflow Java API packages you need for your actions.

You then need to locate your execution agent object. You do this by allocating and asking the appropriate agent.

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

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQ Workflow program execution agent. The return value of your program or the _RC value in the output container tells the program execution agent about the overall outcome of your program.

The output container is passed back to the MQ Workflow server which requested the execution of the activity implementation. The return code (_RC) can be used in exit or transition conditions in order to guide MQ Workflow navigation.

Your activity implementation can as well behave like a client application (see "An MQ Workflow client application" on page 137) and request services from an MQ Workflow server, normally the server from where its execution had been triggered. The Passthrough() method is then used instead of the Logon() method in order to log on to the server which caused the program execution with the user identification and authority known to the server from the work item start request.

Note: An activity implementation currently supports the LOC_LOCATOR policy only.

Refer also to

```
<MQ Workflow installation directory>\smp\java\actimpl\ActImpl.java
```

for an example of an activity implementation in Java.

Compiling

All programs developed for use with the MQ Workflow Java API Classes must import the packages provided by MQ Workflow. These files have been installed on your system if you selected to install the MQ Workflow Development Kit. They are installed by default in the \bin\java3300 subdirectory of the installation directory.

JDK 1.3 can be used to compile and run your applications accessing the MQ Workflow Java API. A sample compile statement is:

```
javac <java file>.java
```

The CLASSPATH must include either fmcojagt.jar or fmcojapi.jar. Make sure that only one of the jar files is included in the CLASSPATH since fmcojagt.jar is a strict superset of fmcojapi.jar. fmcojapi.jar contains all client side stubs for the JNDI, RMI, OSA, COS, and IOR locator policies. fmcojagt.jar additionally contains the corresponding skeletons and the LOC locator policy.

JNDI locator policy

When using the JNDI locator policy, the naming context factory class and the URL of the JNDI Naming Service have to be specified. The *IBM MQSeries Workflow: Installation Guide* describes how these parameters are set on the command line. Alternatively, the parameters can also be set in the application. For example:

```
public static void main ( String[] args )
{
...
Agent       agent= new Agent();
Properties  prop = System.getProperties();

prop.put( javax.naming.Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.cosnaming.CNCtx.Factory") ;
prop.put( javax.naming.Context.PROVIDER_URL,
          "iiop://<Hostname of JNDI Name Server>:<JNDI Name Server Ports"" );

agent.setContext( args, prop );
agent.setLocator( Agent.JNDI_LOCATOR );
agent.setName( "<AgentName>");
...
}
```

The previous example shows a COS Naming implementation of the JNDI specification. Other JNDI service providers, for example, an LDAP server, can require the specification of additional or different properties to perform a lookup for objects.

How to use the MQ Workflow Java API from within IBM VisualAge for Java

1. If you are using the Inprise VisiBroker for Java ORB, you need to create it first:
   - In the Workbench, add a project for the ORB, for example, “VisiBroker ORB”.
   - Import the files vbjorb.jar and vbjcosnm.jar into the newly created project. You can ignore any problems concerning the package ’com.visigenic.vbroker.CORBA’.
2. Create a project for the MQ Workflow JAVA API, for example, “MQWF Java API”, and import the file fmcojagt.jar into that project. If you did not import the VisiBroker ORB first because you are using local, JNDI, or RMI bindings, you will encounter several problems in the com.ibm.workflow.corba package which you can ignore.

Note that importing the MQ Workflow Java API into VisualAge for Java is necessary. It is not sufficient to enter fmcojagt.jar in the “Window” - “Options...” - “Resources” - “Workspace class path:” field.

3. Create your actual project which will use the MQ Workflow Java APIs, for example, ’MyProject’. Write your program as usual, for example, a class HelloWorld which contains the public ’main’ method.
4. Before running your program, select ’Properties’ on your HelloWorld class. Open the “Classpath” page and select “Project path”. Select “Edit”, then select the “VisiBroker ORB”, if necessary, and “MQWF Java API” packages to be included in your project classpath. Now you are ready to run your application.

Running the MQ Workflow Java CORBA Agent inside the WebSphere Test Environment

If you want to start the MQ Workflow Java CORBA Agent inside the WebSphere Test Environment of Visual Age for Java, you have to perform the following steps:

1. Install and configure the Java CORBA Agent as usual.
2. Create a project for the Java CORBA Agent as described above.
3. In the ″Projects″ tab, select your Java CORBA Agent project. Select the package ″com.ibm.workflow.agent″ and then the class ″Main″. Right-click on the ″Main″ class and select ″Properties″ from the context menu.
4. In the ″Program″ tab go to the ″Command line arguments″ field and enter ″-y <name of your configuration>. For example, ″-y FMC1″.
5. In the ″Class Path″ tab, click on the ″Edit″ button of the Project path and select ″IBM WebSphere Test Environment″. If required by your application, select additional projects. Click ″OK″ to continue.
6. Click on the ″Edit″ button of the Extra directories path. Click on the ″Add Jar/Zip″ button and select the file ″fmcojagt.jar″ located in the directory ″<MQ Workflow installation directory>\bin\java330\″. Click on ″Open″ and then ″OK″ to continue.
7. Make sure that ″Save in repository (as default)″ is checked and click on ″OK″ to finish.
8. To start the Java CORBA Agent you have to start the WebSphere Test Environment first. If you configured the Java CORBA Agent for the JNDI locator policy, you also have to start the Persistent Name Server. Start the Java CORBA Agent by right-clicking the class ″Main″ in the package ″com.ibm.workflow.agent″ and then clicking ″Run″ followed by ″Run main″ in the context menu.
9. You can stop the Java CORBA Agent by selecting it and clicking ″Terminate″ in the WebSphere Test Environment Console.

Troubleshooting

As a general rule of thumb: Whenever you experience problems with the MQ Workflow Java API:

- Turn off the JIT (Just in Time) compiler.
- Make sure you are using the most recent service level of the Java Development Kit.
- Consult your JDK/JRE Application Server documentation for detailed guidelines.

Following these recommendations is especially important when you experience exceptions providing a stack trace. Stack traces can be inaccurate when used with the JIT Compiler; the JIT Compiler may have optimized your code in a way that

method calls no longer appear in the stack trace. Therefore, when you contact the MQ Workflow support team, provide only stack traces with the JIT Compiler turned off.

Object management

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

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

The lifetime of transient objects is *fully managed* by you, because you know best when those objects are no longer needed, that is, when objects are unreferenced. Transient objects are, however, no longer available when your application program ends.

Some transient objects are *explicitly allocated* by you. These are supporting objects, which do not reflect persistent ones. An example is the Agent object, which allows to obtain a reference to an ExecutionService object.

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

Although the life time of transient objects is fully managed by you, their actual internal object structure is encapsulated by the MQ Workflow API.

As all resource memory is finally owned by the application process itself, you can access all objects from different threads within that process. MQ Workflow does not hinder you from using threads; it is coded reentrantly. On the other hand, the MQ Workflow Java API explicitly supports threads for action methods only. That is, all action methods are synchronized. Accessor methods are not synchronized because they normally read values only. When, however, the accessed value is not yet available in the API cache, the object is automatically refreshed from the server so that you need to synchronize parallel accesses on that object. Refer also to the discussions on “Accessor/mutator API calls” on page 92 and “Stateless server support” on page 72.

Garbage Collection when using Java API classes

Garbage collection is normally running in the background without intervention by the Java programmer. This is also true in a distributed Java environment when objects communicate via the RMI transmission protocol. However, for other protocols, like CORBA’s IIOP, provisions to remove nonreferenced objects on the agent side have to be made. When CORBA is used, then the memory management implicitly run by a Java Virtual Machine does not synchronize object removal on a client and the agent. Agent-side pendants of not referenced client objects are not automatically marked for removal. The Object Request Broker (ORB) cannot determine if any client is holding or not holding references to objects that it has registered (some ORBs, in fact, can do that, however, they are using proprietary CORBA extensions to achieve this). Agent-side pendants of client objects registered with an ORB by using a connect method have to be disconnected explicitly. When using MQ Workflow Java API classes, the user is provided with a build-in garbage collection mechanism, the MQ Workflow Java API Classes Reaper, that does

housekeeping when the transmission of data is done by a CORBA Object Request Broker (ORB). Before starting the MQ Workflow Java API CORBA Agent a set of parameters controlling the reaper have to be set. These control parameters are:

- The reaper cycle time value, defined in milliseconds, is valid for both the client's reaper and the server's reaper. Default value is 300000 msec.
- The reaper threshold value is set to determine a maximum count for accumulated objects that are no longer referenced. The threshold takes precedence over the cycle time. Default value is 1000.
- The reaper ratio defines the relation between cycle times of both, client side reaper and server side reaper. The ratio is used as a multiplier for the server's reaper cycle, to calculate the cycle time for the client's reaper. The default value is 90, that means in fact 90% of the server's reaper cycle time. This ensures that the client side reaper actions always precede the server's side reaper actions.

The parameters are initially set at configuration time.

COBOL interface

The COBOL API runs as a layer on top of the C API, that is, the COBOL CALL statements are effectively translated into the C functions with the same name. For this reason, what is said about the C API in this book generally applies to COBOL as well.

There are two ways to use the COBOL API:

1. directly through the Language Environment (LE) ″CALL″ mechanism.
2. by using the FMCPERF copybook and the COBOL ″PERFORM″ mechanism, which in turn uses the ″CALL″ mechanism.

Note: The COBOL calling sequences illustrated in the individual API descriptions are excerpts from FMCPERF.CPY. The paragraph name shown in each case is the name to be used as the PERFORM operand.

Calling the API

The COBOL API uses Language Environment (LE) interlanguage calls (ILCs) to call the C API. Therefore the compiler option PGMNAME(LM) must be used to allow

a) calls to functions with names longer than 30 characters

b) case sensitivity in function names

String handling

Since C strings (char *) are null-terminated, the COBOL programmer must provide null-terminated string parameters. String output parameters must be checked for the first occurrence of X'00' to get the correct value. A function cannot be called with String/PIC X(n) constants but rather must use a pointer referencing such a null-terminated PIC X(n).

Coding an MQ Workflow client application in COBOL

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

```
                IDENTIFICATION DIVISION.
                PROGRAM-ID. "SAMPAPPL"
                   :
                ENVIRONMENT DIVISION.
                   :
                DATA DIVISION.
                WORKING-STORAGE SECTION.

Setup           copy fmcrcs.
                copy fmcvars.
                copy fmcconst.

                Declare objects

                PROCEDURE DIVISION.
                  Connect
                  Allocate service object
                  Logon

Actions           MQSeries Workflow API calls

Cleanup           Logoff
                  Deallocate service object
                  Disconnect
                  GOBACK.

                  copy fmcperf.
                END PROGRAM "SAMPAPPL"
```

To set up your program, you typically declare the program variables or objects you are going to use and copy the MQSeries Workflow API copybooks you need for your actions.

You should then initialize the MQ Workflow API via the Connect API call so that resources held by the API are allocated correctly. Connect and Disconnect are to be called at the beginning and end of each thread, respectively.

You then need to allocate a service object which represents the server you are going to ask services from. Once the service object is allocated, you can log on. Logon establishes a session between the user logging on and the server represented by your service object. All subsequent calls requiring client/server communication run through this session.

After a successful logon, you can issue action or program execution management API calls in order to query or manage 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.

Coding an MQ Workflow activity implementation in COBOL

An MQ Workflow or activity implementation typically contains the following parts:

```
                IDENTIFICATION DIVISION.
                PROGRAM-ID. "SAMPAI"
                   :
                ENVIRONMENT DIVISION.
                   :
                DATA DIVISION.
                WORKING-STORAGE SECTION.

Setup             copy fmcrcs.
                  copy fmcvars.
                  copy fmcconst.

                  Declare objects

                PROCEDURE DIVISION.
                  Connect
                  InContainer
                  OutContainer

Actions           read values
                  set values

                  SetOutContainer
Cleanup           Deallocate object
                  Disconnect
                  GOBACK.

                  copy fmcperf.
                END PROGRAM "SAMPAI"
```

To set up your program, you typically declare the program variables or objects you are going to use and copy the MQSeries Workflow API copybooks you need for your actions.

You should then initialize the MQ Workflow API via the Connect API call so that resources held by the API are allocated correctly. Connect and Disconnect are to be called at the beginning and end of each thread, respectively.

An activity implementation can then retrieve the activity's input and output containers from the MQ Workflow program execution server that started this program.

Having access to the containers, you can read and set values according to your programming logic.

At the end of your program, the activity implementation returns the final output container to the MQ Workflow program execution server. Any resources held by your program are deallocated. The return value tells the program execution server about the overall outcome of your program.

The output container as well as the return code of your program are passed back to the 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 also act like a client application (see "An MQ Workflow client application" on page 137) and request services from an MQ Workflow server, normally the server from where its execution was triggered. The Passthrough API call is then used instead of Logon in order to log on to the server which initiated program execution, with the user identification and authority known to the server from the work item start request.

Compiling and linking

COBOL programs must be compiled using IBM COBOL for OS/390 and VM, Version 2 Release 1 or higher.

The following copybooks are delivered with MQSeries Workflow for OS/390:

Table 4. Copybooks provided for COBOL programs

Copybook	Contents
FMCCONST	Constants
FMCRCS	Return codes
FMCPERF	Subprograms of full API
FMCPERFL	Subprograms of the Container API
FMCVARS	Variables

If you use the PERFORM mechanism, you must include FMCCONST, FMCRCS, FMCVARS, and either FMCPERF (if your program is using the full API) or FMCPERFL (if your program is an activity implementation using only the Container API).

If you use the CALL mechanism, you need not include any copybooks, but using FMCCONST and FMCRCS will provide you with all the values that MQSeries Workflow defines. FMCVARS can spare you some effort in declaring variables. FMCPERF and FMCPERFL are not useful in this case.

The following JCLs are provided as samples for the development and execution of MQ Workflow applications. They are located in the SFMCCNTL library delivered with MQSeries Workflow.

Table 5. JCLs provided for COBOL programs

Job	Sample
Native OS/390 COBOL Full API compile job	FMCHJ1BF
Native OS/390 COBOL API run job	FMCHJ1BR
CICS COBOL Full API compile job	FMCHJ2BF
CICS COBOL Container API Compile Job	FMCHJ2BC
IMS COBOL Container API Compile Job	FMCHJ3BC

For more information about CICS/IMS specifics like stubs or precompiler, refer to the documentation of these components.

The compiler given as a prerequisite or newer versions can be used to compile and link your applications accessing the MQ Workflow API.

Mapping C to COBOL data types

Table 6 shows how to map C to COBOL data types:

Table 6. Mapping C to COBOL data types

Type in C	Type in COBOL	BY VALUE / BY REFERENCE
XxxHandle	01 ptr USAGE IS POINTER. (pointing to an object)	BY VALUE
XxxHandle *	01 ptr USAGE IS POINTER. (pointing to a pointer to an object)	BY REFERENCE
char *, char const *	01 ptr USAGE IS POINTER. (pointing to a PIC X(n))	BY VALUE
FmcjCorrelID *	01 ptr USAGE IS POINTER. (pointing to a PIC X(24))	BY VALUE
FmcjBinary *	01 ptr USAGE IS POINTER.	BY VALUE
FmcjCDateTime const *	01 FmcjCDateTime. 05 the-year PIC 9(4) BINARY. 05 the-month PIC 9(4) BINARY. 05 the-day PIC 9(4) BINARY. 05 the-hour PIC 9(4) BINARY. 05 the-minute PIC 9(4) BINARY. 05 the-second PIC 9(4) BINARY.	BY REFERENCE
int, long, signed long, enum	01 int PIC S9(9) BINARY.	BY VALUE
APIRET	01 int PIC S9(9) BINARY.	n/a (not used as parameters)
unsigned long	01 ulong PIC 9(9) BINARY.	BY VALUE
unsigned short	01 ushort PIC 9(4) BINARY.	BY VALUE
double	01 double COMP-2.	BY VALUE
bool (1=true 0=false)	01 bool PIC 9 BINARY.	BY VALUE
long *	01 int PIC S9(9) BINARY.	BY REFERENCE
double *	01 double COMP-2.	BY REFERENCE
unsigned long const *	01 ulong PIC 9(9) BINARY.	BY VALUE[8]

Name changes between COBOL and C

For some of the C functions, synonyms are declared in the form of

```
#define functionA functionB
```

This is to reflect the fact that routines belonging to a superclass are already available to perform the desired function.

8. The respective C routine expects the actual value to be passed rather than a pointer thereto. This is apparently the result of internal optimization when the C routine is compiled.

In these cases, if you want to call functionA directly via CALL in COBOL, you must use functionB instead. All functions belonging to this category are listed in the table below.

Note: If you use the PERFORM statement with the FMCPERF copybook, you must perform a paragraph whose name is an abbreviated version (see Table 8 on page 159) of the name shown in the left-hand column of the following table. This paragraph then contains a call to the proper COBOL subprogram. For example, to call "FmcjActivityInstanceNotificationSetDescription" via PERFORM, you would code:

```
PERFORM FmcjAINSetDescription
```

which contains

```
CALL FmcjItemSetDescription
```

.

Table 7. Function name mapping

C function	Corresponding COBOL subprogram
FmcjActivityInstanceNotificationCategory	FmcjItemCategory
FmcjActivityInstanceNotificationCategoryIsNull	FmcjItemCategoryIsNull
FmcjActivityInstanceNotificationCreationTime	FmcjItemCreationTime
FmcjActivityInstanceNotificationDelete	FmcjItemDelete
FmcjActivityInstanceNotificationDescription	FmcjItemDescription
FmcjActivityInstanceNotificationDescriptionIsNull	FmcjItemDescriptionIsNull
FmcjActivityInstanceNotificationDocumentation	FmcjItemDocumentation
FmcjActivityInstanceNotificationDocumentationIsNull	FmcjItemDocumentationIsNull
FmcjActivityInstanceNotificationEndTime	FmcjItemEndTime
FmcjActivityInstanceNotificationEndTimeIsNull	FmcjItemEndTimeIsNull
FmcjActivityInstanceNotificationEqual	FmcjItemEqual
FmcjActivityInstanceNotificationIcon	FmcjItemIcon
FmcjActivityInstanceNotificationInContainerName	FmcjItemInContainerName
FmcjActivityInstanceNotificationIsComplete	FmcjItemIsComplete
FmcjActivityInstanceNotificationKind	FmcjItemKind
FmcjActivityInstanceNotificationLastModificationTime	FmcjItemLastModificationTime
FmcjActivityInstanceNotificationName	FmcjItemName
FmcjActivityInstanceNotificationObtainProcessMonitor	FmcjItemObtainProcessMonitor
FmcjActivityInstanceNotificationOutContainerName	FmcjItemOutContainerName
FmcjActivityInstanceNotificationOwner	FmcjItemOwner
FmcjActivityInstanceNotificationPersistentOid	FmcjItemPersistentOid
FmcjActivityInstanceNotificationPersistentOidOfProcessInstance	FmcjItemPersistentOidOfProcessInstance
FmcjActivityInstanceNotificationProcessAdmin	FmcjItemProcessAdmin
FmcjActivityInstanceNotificationProcessInstance	FmcjItemProcessInstance
FmcjActivityInstanceNotificationProcessInstanceName	FmcjItemProcessInstanceName
FmcjActivityInstanceNotificationProcessInstanceState	FmcjItemProcessInstanceState
FmcjActivityInstanceNotificationProcessInstanceSystemGroupName	FmcjItemProcessInstanceSystemGroupName

COBOL interface

Table 7. Function name mapping (continued)

C function	Corresponding COBOL subprogram
FmcjActivityInstanceNotificationProcessInstanceSystemName	FmcjItemProcessInstanceSystemName
FmcjActivityInstanceNotificationReceivedAs	FmcjItemReceivedAs
FmcjActivityInstanceNotificationReceivedTime	FmcjItemReceivedTime
FmcjActivityInstanceNotificationRefresh	FmcjItemRefresh
FmcjActivityInstanceNotificationSetDescription	FmcjItemSetDescription
FmcjActivityInstanceNotificationSetName	FmcjItemSetName
FmcjActivityInstanceNotificationStartTime	FmcjItemStartTime
FmcjActivityInstanceNotificationStartTimeIsNull	FmcjItemStartTimeIsNull
FmcjActivityInstanceNotificationTransfer	FmcjItemTransfer
FmcjActivityInstanceNotificationUpdate	FmcjItemUpdate
FmcjExecutionServiceIsLoggedOn	FmcjServiceIsLoggedOn
FmcjExecutionServiceRefresh	FmcjServiceRefresh
FmcjExecutionServiceSetPassword	FmcjServiceSetPassword
FmcjExecutionServiceSetTimeout	FmcjServiceSetTimeout
FmcjExecutionServiceSystemGroupName	FmcjServiceSystemGroupName
FmcjExecutionServiceSystemName	FmcjServiceSystemName
FmcjExecutionServiceTimeout	FmcjServiceTimeout
FmcjExecutionServiceUserID	FmcjServiceUserID
FmcjExecutionServiceUserSettings	FmcjServiceUserSettings
FmcjProcessInstanceListDelete	FmcjPersistentListDelete
FmcjProcessInstanceListDescription	FmcjPersistentListDescription
FmcjProcessInstanceListDescriptionIsNull	FmcjPersistentListDescriptionIsNull
FmcjProcessInstanceListFilter	FmcjPersistentListFilter
FmcjProcessInstanceListFilterIsNull	FmcjPersistentListFilterIsNull
FmcjProcessInstanceListName	FmcjPersistentListName
FmcjProcessInstanceListOwnerOfList	FmcjPersistentListOwnerOfList
FmcjProcessInstanceListOwnerOfListIsNull	FmcjPersistentListOwnerOfListIsNull
FmcjProcessInstanceListPersistentOid	FmcjPersistentListPersistentOid
FmcjProcessInstanceListRefresh	FmcjPersistentListRefresh
FmcjProcessInstanceListSetDescription	FmcjPersistentListSetDescription
FmcjProcessInstanceListSetFilter	FmcjPersistentListSetFilter
FmcjProcessInstanceListSetSortCriteria	FmcjPersistentListSetSortCriteria
FmcjProcessInstanceListSetThreshold	FmcjPersistentListSetThreshold
FmcjProcessInstanceListSortCriteria	FmcjPersistentListSortCriteria
FmcjProcessInstanceListSortCriteriaIsNull	FmcjPersistentListSortCriteriaIsNull
FmcjProcessInstanceListThreshold	FmcjPersistentListThreshold
FmcjProcessInstanceListThresholdIsNull	FmcjPersistentListThresholdIsNull
FmcjProcessInstanceListType	FmcjPersistentListType
FmcjProcessInstanceNotificationCategory	FmcjItemCategory
FmcjProcessInstanceNotificationCategoryIsNull	FmcjItemCategoryIsNull

Table 7. Function name mapping (continued)

C function	Corresponding COBOL subprogram
FmcjProcessInstanceNotificationCreationTime	FmcjItemCreationTime
FmcjProcessInstanceNotificationDelete	FmcjItemDelete
FmcjProcessInstanceNotificationDescription	FmcjItemDescription
FmcjProcessInstanceNotificationDescriptionIsNull	FmcjItemDescriptionIsNull
FmcjProcessInstanceNotificationDocumentation	FmcjItemDocumentation
FmcjProcessInstanceNotificationDocumentationIsNull	FmcjItemDocumentationIsNull
FmcjProcessInstanceNotificationEndTime	FmcjItemEndTime
FmcjProcessInstanceNotificationEndTimeIsNull	FmcjItemEndTimeIsNull
FmcjProcessInstanceNotificationEqual	FmcjItemEqual
FmcjProcessInstanceNotificationIcon	FmcjItemIcon
FmcjProcessInstanceNotificationInContainerName	FmcjItemInContainerName
FmcjProcessInstanceNotificationIsComplete	FmcjItemIsComplete
FmcjProcessInstanceNotificationKind	FmcjItemKind
FmcjProcessInstanceNotificationLastModificationTime	FmcjItemLastModificationTime
FmcjProcessInstanceNotificationName	FmcjItemName
FmcjProcessInstanceNotificationObtainProcessMonitor	FmcjItemObtainProcessMonitor
FmcjProcessInstanceNotificationOutContainerName	FmcjItemOutContainerName
FmcjProcessInstanceNotificationOwner	FmcjItemOwner
FmcjProcessInstanceNotificationPersistentOid	FmcjItemPersistentOid
FmcjProcessInstanceNotificationPersistentOidOfProcessInstance	FmcjItemPersistentOidOfProcessInstance
FmcjProcessInstanceNotificationProcessAdmin	FmcjItemProcessAdmin
FmcjProcessInstanceNotificationProcessInstance	FmcjItemProcessInstance
FmcjProcessInstanceNotificationProcessInstanceName	FmcjItemProcessInstanceName
FmcjProcessInstanceNotificationProcessInstanceState	FmcjItemProcessInstanceState
FmcjProcessInstanceNotificationProcessInstanceSystemGroupName	FmcjItemProcessInstanceSystemGroupName
FmcjProcessInstanceNotificationProcessInstanceSystemName	FmcjItemProcessInstanceSystemName
FmcjProcessInstanceNotificationReceivedAs	FmcjItemReceivedAs
FmcjProcessInstanceNotificationReceivedTime	FmcjItemReceivedTime
FmcjProcessInstanceNotificationRefresh	FmcjItemRefresh
FmcjProcessInstanceNotificationSetDescription	FmcjItemSetDescription
FmcjProcessInstanceNotificationSetName	FmcjItemSetName
FmcjProcessInstanceNotificationStartTime	FmcjItemStartTime
FmcjProcessInstanceNotificationStartTimeIsNull	FmcjItemStartTimeIsNull
FmcjProcessInstanceNotificationTransfer	FmcjItemTransfer
FmcjProcessInstanceNotificationUpdate	FmcjItemUpdate
FmcjProcessTemplateListDelete	FmcjPersistentListDelete
FmcjProcessTemplateListDescription	FmcjPersistentListDescription
FmcjProcessTemplateListDescriptionIsNull	FmcjPersistentListDescriptionIsNull
FmcjProcessTemplateListFilter	FmcjPersistentListFilter
FmcjProcessTemplateListFilterIsNull	FmcjPersistentListFilterIsNull

COBOL interface

Table 7. Function name mapping (continued)

C function	Corresponding COBOL subprogram
FmcjProcessTemplateListName	FmcjPersistentListName
FmcjProcessTemplateListOwnerOfList	FmcjPersistentListOwnerOfList
FmcjProcessTemplateListOwnerOfListIsNull	FmcjPersistentListOwnerOfListIsNull
FmcjProcessTemplateListPersistentOid	FmcjPersistentListPersistentOid
FmcjProcessTemplateListRefresh	FmcjPersistentListRefresh
FmcjProcessTemplateListSetDescription	FmcjPersistentListSetDescription
FmcjProcessTemplateListSetFilter	FmcjPersistentListSetFilter
FmcjProcessTemplateListSetSortCriteria	FmcjPersistentListSetSortCriteria
FmcjProcessTemplateListSetThreshold	FmcjPersistentListSetThreshold
FmcjProcessTemplateListSortCriteria	FmcjPersistentListSortCriteria
FmcjProcessTemplateListSortCriteriaIsNull	FmcjPersistentListSortCriteriaIsNull
FmcjProcessTemplateListThreshold	FmcjPersistentListThreshold
FmcjProcessTemplateListThresholdIsNull	FmcjPersistentListThresholdIsNull
FmcjProcessTemplateListType	FmcjPersistentListType
FmcjProgramDataExecutionMode	FmcjProgramTemplateExecutionMode
FmcjProgramDataExecutionUser	FmcjProgramTemplateExecutionUser
FmcjProgramDataProgramTrusted	FmcjProgramTemplateProgramTrusted
FmcjReadOnlyContainerAllLeafCount	FmcjContainerAllLeafCount
FmcjReadOnlyContainerAllLeaves	FmcjContainerAllLeaves
FmcjReadOnlyContainerArrayBinaryLength	FmcjContainerArrayBinaryLength
FmcjReadOnlyContainerArrayBinaryValue	FmcjContainerArrayBinaryValue
FmcjReadOnlyContainerArrayFloatValue	FmcjContainerArrayFloatValue
FmcjReadOnlyContainerArrayLongValue	FmcjContainerArrayLongValue
FmcjReadOnlyContainerArrayStringLength	FmcjContainerArrayStringLength
FmcjReadOnlyContainerArrayStringValue	FmcjContainerArrayStringValue
FmcjReadOnlyContainerAsStream	FmcjContainerAsStream
FmcjReadOnlyContainerBinaryLength	FmcjContainerBinaryLength
FmcjReadOnlyContainerBinaryValue	FmcjContainerBinaryValue
FmcjReadOnlyContainerFloatValue	FmcjContainerFloatValue
FmcjReadOnlyContainerGetElement	FmcjContainerGetElement
FmcjReadOnlyContainerLeafCount	FmcjContainerLeafCount
FmcjReadOnlyContainerLeaves	FmcjContainerLeaves
FmcjReadOnlyContainerLongValue	FmcjContainerLongValue
FmcjReadOnlyContainerMemberCount	FmcjContainerMemberCount
FmcjReadOnlyContainerSetStringCcsid	FmcjContainerSetStringCcsid
FmcjReadOnlyContainerStringLength	FmcjContainerStringLength
FmcjReadOnlyContainerStringValue	FmcjContainerStringValue
FmcjReadOnlyContainerStructMembers	FmcjContainerStructMembers
FmcjReadOnlyContainerType	FmcjContainerType
FmcjReadWriteContainerAllLeafCount	FmcjContainerAllLeafCount

Table 7. Function name mapping (continued)

C function	Corresponding COBOL subprogram
FmcjReadWriteContainerAllLeaves	FmcjContainerAllLeaves
FmcjReadWriteContainerArrayBinaryLength	FmcjContainerArrayBinaryLength
FmcjReadWriteContainerArrayBinaryValue	FmcjContainerArrayBinaryValue
FmcjReadWriteContainerArrayFloatValue	FmcjContainerArrayFloatValue
FmcjReadWriteContainerArrayLongValue	FmcjContainerArrayLongValue
FmcjReadWriteContainerArrayStringLength	FmcjContainerArrayStringLength
FmcjReadWriteContainerArrayStringValue	FmcjContainerArrayStringValue
FmcjReadWriteContainerAsStream	FmcjContainerAsStream
FmcjReadWriteContainerBinaryLength	FmcjContainerBinaryLength
FmcjReadWriteContainerBinaryValue	FmcjContainerBinaryValue
FmcjReadWriteContainerFloatValue	FmcjContainerFloatValue
FmcjReadWriteContainerGetElement	FmcjContainerGetElement
FmcjReadWriteContainerLeafCount	FmcjContainerLeafCount
FmcjReadWriteContainerLeaves	FmcjContainerLeaves
FmcjReadWriteContainerLongValue	FmcjContainerLongValue
FmcjReadWriteContainerMemberCount	FmcjContainerMemberCount
FmcjReadWriteContainerSetStringCcsid	FmcjContainerSetStringCcsid
FmcjReadWriteContainerStringLength	FmcjContainerStringLength
FmcjReadWriteContainerStringValue	FmcjContainerStringValue
FmcjReadWriteContainerStructMembers	FmcjContainerStructMembers
FmcjReadWriteContainerType	FmcjContainerType
FmcjWorkitemCategory	FmcjItemCategory
FmcjWorkitemCategoryIsNull	FmcjItemCategoryIsNull
FmcjWorkitemCreationTime	FmcjItemCreationTime
FmcjWorkitemDelete	FmcjItemDelete
FmcjWorkitemDescription	FmcjItemDescription
FmcjWorkitemDescriptionIsNull	FmcjItemDescriptionIsNull
FmcjWorkitemDocumentation	FmcjItemDocumentation
FmcjWorkitemDocumentationIsNull	FmcjItemDocumentationIsNull
FmcjWorkitemEndTime	FmcjItemEndTime
FmcjWorkitemEndTimeIsNull	FmcjItemEndTimeIsNull
FmcjWorkitemEqual	FmcjItemEqual
FmcjWorkitemIcon	FmcjItemIcon
FmcjWorkitemInContainerName	FmcjItemInContainerName
FmcjWorkitemIsComplete	FmcjItemIsComplete
FmcjWorkitemKind	FmcjItemKind
FmcjWorkitemLastModificationTime	FmcjItemLastModificationTime
FmcjWorkitemName	FmcjItemName
FmcjWorkitemObtainProcessMonitor	FmcjItemObtainProcessMonitor
FmcjWorkitemOutContainerName	FmcjItemOutContainerName

COBOL interface

Table 7. Function name mapping (continued)

C function	Corresponding COBOL subprogram
FmcjWorkitemOwner	FmcjItemOwner
FmcjWorkitemPersistentOid	FmcjItemPersistentOid
FmcjWorkitemPersistentOidOfProcessInstance	FmcjItemPersistentOidOfProcessInstance
FmcjWorkitemProcessAdmin	FmcjItemProcessAdmin
FmcjWorkitemProcessInstance	FmcjItemProcessInstance
FmcjWorkitemProcessInstanceName	FmcjItemProcessInstanceName
FmcjWorkitemProcessInstanceState	FmcjItemProcessInstanceState
FmcjWorkitemProcessInstanceSystemGroupName	FmcjItemProcessInstanceSystemGroupName
FmcjWorkitemProcessInstanceSystemName	FmcjItemProcessInstanceSystemName
FmcjWorkitemReceivedAs	FmcjItemReceivedAs
FmcjWorkitemReceivedTime	FmcjItemReceivedTime
FmcjWorkitemRefresh	FmcjItemRefresh
FmcjWorkitemSetDescription	FmcjItemSetDescription
FmcjWorkitemSetName	FmcjItemSetName
FmcjWorkitemStartTime	FmcjItemStartTime
FmcjWorkitemStartTimeIsNull	FmcjItemStartTimeIsNull
FmcjWorkitemTransfer	FmcjItemTransfer
FmcjWorkitemUpdate	FmcjItemUpdate
FmcjWorklistDelete	FmcjPersistentListDelete
FmcjWorklistDescription	FmcjPersistentListDescription
FmcjWorklistDescriptionIsNull	FmcjPersistentListDescriptionIsNull
FmcjWorklistFilter	FmcjPersistentListFilter
FmcjWorklistFilterIsNull	FmcjPersistentListFilterIsNull
FmcjWorklistName	FmcjPersistentListName
FmcjWorklistOwnerOfList	FmcjPersistentListOwnerOfList
FmcjWorklistOwnerOfListIsNull	FmcjPersistentListOwnerOfListIsNull
FmcjWorklistPersistentOid	FmcjPersistentListPersistentOid
FmcjWorklistRefresh	FmcjPersistentListRefresh
FmcjWorklistSetDescription	FmcjPersistentListSetDescription
FmcjWorklistSetFilter	FmcjPersistentListSetFilter
FmcjWorklistSetSortCriteria	FmcjPersistentListSetSortCriteria
FmcjWorklistSetThreshold	FmcjPersistentListSetThreshold
FmcjWorklistSortCriteria	FmcjPersistentListSortCriteria
FmcjWorklistSortCriteriaIsNull	FmcjPersistentListSortCriteriaIsNull
FmcjWorklistThreshold	FmcjPersistentListThreshold
FmcjWorklistThresholdIsNull	FmcjPersistentListThresholdIsNull
FmcjWorklistType	FmcjPersistentListType

To cope with the COBOL restriction of 30 characters per COBOL word, some class name prefixes and function and constant names have been abbreviated. The

abbreviations for the class name prefixes are listed in the table below.

Table 8. Class prefix abbreviations

Class Name	Abbreviation
FmcjActivityInstance	FmcjAI
FmcjActivityInstanceNotification	FmcjAIN
FmcjActivityInstanceNotificationVector	FmcjAINV
FmcjActivityInstanceVector	FmcjAIV
FmcjContainer	FmcjC
FmcjContainerElement	FmcjCE
FmcjContainerElementVector	FmcjCEV
FmcjControlConnectorInstance	FmcjCCI
FmcjControlConnectorInstanceVector	FmcjCCIV
FmcjDllOptions	FmcjDO
FmcjExecutionData	FmcjED
FmcjExecutionService	FmcjES
FmcjExeOptions	FmcjExeO
FmcjExternalOptions	FmcjExtO
FmcjImplementationData	FmcjID
FmcjImplementationDataVector	FmcjIDV
FmcjInstanceMonitor	FmcjIM
FmcjPersistentList	FmcjPL
FmcjPerson	FmcjP
FmcjPoint	FmcjPnt
FmcjPointVector	FmcjPntV
FmcjProcessInstance	FmcjPI
FmcjProcessInstanceList	FmcjPIL
FmcjProcessInstanceListVector	FmcjPILV
FmcjProcessInstanceNotification	FmcjPIN
FmcjProcessInstanceNotificationVector	FmcjPINV
FmcjProcessInstanceVector	FmcjPIV
FmcjProcessTemplate	FmcjPT
FmcjProcessTemplateList	FmcjPTL
FmcjProcessTemplateListVector	FmcjPTLV
FmcjProcessTemplateVector	FmcjPTV
FmcjProgramData	FmcjPD
FmcjProgramTemplate	FmcjPgT
FmcjReadOnlyContainer	FmcjROC
FmcjReadWriteContainer	FmcjRWC
FmcjService	FmcjSrv
FmcjStringVector	FmcjStrV
FmcjSymbolLayout	FmcjSL
FmcjWorkitem	FmcjWI

Table 8. Class prefix abbreviations (continued)

Class Name	Abbreviation
FmcjWorkitemVector	FmcjWIV
FmcjWorklist	FmcjWL

The abbreviations for function, constant, and error code names are listed in the table below (note that these abbreviations apply after the class name prefix abbreviations). Instead of constructing the names with the help of this table you can also search the FMCPERF copybook for the C function name to get the corresponding COBOL function and variable names.

Table 9. Abbreviations for COBOL naming

Original String	Abbreviation
Activity	Act
ACTIVITY	ACT
Administration	Admin
ADMINSTRATION	ADMIN
Already	Alr
ALREADY	ALR
Authorization	Auth
AUTHORIZATION	AUTH
Authorized	Auth
AUTHORIZED	AUTH
Backward	Backw
BACKWARD	BACKW
Categories	Categs
CATEGORIES	CATEGS
Category	Categ
CATEGORY	CATEG
CHECKOUT	CHKOUT
Container	Ctnr
CONTAINER	CTNR
ControlConnector	ContrConn
CONTROLCONNECTOR	CONTRCONN
DATA-MEMBER	D-M
Definition	Def
DEFINITION	DEF
Directory	Dir
DIRECTORY	DIR
DOCUMENT	DOC
Executable	Exec
EXECUTABLE	EXEC
EXTERNAL	EXT

Table 9. Abbreviations for COBOL naming (continued)

Original String	Abbreviation
ForeGround	ForeGr
FOREGROUND	FOREGR
Forward	Forw
FORWARD	FORW
FOUND-FOR-AUTO-START	FND-FR-AUT-ST
FROM	FRM
IMPLEMENTATION	IMPL
Instance	Inst
INSTANCE	INST
INVALID	INVAL
Invocation	Inv
INVOCATION	INV
Location	Loc
LOCATION	LOC
Mapping	Map
MAPPING	MAP
MESSAGE	MSG
Monitor	Mon
MONITOR	MON
Notification	Notif
NOTIFICATION	NOTIF
Notified	Notif
NOTIFIED	NOTIF
Object	Obj
OBJECT	OBJ
Organization	Org
ORGANIZATION	ORG
Parameter	Parm
PARAMETER	PARM
PersistentOidOf	PersOidOf
PERSISTENTOIDOF	PERSOIDOF
Persons	Pers
PERSON	PERS
Process	Proc
PROCESS	PROC
Program	Prog
PROGRAM	PROG
Second	Sec
SECOND	SEC
Service	Serv

Table 9. Abbreviations for COBOL naming (continued)

Original String	Abbreviation
SERVICE	SERV
Started	Strtd
STARTED	STRTD
STRUCTURE	STR
SUB-PROC	SB-PRC
SUPPORTED	SUPP
Suspension	Susp
SUSPENSION	SUSP
System	Syst
SYSTEM	SYST
Template	Templ
TEMPLATE	TEMPL
Terminated	Term
TERMINATED	TERM
Transition	Trans
TRANSITION	TRANS

The C API uses some variable names that are reserved words in COBOL. These variable names were changed in the COBOL API by appending ″Value″ to the variable name. All variables belonging to this category are listed below.

```
year      month     day       hour    minute  second   data
file      function  index     input   line    output   owner
password  program   service   time    type    timeout
```

Finally, the variable name ″value″ is used in the C API with different types. In COBOL this variable is renamed according to its type:

intValue, doubleValue, pointerValue

Example of the use of strings

The following code calls a C function with the signature char* cfunc(char* x).

Note: The SETADDR routine sets a pointer variable to the address of a local string. The GETADDR routine copies a string referred to by a pointer variable to a local string.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "STRTEST".
DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 PTR1 USAGE IS POINTER VALUE NULL.
  01 PTR2 USAGE IS POINTER VALUE NULL.
  01 STRING-STRUCT1.
     05 X PIC X(20).
  01 STRING-STRUCT2.
     05 Y PIC X(20).
  01 STRLEN PIC 99 VALUE 0.
PROCEDURE DIVISION.
    MOVE z"Initial String" TO X.
    CALL "SETADDR" USING STRING-STRUCT1 PTR1.
```

```
    CALL "cfunc" USING BY VALUE PTR1
                 RETURNING PTR2.
    CALL "GETADDR" USING STRING-STRUCT2 PTR2.
    INSPECT Y TALLYING STRLEN FOR CHARACTERS
        BEFORE INITIAL X"00".
    DISPLAY "Y is " Y(1:STRLEN).
    STOP RUN.
END PROGRAM "STRTEST".

IDENTIFICATION DIVISION.
PROGRAM-ID. "SETADDR".
DATA DIVISION.
  LINKAGE SECTION.
  01 PTR3 USAGE IS POINTER.
  01 STRING-STRUCT3.
     05 Z PIC X(20).
PROCEDURE DIVISION USING BY REFERENCE STRING-STRUCT3 PTR3.
    SET PTR3 TO ADDRESS OF Z.
    GOBACK.
END PROGRAM "SETADDR".

IDENTIFICATION DIVISION.
PROGRAM-ID. "GETADDR".
DATA DIVISION.
  LINKAGE SECTION.
  01 PTR4 USAGE IS POINTER.
  01 STRING-STRUCT4.
     05 Z PIC X(20).
  01 DUMMY-STRUCT.
     05 W PIC X(20).
PROCEDURE DIVISION USING BY REFERENCE STRING-STRUCT4 PTR4.
    SET ADDRESS OF DUMMY-STRUCT TO PTR4.
    MOVE W TO Z.
    GOBACK.
END PROGRAM "GETADDR".
```

The XML message interface

The following chapters provide a description of the MQ Workflow XML message-based interface. It explains the format of a message and how XML can be used:

- Sending requests to MQ Workflow

  An action can be started on an MQ Workflow server by sending a message to the MQ Workflow XML input queue.

  This allows any application that supports the MQ Workflow XML message format to request an action from MQ Workflow.

- Invoking an activity implementation

  An activity implementation is invoked by MQ Workflow by sending an appropriate message to a user-defined MQSeries queue.

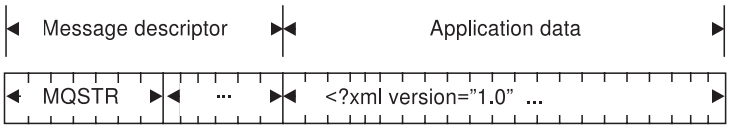
  This allows you to start any application listening on an MQSeries queue. The queue can be input to any MQSeries application that can handle XML messages. This can be your own in-house application, or a standard program, such as MQSeries Integrator V2.

The MQ Workflow XML message

MQ Workflow uses MQSeries to exchange messages. An MQSeries message consists of two parts:

1. The MQSeries message descriptor (MQMD), which contains structured data describing the message

2. The application data, which contains the MQ Workflow XML message itself



For more information about the MQMD, the application data, and how to send and receive messages in an MQSeries network, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries messages".

Relevant MQSeries Message Descriptor (MQMD) fields

The following fields of the MQSeries message descriptor are used by MQ Workflow:

- UserIdentifier

  The user who sent the message. For request messages sent to MQ Workflow this information is used as the MQ Workflow user on whose behalf the request is performed. Also, authorization checks are performed using this user ID. For invoke messages sent by MQ Workflow this field contains the user on whose behalf the activity implementation is to be started.

- Format

  A fixed string indicating that this message contains an MQ Workflow XML message. Its value is defined by the MQSeries constant MQFMT_STRING (MQSTR). For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

- ReplyToQ/ReplyToQMgr

  Specifies the queue and queue manager the response should be sent to.

- Persistence

  Specifies whether the message is persistent or transient. For requests sent to MQ Workflow, the MQ Workflow response has the same persistence as the request. XML requests sent by MQ Workflow are persistent and responses sent by invoked activity implementations should also be persistent.

- Expiry

  Can be set to a period of time expressed in tenths of a second for transient messages; should be set to unlimited (MQEI_UNLIMITED) for persistent messages. For requests sent to MQ Workflow, the expiry of the MQ Workflow response is set to the expiration value in the request minus the time spent on execution. XML requests sent by MQ Workflow have an unlimited expiration time.

- CorrelID

  Data that can be used to relate a response message to a request message. For requests sent to MQ Workflow, the MQ Workflow response contains the same correlation ID as the request. XML requests sent by MQ Workflow contain a correlation ID and responses sent by an application should return that correlation ID.

- BackoutCount

  A count of the number of times the message has been returned to the input queue because the transaction failed and was rolled back. In other words, the backout count denotes the number of times processing of the message was not successful.

For detailed information, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries Messages".

The application data

MQ Workflow uses the XML 1.0 standard for message description. Refer to http://www.w3.org/TR/REC-xml for the XML Reference.

In general, an MQ Workflow XML message contains the following information:

- An MQ Workflow XML message header, that is, information that is common for all messages, for example, the user context
- An MQ Workflow message name which specifies the request or response contained in the message, for example, a ″ProcessTemplateExecute″ request
- The parameters that are needed to execute the request or to analyze the response, for example, an input container

When processing an MQ Workflow XML message, MQ Workflow checks if the message has the correct format - see "Error Handling" on page 175.

The MQ Workflow XML message header: The MQ Workflow XML message header contains the following information:

- If a response should be sent.

  With a message-based interface, both synchronous and asynchronous request/response scenarios can occur. That is why the creation of a response to a given request is made optional. However, even if responses are generally not desired, an exceptional response to report an error can still be required. Such options are provided to request:

  – No (″No″)
  – Only error (″IfError″)
  – All (″Yes″)

  responses which are sent to the reply queue specified in the MQMD of the request message. If *ResponseRequired* is not set, then the default value assumed by MQ Workflow for requests is ″Yes″, while the default value for responses is ″No″.

- The user context

  In this field, you can specify up to 254 bytes of context data that can be used for correlating a request and a response. The user context data specified in a request to MQ Workflow is returned in the associated response.

  Therefore, the necessity to keep state information in the component sending the message is avoided. For example, when a message is routed through an intermediary like MQSeries Integrator V2, it can be desirable to route the response back through the intermediary, which then in turn will send the message back to the original requester. The user context data can contain information in order to keep the original requester, or even an entire route, without requiring the intermediary to maintain state information.

Container data: For a general introduction on containers refer to "Handling containers" on page 30. The following example shows a container of type *CreditData* containing three container elements of a basic type, *Amount* of basic type LONG and *Currency* and *Risk* of basic type STRING, and a nested container element *Customer* of type CustomerData. *CustomerData* contains a container element *Name* of basic type STRING and an array *Account* with two elements of basic type LONG.

```
<CreditData>
   <Customer>
      <Name>User1</Name>
      <Account>4711</Account>
      <Account>1100</Account>
   </Customer>
   <Amount>100000</Amount>
   <Currency>CurrencyX</Currency>
   <Risk>high</Risk>
</CreditData>
```

The following rules apply to containers in the message-based interface:

- A container, actually the user-defined part of the container without the pre-defined data members, is identified by its type, that is, the name of the associated data structure. In other words, the name of the XML element describing the container is the name of the data structure specifying the type of the container; CreditData in the example above.
- Container elements are specified by their name; their type is not part of the XML message.
- Container elements can be of a basic type or denote another data structure as their type. Basic container elements are mapped to PCDATA elements, while container elements of a non-basic type are decomposed into XML subelements according to their structure.
- The structure of XML elements representing a container or container element of a non-basic type reflects the structure of the associated data structure. Therefore, data member names are not prefixed; there is no dotted name representation.

  Note that the context-free nature of XML does not allow for data structures having the same names as data members. Also, two data members with the same name must be of the same type even if they are contained in different data structures. When the MQ Workflow Buildtime Verification encounters one of these situations, it issues a warning.
- XML tags must not contain blanks. It follows that data structure, and data member names must not contain blanks. If such a name containing blanks is to be referred to in an XML message, then the name without blanks is to be used. For example, the "Default Data Structure" is to be referred to as <DefaultDataStructure>.

  Note: Ambiguities arise when a model contains data structures with names differing only by blanks. The data modeler has to prevent that situation.
- Data structure and data member names must not contain reserved XML characters as there are '<', '>', and '/'. If such a name is used in an XML message, the generated XML is not well formed and cannot be parsed.
- The specification of container elements (data structure members) in the XML message is optional. If a data member is not specified in the XML message, MQ Workflow sets its value to *Null* (not set).

Data type encoding rules:

- The values of container elements of basic types STRING, LONG, and FLOAT can be coded directly.
- For boolean parameters, values "false" and "true" (case insensitive) should be used. Values 0 and 1 are also supported.
- Binary data has to be encoded into its printable version. MQ Workflow uses base64 encoding to represent binary data in XML messages. Note that this encoding preserves length information. Refer to http://www.cis.ohio-state.edu/htbin/rfc/rfc2045.html for base64 encoding rules.

- Leading blanks, trailing blanks, new line characters, and tab characters are removed from values of LONG, FLOAT, and boolean types. Such characters are allowed to support you in formatting your XML message. They are, however, not removed from values of types STRING or BINARY so that the value specified remains unchanged.

Representation of arrays:

- Arrays are specified as a sequence of elements.
- Since arrays are to be specified as a sequence of elements, arrays with one element only are not supported because they cannot be distinguished from non-array elements.
- The <null/> element can be used to specify that an array element is *Null* (not set). For example, anticipate a container of type *Error*. Error contains an ID and three reason codes (an array of reason codes). Assumed that the second reason code is not set, this can be specified as:

```
<Error>
    <ID>111</ID>
    <ReasonCode>12</ReasonCode>
    <ReasonCode><null/></ReasonCode>
    <ReasonCode>5050</ReasonCode>
</Error>
```

Pre-defined data members:

- Pre-defined data members are specified in the same way as user-defined data members. They are to be defined right after the XML container element, for example, after <ProcInstInputData>. Again, they have to be decomposed according to their data structure. For example:

```
<_ACTIVITY_INFO>
   <Priority>1</Priority>
</_ACTIVITY_INFO>
```

Execute process instance example: The following example shows an XML message that requests the execution of a process instance:

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateExecute>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <ProcInstName>Credit_Request#658321</ProcInstName>
      <KeepName>true</KeepName>
      <ProcInstInputData>
          <_ACTIVITY_INFO>
            <Priority>1</Priority>
          </_ACTIVITY_INFO>
          <CreditData>
            <Customer>
              <Name>User1</Name>
              <Account>4711</Account>
              <Account>1100</Account>
            </Customer>
            <Amount>100000</Amount>
            <Currency>CurrencyX</Currency>
            <Risk>high<Risk>
          </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage
```

Code page support

XML allows for the specification of messages in Unicode, as well as in ISO-defined character sets. XML messages sent to MQ Workflow are converted from their format as specified in the encoding keyword to the MQ Workflow code page as necessary. XML messages sent by MQ Workflow (responses and activity implementation invocation requests) are always encoded in UTF-8.

Refer to the readme.1st, chapter "XML code page support", for a list of code pages supported by the MQ Workflow XML parser.

Sending requests to MQ Workflow

The MQ Workflow message-based interface can be used to request services from MQ Workflow. This is depicted in the following figure:

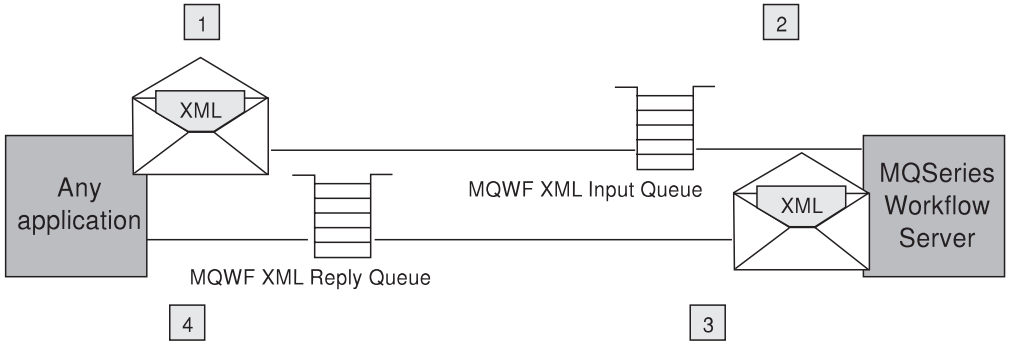


Figure 21. Sending requests to MQ Workflow

1. An application creates an MQ Workflow XML message and puts it into the MQ Workflow XML input queue.
2. MQ Workflow reads the XML message out of the XML input queue and processes the request.
3. MQ Workflow creates a response MQ Workflow XML message and puts it into the reply queue. Note that the reply queue information is part of the MQSeries Message Descriptor (MQMD) of the incoming XML message.

   For more information about the MQMD, the application data, and how to send and receive messages in an MQSeries network, refer to the *IBM MQSeries Application Programming Guide*, chapter "MQSeries messages".
4. The application reads the incoming message and processes the response.

Supported functions

The following requests are supported by the XML message interface:

- "CreateAndStartInstance()" on page 491.
- "ExecuteProcessInstance()" on page 502.

XML input queue

The XML input queue <prefix>.<SystemGroupName>.<SystemName>.EXE.XML respectively <prefix>.<SystemGroupName>.EXE.XML is an MQSeries queue to which MQ Workflow is listening.

Only XML messages are accepted as input to this queue. The XML message has to conform to the MQ Workflow XML message format. If it does not conform, then a GeneralError XML message is put into the reply queue.

For more information refer to chapters “Relevant MQSeries Message Descriptor (MQMD) fields” on page 164 and “The application data” on page 165. For more information on error handling refer to “Error Handling” on page 175.

Authentication and authorization

For authentication MQ Workflow’s message-based interface relies on MQSeries. MQ Workflow does not perform any additional authentication. For setting up MQSeries security, refer to *IBM MQSeries System Administration*, chapter “Protecting MQSeries Objects”.

The value of the UserIdentifier field in the MQMD of the incoming message is used as the MQ Workflow user on whose behalf the request is to be performed. Authorization checks for that user are performed as usual.

Note: MQSeries UserIdentifier constraints differ from the ones defined for the MQ Workflow system. Since authorization is checked by MQ Workflow, the UserIdentifier in the MQMD of an XML message must be a valid MQ Workflow user. This has to be ensured by the application programmer and MQ Workflow administrator.

Create and start a process instance example

The following example shows an XML message that requests the creation and start of a process instance. Assumed that the process input data is described by data structure *Application*:

```
STRUCTURE 'Application'
  'InsuredID'      : LONG;
  'Type'           : LONG;
  'SpecialRisk'    : Long;
  'Accident'       : Long;
  'Ammount'        : FLOAT;
  'StartDate'      : STRING;
  'Term'           : Long;
  'Payment'        : Long;
  'Doctor'         : STRING;
  'Weight'         : Long;
  'Height'         : STRING;
  'Smoker'         : Long;
  'Illness'        : STRING;
  'Hospitalization': STRING;
  'Risks'          : STRING;
END 'Application'
```

then the XML message can look like following:

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateCreateAndStartInstance>
      <ProcTemplName>Medical_Opinion</ProcTemplName>
      <ProcInstName>Medical_Opinion#448</ProcInstName>
      <KeepName>false</KeepName>
      <ProcInstInputData>
          <Application>
            <InsuredID>A</InsuredID>
            <Type>4711</Type>
            <StartDate>12.01.2000</StartDate>
            <Doctor>DoctorX</Doctor>
            <Weight>200</Weight>
            <Smoker>1<Smoker>
```

```
        </Application>
    </ProcInstInputData>
  </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

Invoking an activity implementation

Activity implementations are usually started by MQ Workflow by sending an internal invocation request message to a program execution agent or program execution server. They, in turn, invoke the program that was modeled to implement the activity. Using the message-based interface, it is also possible that MQ Workflow sends that invocation request message in XML format to a user-defined MQSeries queue.

From the point of view of MQ Workflow, the MQSeries application listening on that queue has to invoke the program that is modeled as the implementation of the activity. For doing so, all the necessary information is passed to the MQSeries application by means of an XML message. The MQSeries application must return with an appropriate XML response, if requested by MQ Workflow.

Therefore, such an application is called a user-defined program execution server (UPES). A user-defined program execution server can be any application you write or a program such as MQSeries Integrator, provided it can deal with the MQ Workflow XML message format.

A UPES and a program activity to be performed by that UPES are modeled in the MQ Workflow Buildtime. The program activity is modeled using the activity property sheet.

Two invocation modes for the activity implementation can also be modeled:

- Synchronous invocation (the standard case), where MQ Workflow waits for a completion message containing result data from the UPES before the activity instance is considered to be complete.
- Asynchronous invocation, where no completion message is required and the activity instance is considered to be complete right after the invocation message has been sent. No result data is expected by MQ Workflow and process navigation continues.

The following figure depicts the synchronous invocation of an activity implementation:

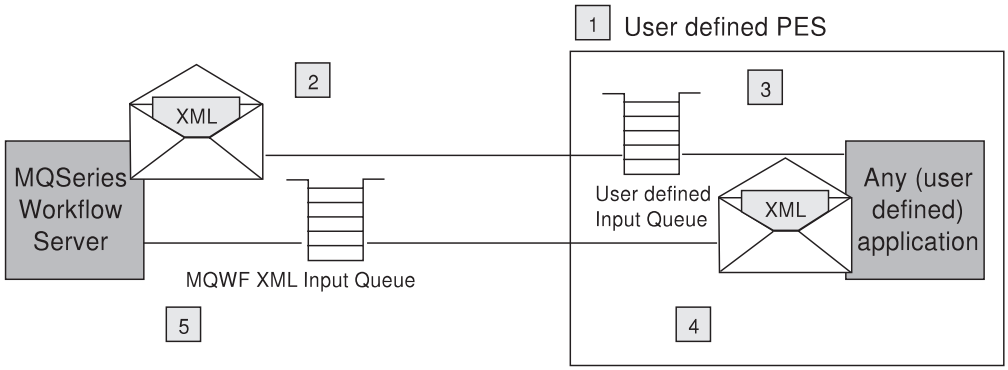


Figure 22. Starting an activity implementation

1. A UPES must have been defined using MQ Workflow Buildtime.

2. When an activity implementation is to be started, MQ Workflow sends a program invocation message to the UPES.
3. An application listening to the UPES queue reads the XML message and performs the appropriate action. Possible actions are:
   - Transform the message into another format and route it to another recipient, for example, send an EDI message to another company.
   - Perform a transaction that involves the *get* of the request from the queue, the update of one or more DBMS or other resource managers, and the *put* of the response, in a single unit of work.
   - Invoke the specified activity implementation, for example, call a program on a platform not yet supported by MQ Workflow.
4. When the activity implementation has finished, the application creates a response MQ Workflow XML message, if required, and puts it into the reply queue. Note that the reply queue information is part of the MQMD of the incoming XML invocation message.
5. MQ Workflow reads the response message, processes it, and changes the state of the activity accordingly.

User-defined program execution server (UPES)

A UPES is defined and configured for an MQ Workflow system by modeling it in MQ Workflow Buildtime. Essential attributes are the name, the version, and the queue it represents.

The UPES version denotes the MQ Workflow API version that is supported by the UPES. MQ Workflow only sends messages to the UPES that are supported. When modeled in the MQ Workflow Buildtime, it becomes a tag in the FDL when the UPES is exported. If no UPES version is specified when imported into the MQ Workflow Runtime, the FDL version becomes also the UPES version.

Note: You need to upgrade the UPES version in order to receive new messages that become available with new MQ Workflow versions or releases.

For more information on UPES attributes refer to the online help of MQ Workflow Buildtime.

The application that is listening to the UPES queue is not managed by MQ Workflow. A system administrator is responsible for administering the application. From an MQ Workflow point of view, the invocation of an activity implementation was successful when the invocation message is successfully put into the UPES queue.

For more information on how to create and administer MQSeries queues, refer to the *IBM MQSeries System Administration*.

Messages sent to a UPES: Following messages are sent to a UPES by MQ Workflow, depending on the version of the UPES:

Message	UPES Version	Remarks
ActivityImplInvoke	Version 3.2.2 and higher	Message to invoke an activity implementation. A response is required in case of a synchronous invocation.
ActivityExpired	Version 3.3.0 and higher	Message to inform about the expiration of an activity. No response is expected.

Message	UPES Version	Remarks
TerminateProgram	Version 3.3.0 and higher	Message to inform about the termination of an activity. No response is expected.

In general, when a message has been placed into a UPES input queue, it becomes the responsibility of the UPES to get that message from the queue, process the message, and to put a response message into the reply queue if requested. The UPES can use MQSeries transactional capabilities to bring get and put messages into a transactional context.

ActivityImplInvoke message: When the UPES receives an *ActivityImplInvoke* message, the UPES is asked to invoke the program specified in the *ImplementationData*[9] XML element.

It is up to the UPES how, when, and where to invoke that program. Note that the UPES has to remember the ActImplCorrelID of an incoming ActivityImplInvoke message so that it can be returned to MQ Workflow in the ActivityImplInvokeResponse and to correlate ActivityExpired or TerminateProgram messages to an invocation.

Depending on the nature of the activity instance, the activity implementation, that is, the specified program, may only need to be triggered and then runs asynchronously to the MQ Workflow process instance, or the process instance navigation has to be synchronized with its completion. In the latter case, a completion message has to be sent to MQ Workflow to inform it about the result of execution. In the former case, the activity instance is considered finished as soon as the invocation request is successfully sent. The information whether an implementation is to be started synchronously or asynchronously is modeled in Buildtime:

- Synchronous

  The activity implementation is started and the activity instance put into state *Running*. When the activity implementation ends and MQ Workflow receives a completion message, the activity instance is set into the appropriate state, for example, *Finished*.

  Correlation between the request and the response is done by means of the activity implementation correlation ID (XML element *ActImplCorrelID*), which is passed in the invocation request by MQ Workflow, and has to be passed back in the response.

  The *ResponseRequired* element in the MQ Workflow ActivityImplInvoke message is set to 'Yes' to specify that the invocation is synchronous and that MQ Workflow expects a response.
- Asynchronous

  The activity implementation is started and the activity instance is put into the appropriate state, for example, *Finished*. No information on the completion of the activity implementation is expected. If a completion message is received, it is ignored.

  The *ResponseRequired* element in the MQ Workflow ActivityImplInvoke message is set to 'No' to specify that the invocation is asynchronous and that MQ Workflow does not expect a response.

9. It is recommended to use the information from the ImplementationData XML element and to not misuse information from other elements.

The ActivityImplInvoke message provides the input container data as well as initial values set in the output container of the program to be invoked. The input container data is part of the *ProgramInputData* XML element; the initial values of the output container are part of the *ProgramOutputDataDefaults* XML element.

The UPES is responsible for processing of the container data, that is, to:

1. Pass the input container values to the activity implementation.
2. Copy the initial values of the output container into the output container (ProgramOutputData XML element) that is passed back to MQ Workflow. MQ Workflow only takes the values passed back in the ActivityImplInvokeResponse as output data; this allows values to be unset (to be set to null).
3. Copy the output values of the activity implementation into the output container (ProgramOutputData XML element) that is passed back to MQ Workflow.

Completion message: If the activity implementation is specified to run asynchronously, no completion message is expected. In that case, the successful *put* of the outgoing start activity implementation message is considered to be the complete invocation.

If the activity implementation is specified to run synchronously, a completion message ActivityImplInvokeResponse is expected by the MQ Workflow execution server. This message has to provide:

- The ActImplCorrelID of the associated ActivityImplInvoke message so that the MQ Workflow execution server can correlate the response with the Invoke request.
- For a successful execution, the return code and output container.
- For an unsuccessful execution, an exception passing the error code. The error code must be understood by MQ Workflow. See the file *fmcmretc.h* for a list of valid codes.

ActivityExpired message: This message informs the UPES about the expiration of an activity. It can only be sent after an ActivityImplInvoke message has been placed into the UPES input queue. If an ActivityImplInvokeResponse is expected, it informs the UPES that the response is not expected anymore. If the UPES sends a response, that response is ignored; the activity already expired. As a consequence, the UPES may interrupt program execution.

Correlation between the ActivityExpired message and the ActivityImplInvoke message can be done by means of the ActImplCorrelID.

TerminateProgram message: This message informs the UPES about the termination of an activity or work item. It can only be sent after an ActivityImplInvoke message has been placed into the UPES input queue. If an ActivityImplInvokeResponse is expected, it informs the UPES that the response is not expected anymore. If the UPES sends a response, that response is ignored; the activity is already terminated. As a consequence, the UPES may terminate program execution and perform compensation actions.

Correlation between the TerminateProgram message and the ActivityImplInvoke message can be done by means of the ActImplCorrelID.

Authorization

For invocation messages sent by MQ Workflow, the UserIdentifier field in the MQMD is set to the user ID of the user on whose behalf the activity instance has

been started. Additionally, the <Starter> element in the invocation message itself is set to that user ID. The UPES applications can use this information to implement their own authorization schemes.

Note: The CorrelID in the MQMD is also set to the user ID. This information can be used by a UPES to listen to XML messages for specific users only.

Synchronous invocation example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>Yes</ResponseRequired>
  </WfMessageHeader>
  <ActivityImplInvoke>
    <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
    <Starter>user1</Starter>
    <ProgramID>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProgramName>PerformOrder</ProgramName>
    </ProgramID>
    <ImplementationData>
       <ImplementationPlatform>AIX</ImplementationPlatform>
       <ProgramParameters>custNo=1234</ProgramParameters>
       <ExeOptions>
          <PathAndFileName>/usr/local/bin/perforder</PathAndFileName>
          <WorkingDirectoryName>/usr/local/data</WorkingDirectoryName>
          <InheritEnvironment>true</InheritEnvironment>
          <StartInForeGround>true</StartInForeGround>
          <AutomaticClose>true</AutomaticClose>
          <WindowStyleVisible>true</Visible>
          <RunInXTerm>true</RunInXTerm>
       </ExeOptions>
    </ImplementationData>
    <ImplementationData>
        <ImplementationPlatform>OS390</ImplementationPlatform>
        <ExternalOptions>
          <ServiceName>CICS42</ServiceName>
          <ServiceType>CICS</ServiceType>
          <InvocationType>EXCI</InvocationType>
          <ExecutableName>ORDR</ExecutableName>
          <ExecutableType>REG1</ExecutableType>
          <IsLocalUser>true</IsLocalUser>
          <IsSecurityRoutineCall>true</IsSecurityRoutineCall>
          <CodePage>850</CodePage>
          <TimeoutPeriod>TimeInterval</TimeoutPeriod>
          <TimeoutInterval>60</TimeoutInterval>
          <IsMappingRoutineCall>false</IsMappingRoutineCall>
        </ExternalOptions>
    </ImplementationData>
    <ProgramInputData>
        <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
        <_PROCESS>CreditRequest#123</_PROCESS>
        <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
        <CreditData>
          <Customer>
            <Name>User1</Name>
          </Customer>
          <Amount>1000</Amount>
          <Currency>CurrencyX</Currency>
        </CreditData>
    </ProgramInputData>
    <ProgramOutputDataDefaults>
        <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
        <_PROCESS>CreditRequest#123</_PROCESS>
        <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
```

```
        <CreditData>
          <Risk>high</Risk>
        </CreditData>
    </ProgramOutputDataDefaults>
  </ActivityImplInvoke>
<WfMessage>
```

UPES response example

```
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <ActivityImplInvokeResponse>
    <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
    <ProgramRC>0</ProgramRC>
    <ProgramOutputData>
       <CreditData>
          <Customer>
            <Name>User1</Name>
          </Customer>
          <Amount>1000</Amount>
          <Currency>CurrencyX</Currency>
          <Risk>low</Risk>
       </CreditData>
    </ProgramOutputData>
  </ActivityImplInvokeResponse>
</WfMessage>
```

Error Handling

This chapter describes how MQ Workflow handles errors which can occur during the processing of an incoming XML message.

MQ Workflow XML message life cycle

Each message that is received by MQ Workflow goes through a certain chain of processing steps. During each of these processing steps, errors can occur.

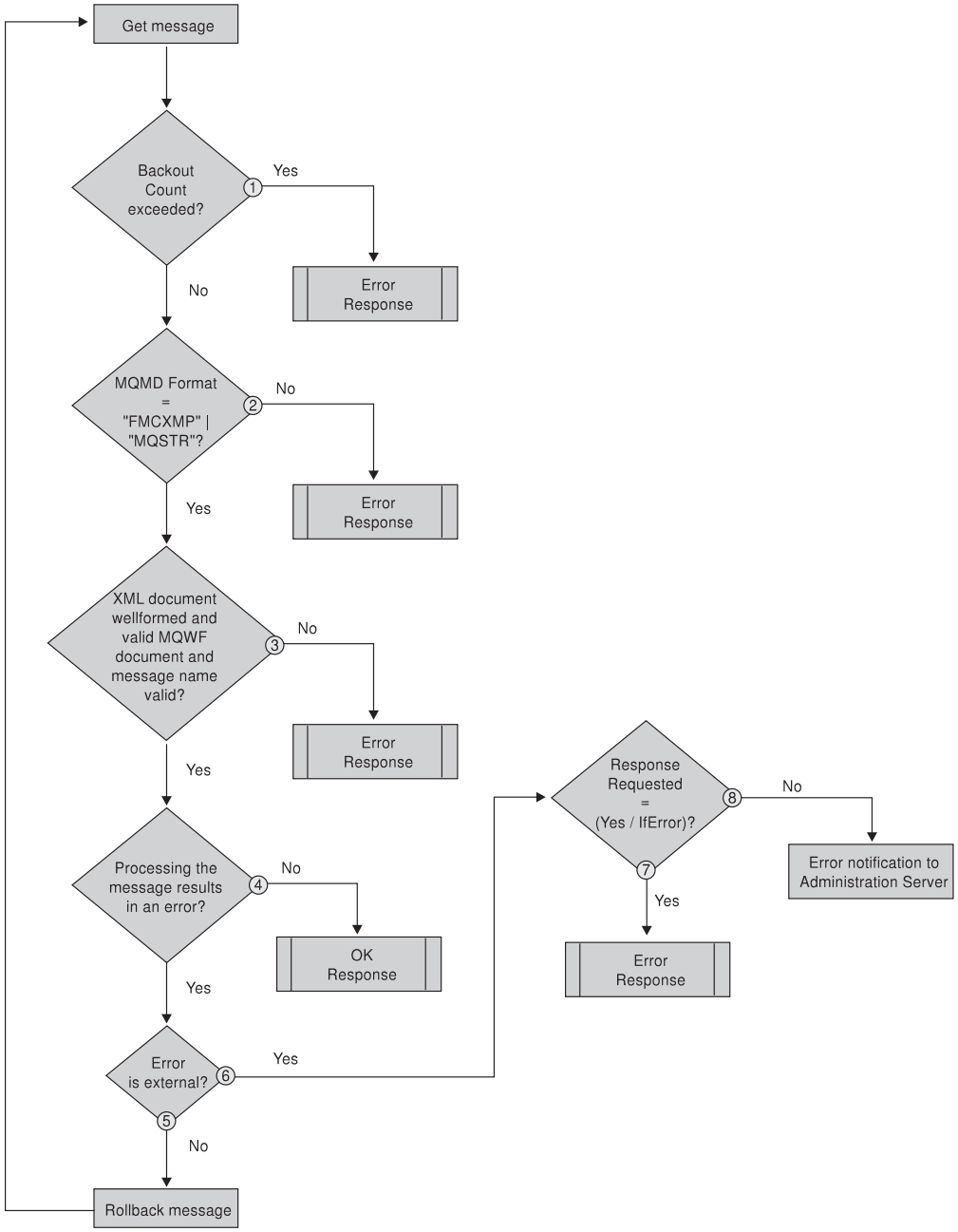
In general, the life cycle of an XML message within MQ Workflow can be described as follows:

1. Get the XML message from the MQ Workflow XML input queue
2. Parse the message
3. Determine the message name, for example, ProcessTemplateExecute
4. Check the validity of the application data parameters
   - Whether mandatory parameters are provided
   - Whether parameter values are syntactically correct
   - Whether semantical interdependencies are fulfilled
   - Whether container values fit to the container schema
   - Whether container values are syntactically correct
5. Process the message
6. Put the XML response message into the reply queue
7. Commit the transaction

When an error occurs, a response describing the error is created and put into the reply queue identified by the Reply2q and ReplyToQMgr fields in the MQMD fields of the input message. The reply queue address is abbreviated to REPLY2Q in the rest of this chapter.

General error processing

The following flowchart describes in more detail the life cycle of an MQ Workflow XML message and the errors which can occur and how they are handled.



1. When a message has been received from the XML input queue, then its backout count, that is, the number of times it has already been tried to process, is checked. If the backout count is exceeded, a GeneralError response message is returned. See “Sending a response” on page 177 for more information.

   Note: The backout count is increased when the transaction is rolled back - see 5 on page 177.

2. The MQMD Format field is checked for a correct value, namely, whether it is set to the MQSeries constant FMTMQ_STRING (MQSTR). If not, a GeneralError response is returned.

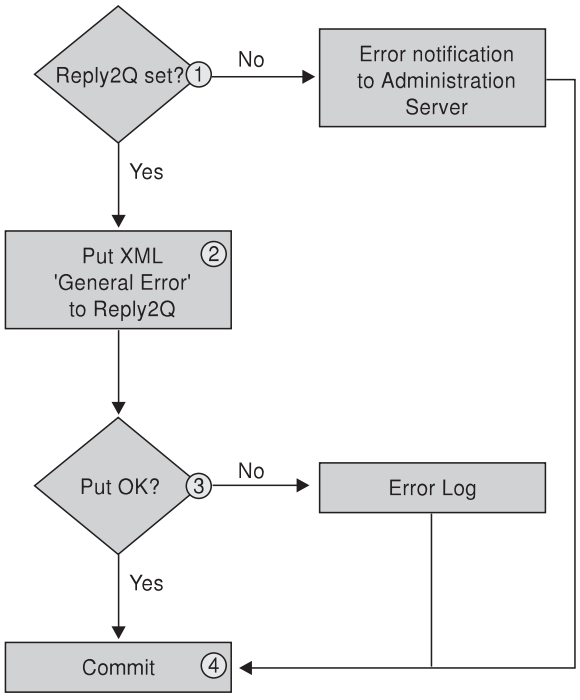
Note: For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

3. The XML message is checked for being well formed. Additionally, the message name must specify a function supported in the XML interface. If the XML message is not well formed or if the message name is not supported, a GeneralError response is returned.
4. The XML message is processed. If processing has been successful, the transaction is committed and a response denoting the successful processing is returned. If an error occurs, continuation depends on the kind of the error.
5. If the error is caused by internal reasons, for example, by a database lock, then MQ Workflow tries to newly process the message because such an error is usually only temporarily encountered.

   The message is rolled back so that it can be processed again. Backing out the message increases the backout count.

   Note: Depending on the severity of the error, rolling back a message can cause the MQ Workflow server to shut down.
6. If the error is caused by external reasons, for example, because a parameter is syntactically incorrect, then the response, if any, for the function requested describes the error.
7. A response describing an error is returned if the *ResponseRequired* element of the incoming message is set to 'Yes' or to 'IfError'.
8. Otherwise, an error notification which describes the error encountered is sent to the MQ Workflow administration server.

Sending a response:



When a response is to be returned to the sender of an XML message

1. It is checked whether the REPLY2Q is set. If not, an error notification is sent to the MQ Workflow administration server.

The administration server logs the error into a database. It can then be queried using the administration client. For more information refer to the *IBM MQSeries Workflow Administration Guide*, chapter "The error log".

2. If the REPLY2Q is set, the response is put into the specified reply queue.
3. If the 'Put' failed, an error log entry is written.
4. In either case, whether the 'Put' was successful or not, the transaction and thus the message read is committed so that the next message can be processed.

Detailed error processing

This chapter discusses some errors in more detail.

Wrong message format in the MQMD: The first time an error can occur is when the XML incoming message specifies an invalid Format field in the MQMD. Valid formats are MQFMT_STRING (MQSTR). For compatibility reasons, the format FMCXML is also supported; its usage is, however, deprecated.

If the *Format* field is wrong, a GeneralError XML response message is sent to the REPLY2Q. Refer to "The GeneralError message" on page 181 for the specification of a GeneralError message.

When putting the response into the reply queue fails, an error notification is sent to the MQ Workflow administration server. The original XML message is committed and thus removed from the input queue.

The error returned is:

```
:msgID.       FMC_ERROR_XML_DOCUMENT_FORMAT
:msgNum.      1107
:severity.    Error
:msgText.     "The MQMD format field value '%1$s' of the XML document is
              incorrect.\\n"
$1: The value of the MQMD format field
```

The following XML message is returned:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
            FMC01107E The MQMD format field value 'ABC ' of the XML document
            is incorrect.\n
        </MessageText>
        <Origin>p:\v322\src\fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

Wrong message name or XML document not well formed: The next time to detect an error is during parsing of the XML message. The XML message is checked for being well formed[10] and the message name is investigated.

When the XML message is not well formed or when the message name is unknown, a GeneralError response is sent to the REPLY2Q. When putting the response into the reply queue fails, an error notification is sent to the MQ Workflow administration server, that is, put into the ADMINPUTQ. The original XML message is committed and thus removed from the input queue.

Note that, at this stage, any setting of the *ResponseRequired* tag is ignored because its determination is not always possible.

Possible errors returned are:

```
:msgID.       FMC_ERROR_XML_DOCUMENT_INVALID
:msgNum.      1100
:severity.    Error
"Incorrect XML document. The message that is returned by the XML parser is %1$s\\n"
$1: The error message that occurred during parsing of the XML message

:msgID.       FMC_ERROR_NO_MQSWF_DOCUMENT
:msgNum.      1101
:severity.    Error
"The XML document is not a valid MQSeries Workflow XML document.\\n"

:msgID.       FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED
:msgNum.      1102
:severity.    Error
"MQSeries Workflow message '%1$s' is not XML enabled.\\n"
$1: The XML MQSeries Workflow message name, for example, ProcessTemplateDelete
```

Message processing errors: When processing the incoming XML message, the validity of the application data is checked. It is checked whether:

- The *ResponseRequired* tag in the XML message header is set correctly
- The 'UserContext' obeys its rules, that is, obeys its length constraints (<= 254 bytes)
- All parameters are correct:
  - Whether mandatory parameters are provided
  - Whether parameter values are syntactically correct
  - Whether semantical interdependencies are fulfilled
  - Whether container values fit to the container schema
  - Whether container values are syntactically correct

If an error occurs, the response message for the requested function is used to describe the error, for example, a ProcessTemplateCreateAndStartInstanceResponse. A GeneralError response is not sent.

If an error occurs and *ResponseRequired* is set to:

Yes A response is sent in any case.

IfError Only error responses are sent.

10. A well formed XML document is defined by the XML standard and guarantees a certain level of syntactical correctness. A valid MQ Workflow document additionally requires that the root element name is 'WfMessage' with an optional nested element 'WfMessageHeader' followed by the message name.

No No response is sent.

Note: Errors of incoming XML responses, for example, an ActivityImplInvokeResponse, are treated the same way as incoming requests. The only difference is that, in case of an error, a GeneralError is sent instead of a specific error response message.

XML processing error response example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
     <ResponseRequired>No</ResponseRequired>
   </WfMessageheader>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
      <RC>1105</RC>
      <Parameters>
         <Parameter>InsuredID</Parameter>
         <Parameter>Application</Parameter>
      </Parameters>
      <MessageText>FMC01105E Data member 'InsuredID' value of data structure
      'Application' has the wrong type.\n</MessageText>
      <Origin>d:\v32_67\src\fmcmctnm.cxx(340)</Origin>
    </Exception>
   </ProcessTemplateCreateAndStartInstanceResponse>
<WfMessage>
```

Errors when returning a response: When an XML response message is put into an MQSeries queue, an error can occur, for example, when the specified queue is undefined or when that queue is full.

Note: A similar error can occur when the UPES queue into which an ActivityImplInvoke message is to be put is not part of the MQ Workflow topology data.

When putting the response into the reply queue fails, the error and the first 500 bytes of the response are logged into an error log. The original XML message is committed and thus removed from the input queue. All *put* errors are assumed to be permanent, that is, it is assumed that all subsequent puts will also fail. Therefore, processing of the incoming message is not retried.

Example of an error log entry:

```
              MQSeries Workflow 3.2 Error Report

Report creation = 09.05.00 17:29:11

Error location  = File=e:\v322\src\fmccdxmm.cxx, Line=565,
Function=
FmcXMLMQDevice::Put(FmcDeviceCtxRef&,FmcDevDataRef&,FmcDeviceControler&)
Error data= FmcMQPUTException, MQ queue manager name=FMCQM, MQ queue name=DDDD,
            MQ completion code=2, MQ reason code=2085,
            Application data (frist 500 bytes)=
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>No</ResponseRequired>
   </WfMessageHeader>
   <UserContext>This data is sent back in response</UserContext>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
```

```
      <RC>1108</RC>
      <Parameters>
         <Parameter>ResponseRequired</Parameter>
```

Backout count exceeded: In general, the incoming XML message is committed after error handling is completed.

For internal errors like a database lock, the open transaction is, however, rolled back and the backout count is increased. When the backout count reaches a limit set in the runtime database, a GeneralError response message is put into the REPLY2Q and the incoming message is committed.

The error returned is:

```
:msgID.       FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED
:msgNum.      1106
:severity.    Error
:msgText.     "The backout count of the XML document is exceeded.
               The XML document cannot be processed.\\n"
```

The GeneralError message: A GeneralError response message is sent to inform the receiver that an error occurred and that a manual intervention might be required. No response is expected.

Following is an excerpt of the DTD describing the GeneralError XML message:

```
<!ELEMENT WfMessage
          WfMessageHeader?,
          GeneralError >
<!ELEMENT WfMessageHeader    (ResponseRequired?, UserContext?)   >
<!ELEMENT UserContext        (#PCDATA)                           >
<!ELEMENT ResponseRequired   (#PCDATA)                           >
          <!-- Expected values: {No,IfError,Yes}               -->
<!ELEMENT GeneralError       (Exception)                         >
<!ELEMENT Exception          (Rc?,Parameters?,MessageText,Origin)>
<!ELEMENT Parameters         (Parameter*)                        >
<!ELEMENT Parameter          (#PCDATA)                           >
<!ELEMENT Origin             (#PCDATA)                           >
```

For example,

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
           FMC01107E The MQMD format field value 'ABC ' of the XML document is
           incorrect.\n
        </MessageText>
        <Origin>p:\v322\src\fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

The MQ Workflow XML message format

The following XML syntax is used to describe the format of MQ Workflow messages. Note that the following format of a container only contains a suggestion, because the format can vary depending on your setup. Therefore, you cannot use this DTD description to validate your XML message without adding the appropriate specifications for the data structures you use.

Note that you do not have to specify your containers. You are, however, encouraged to do so for future use or to validate them by any other XML application.

```
<!-- FmcXMLIF.dtd == DTD for MQSeries Workflow messages -->
<!-- Message ======================================== -->
<!ELEMENT WfMessage
     ( WfMessageHeader?,
       ( ProcessTemplateCreateAndStartInstance
       | ProcessTemplateCreateAndStartInstanceResponse
       | ProcessTemplateExecute
       | ProcessTemplateExecuteResponse
       | ActivityExpired
       | ActivityImplInvoke
       | ActivityImplInvokeResponse
       | AuditTrailRecord
       | TerminateProgram
       | GeneralError ) ) >
<!-- ================================================
       Workflow Message Header
     ================================================ -->
<!ELEMENT WfMessageHeader           (ResponseRequired?,UserContext?)>
<!-- Opaque -->
<!ELEMENT UserContext               (#PCDATA)> <!-- Length<=254 bytes -->
<!-- Enumerated type -->
<!ELEMENT ResponseRequired          (#PCDATA)>
                                     <!-- Expected values: {No,IfError,Yes} -->
<!-- ================================================
       Specific Messages
     ================================================ -->
<!-- ProcessTemplateCreateAndStart ================== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
      ( ProcTemplName,
        ProcInstName?,
        KeepName?,
        ProcInstInputData? ) >
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
      ( ProcessInstance
      | Exception ) >
<!-- ProcessTemplateExecute ========================= -->
<!ELEMENT ProcessTemplateExecute
      ( ProcTemplName,
        ProcInstName?,
        KeepName?,
        ProcInstInputData? ) >
<!ELEMENT ProcessTemplateExecuteResponse
      ( ( ProcessInstance,
        ProcInstOutputData? )
      | Exception ) >
<!-- ActivityExpired ================================ -->
<!ELEMENT ActivityExpired
      ( ActImplCorrelID ) >
<!-- ActivityImplInvoke ============================= -->
<!ELEMENT ActivityImplInvoke
      ( ActImplCorrelID,
        Starter,
        ProgramID,
```

```
        (ImplementationData)*,
        ProgramInputData,
        ProgramOutputDataDefaults ) >
<!ELEMENT ActivityImplInvokeResponse
      ( ActImplCorrelID,
        (( ProgramRC,
           ProgramOutputData )
      | Exception )) >
<!-- AuditTrailRecord ============================== -->
<!ELEMENT AuditTrailRecord
      ( Timestamp,
        AuditEvent,
        ProcInstName,
        ProcInstID,
        ProcInstTopLevelName,
        ProcInstTopLevelID,
        ProcInstParentName?,
        ProcInstParentID?,
        ProcTemplName,
        ProcTemplValidFromDate?,
        BlockNames?,
        UserID?,
        SecondUserID?,
        ActivityName?,
        ActivityType?,
        ActivityState?,
        SecondActivityName?,
        CommandParameters?,
        AssociatedObject?,
        ObjectDescription?,
        ProgramName?,
        ProgramRC?  ) >
<!-- TerminateProgram ============================== -->
<!ELEMENT TerminateProgram
      ( ActImplCorrelID ) >
<!-- GeneralError ================================== -->
<!ELEMENT GeneralError  (Exception) >
<!-- ================================================
           Data Structures
     ================================================ -->
<!ENTITY %STRING  "(#PCDATA)">
<!ENTITY %LONG    "(#PCDATA)">
<!ELEMENT null EMPTY>

<!ELEMENT _PROCESS_INFO
          ( Role?, Organization?, ProcessAdministrator?, Duration? )>
<!ELEMENT _ACTIVITY_INFO
          ( PRIORITY?, MembersOfRoles?, CoordinatorOfRole?,
            Organization?, OrganizationType?,
            LowerLevel?, UpperLevel?,
            People?, PersonToNotify?,
            Duration?, Duration2? )>
<!ELEMENT _ACTIVITY            %STRING;>
<!ELEMENT _PROCESS             %STRING;>
<!ELEMENT _PROCESS_MODEL       %STRING;>
<!ELEMENT _RC                  %LONG;>
<!ELEMENT ROLE                 %STRING;>
<!ELEMENT Organization         %STRING;>
<!ELEMENT ProcessAdministrator %STRING;>
<!ELEMENT Priority             %STRING;>
<!ELEMENT MembersOfRoles       %STRING;>
<!ELEMENT CoordinatorOfRole    %STRING;>
<!ELEMENT OrganizationType     %LONG;>
<!ELEMENT LowerLevel           %LONG;>
<!ELEMENT UpperLevel           %LONG;>
```

XML interface

```
<!ELEMENT People               %STRING;>
<!ELEMENT PersonToNotify       %STRING;>
<!ELEMENT Duration             %LONG;>
<!ELEMENT Duration2            %LONG;>

<!ENTITY %_CONTAINER_INFO
         "_RC?, _PROCESS?, _PROCESS_MODEL?, _ACTIVITY?,
          _PROCESS_INFO?, _ACTIVITY_INFO?" >

<!-- ====================================================
      Sample Data Structure CreditData
===================================================== -->
<!ELEMENT CreditData
          ( Customer, Amount?, Currency?, Risk? ) >
<!ELEMENT Risk     %LONG;>
<!ELEMENT Currency %STRING;>
<!ELEMENT Amount   %LONG;>
<!ELEMENT Customer %STRING;>

<!-- ====================================================
      Sample Entity Container
      any used data structure must be included, for example,
      <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
===================================================== -->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">

<!ELEMENT ProcInstInputData          %CONTAINER;>
<!ELEMENT ProcInstOutputData         %CONTAINER;>
<!ELEMENT ProgramInputData           %CONTAINER;>
<!ELEMENT ProgramOutputData          %CONTAINER;>
<!ELEMENT ProgramOutputDataDefaults  %CONTAINER;>

<!-- Process Instance ================================= -->
<!ELEMENT ProcessInstance
      ( ProcInstID,
        ProcInstName,
        ProcInstParentName?,
        ProcInstTopLevelName,
        ProcInstDescription?,
        ProcInstState,
        LastStateChangeTime,
        LastModificationTime,
        ProcTemplID,
        ProcTemplName,
        Icon,
        Category? ) >

<!-- Program ID ======================================== -->
<!ELEMENT ProgramID
      ( ProcTemplID,
        ProgramName ) >

<!-- Implementation Data ============================== -->
<!ELEMENT ImplementationData
      ( ImplementationPlatform ,
        ProgramParameters,
        ( ExeOptions
        | DllOptions
        | ExternalOptions ) ) >

<!ELEMENT ExeOptions
      ( PathAndFileName,
        WorkingDirectoryName?,
        Environment?,
        InheritEnvironment,
        StartInForeGround?,
        AutomaticClose?,
        WindowStyle?,
        RunInXTerm? ) >
```

```
<!ELEMENT DllOptions
      ( PathAndFileName,
        EntryPointName,
        ExecuteFenced?,
        KeepLoaded? )

<!ELEMENT ExternalOptions
      ( ServiceName,
        ServiceType,
        InvocationType,
        ExecutableName,
        ExecutableType,
        IsLocalUser,
        IsSecurityRoutineCall,
        CodePage?,
        TimeoutPeriod,
        TimeoutInterval?,
        IsMappingRoutineCall,
        MappingType?,
        ForwardMappingFormat?,
        ForwardMappingParameters?,
        BackwardMappingFormat?,
        BackwardMappingParameters? ) >
<!-- Exception ======================================== -->
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
<!ELEMENT Parameters
          (Parameter*) >
<!-- Data Elements ==================================== -->
<!-- Booleans -->
<!ELEMENT AutomaticClose        (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT DllV2Compatible       (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT ExecuteFenced         (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT InheritEnvironment    (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsLocalUser           (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsMappingRoutineCall  (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT IsSecurityRoutineCall (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT KeepLoaded            (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT KeepName              (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT RunInXTerm            (#PCDATA) > <!-- Expected values: {true, false} -->
<!ELEMENT StartInForeGround     (#PCDATA) > <!-- Expected values: {true, false} -->
<!-- Strings -->
<!ELEMENT ActivityName              (#PCDATA) >
<!ELEMENT AssociatedObject          (#PCDATA) >
<!ELEMENT BackwardMappingFormat     (#PCDATA) >
<!ELEMENT BackwardMappingParameters (#PCDATA) >
<!ELEMENT BlockNames                (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT CommandParameters         (#PCDATA) >
<!ELEMENT EntryPointName            (#PCDATA) >
<!ELEMENT Environment               (#PCDATA) >
<!ELEMENT ExecutableName            (#PCDATA) >
<!ELEMENT ExecutableType            (#PCDATA) >
<!ELEMENT ForwardMappingFormat      (#PCDATA) >
<!ELEMENT ForwardMappingParameters  (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT InvocationType            (#PCDATA) >
<!ELEMENT MappingType               (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT ObjectDescription         (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT PathAndFileName           (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
```

```
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgramName               (#PCDATA) >
<!ELEMENT ProgramParameters         (#PCDATA) >
<!ELEMENT SecondActivityName        (#PCDATA) >
<!ELEMENT SecondUserID              (#PCDATA) >
<!ELEMENT ServiceName               (#PCDATA) >
<!ELEMENT ServiceType               (#PCDATA) >
<!ELEMENT Starter                   (#PCDATA) >
<!ELEMENT WorkingDirectoryName      (#PCDATA) >
<!-- Opaque -->
<!ELEMENT ActImplCorrelID           (#PCDATA) > <!-- Length =  80 bytes -->
<!ELEMENT ProcInstID                (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcInstParentID          (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcInstTopLevelID        (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcTemplID               (#PCDATA) > <!-- Length <= 64 bytes -->
<!-- Numbers -->
<!ELEMENT CodePage                  (#PCDATA) >
<!ELEMENT ProgramRC                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT TimeoutInterval           (#PCDATA) >
<!-- Timestamps YYYY-MM-DD-hh.mm.ss.000000 (000000 milliseconds) -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >
<!ELEMENT ProcTemplValidFromDate    (#PCDATA) >
<!ELEMENT Timestamp                 (#PCDATA) >
<!-- Enumerated types -->
<!ELEMENT ImplementationPlatform (#PCDATA) > <!--  Expected values:
                                                  { OS2,       AIX,
                                                    HPUX,      Windows95,
                                                    WindowsNT, OS390,
                                                    Solaris } -->

<!ELEMENT ProcInstState          (#PCDATA) > <!--  Expected values:
                                                  { Ready,     Running,
                                                    Finished,  Terminated,
                                                    Suspended, Terminating,
                                                    Suspending,Deleted } -->

<!ELEMENT WindowStyle            (#PCDATA) > <!-- Expected values:
                                                  { Visible,   Invisible,
                                                    Minimized, Maximized } -->

<!ELEMENT TimeoutPeriod          (#PCDATA) > <!-- Expected values:
                                                  { TimeInterval
                                                    Forever    Never } -->
<!ELEMENT AuditEvent             (#PCDATA)  > <!-- Expected values:
                                                   { 21000, 21001, 21002 ....
                                                     see Administration Guide } -->

<!ELEMENT ActivityType           (#PCDATA)  > <!-- Expected values:
                                                   { Program Activity,
                                                     Process Activity,
                                                     Block Activity } -->
<!ELEMENT ActivityState          (#PCDATA)  > <!-- Expected values:
                                                   { 21200 (Ready),
                                                     21201 (Running),
                                                     21202 (Finished),
                                                     21203 (CheckedOut),
                                                     21204 (Force-Finished),
                                                     21205 (Terminated),
                                                     21206 (Suspended),
                                                     21207 (InError),
                                                     21208 (Executed),
                                                     21209 (Skipped),
                                                     21210 (Deleted),
```

```
21211 (Suspending),
21212 (Terminating),
21213 (Expired) } -->
```

XML interface

Chapter 3. Interfacing with the Program Execution Server

This chapter describes how to write exit routines to interface with the Program Execution Server to map data for legacy applications and to invoke programs.

CICS considerations

In CICS activity implementations, input/output container data is not retrieved from the PEA as in the LAN version but rather sent with the invocation and stored in the COMMAREA. If the user changes the COMMAREA without using the IBM MQSeries Workflow for OS/390 API or once the output container has been set, this data can no longer be retrieved.

For information on how to enable CICS to work with IBM MQSeries Workflow for OS/390, see *IBM MQSeries Workflow: Concepts and Architecture*.

IMS considerations

In IMS activity implementations, input/output container data is not retrieved from the PEA as in the LAN version but rather sent with the invocation and stored in the I/O AREA. If the user changes the I/O AREA without using the IBM MQSeries Workflow for OS/390 API, or once the output container has been set, this data can no longer be retrieved.

IMS programs can use only the Container API, which is a subset of the full MQ Workflow API. The Container API is defined in header files *fmcjccon.h* and *fmcjpcon.hxx* and COBOL copybook *fmcperfl.cpy*.

For information on how to enable IMS to work with IBM MQSeries Workflow for OS/390 see *MQSeries Workflow for z/OS: Customization and Administration*.

Notes:

1. An IMS program using the MQSeries Workflow for OS/390 API must issue *at least one* of the following calls:
   - FmcjContainerInContainer
   - FmcjContainerOutContainer
   - FmcjContainerRemoteInContainer
   - FmcjContainerRemoteOutContainer
2. An IMS program using the MQSeries Workflow for OS/390 API must issue *exactly one* of the following calls to return control to the Workflow system:
   - FmcjContainerSetOutContainer
   - FmcjContainerSetRemoteOutContainer

Program mapping via the Program Execution Server

Introduction

Whenever legacy applications are to be invoked by MQSeries Workflow, a mapping of the MQSeries container data into a format acceptable by the legacy application is needed.

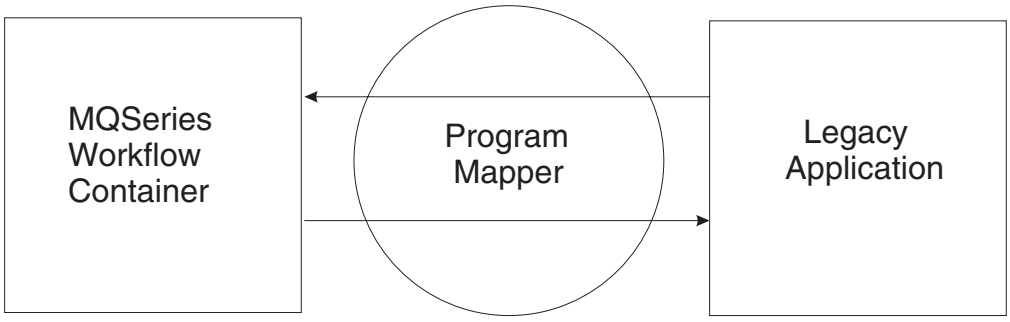


Figure 23. Program mapping illustration

MQSeries Workflow offers a default program mapper with basic functionality. This section gives a brief overview of the program mapping component of the program execution server (PES). This component does the mapping of MQSeries Workflow containers into a format acceptable by legacy applications. This is a basic mapper so that legacy applications like IMS and CICS are supported. If more complicated mappings are to be done, other mapping tools can be used.

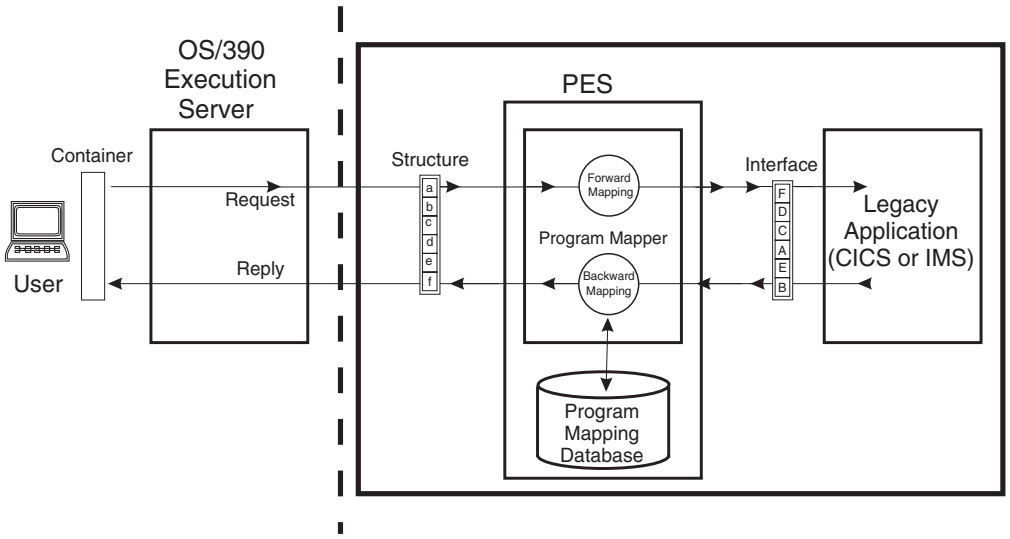


Figure 24. Program mapping control flow

In order to make the format of workflow containers acceptable to legacy applications, the content of the workflow containers is put to an interface called *structure*. The input/output interface for the legacy application is called the *interface*. The task for the program mapper is to convert the data between the structure and interface. Mapping from MQSeries Workflow to the legacy application (a to A, b to B...) is called *forward mapping*, and mapping from legacy applications to MQSeries Workflow (A to a, B to b...) *backward mapping*. If special conversion between structure and interface elements is needed, a usertype exit (which will be explained later) can be used.

Mapping is not necessary if the called application uses Workflow API calls to extract data from the containers.

The way the mapping should be done between structures and interfaces is defined with a **mapping definition language (MDL)**.

To create a mapping, you must first write the definition of the structure and interface elements. You then connect these structures and interfaces with forward/backward mapping definitions, compile the MDL with a parser, and load it into the mapping database. The elements of the mapper are explained in detail in the *MQSeries Workflow for z/OS: Customization and Administration*. The following graphic illustrates the process:

Figure 25. How to create a program mapping

Program mapping definitions

In this section, the mapping definitions will be explained in more detail. For each definition, a simple example is also given.

Structure definition

A structure defines the MQSeries Workflow container structure that is passed into the program execution server (PES). The structure definition syntax is identical to the structure definition syntax used in the Flowmark definition language (FDL). This allows exporting container definitions from Buildtime into a flat file and copying these structure definitions into the mapping definition language (MDL). A structure mainly consists of a collection of members (structure elements) with a type and cardinality.

Example: This example shows a container in MQSeries Workflow representing an account representative structure. The structure contains the name of the holder of the account (first name and last name defined as a string), the corresponding ZIP (postal) code (defined as long), salary, and tax. The last part of the container is to be filled with the data of customers belonging to the holder of the account. The example for the definition of the CustomerStructure is given later. In order to define the structure you must define each element of the MQSeries Workflow container which is to be passed to the legacy application (the code for a sample legacy application is given in "Additional mapping examples" on page 218).

```
STRUCTURE AccountRepStructure
          LastName:   STRING;
          FirstName:  STRING;
          Zip:        LONG;
          Salary:     FLOAT;
          Tax:        FLOAT;
          Customers:  CustomerStructure(3);
END AccountRepStructure
```

You will find a more detailed example under "Simple data structure with default name mapping" on page 220 and the structure definition grammar under "Structure definition" on page 206.

Interface definition

An interface defines the layout and type of the data accepted by a legacy application. Each interface element has a fixed size and location (offset) and will be filled with converted container elements. There is no way to verify whether the size, location, and type of the elements actually match the size, location, and type expected by the legacy application. This means that the interface definitions must be created carefully. Otherwise, conversion results are unpredictable and runtime mapping errors can occur because of invalid data. Each element of an interface is

mapped to a structure element with the same name. If there is no element with the same name, the interface element is skipped and the container element is unaffected. It is also possible to define a constant for an interface element. See "Constants" on page 197 for more details.

Example: This example shows an interface of a legacy application representing an account representative structure. In this case the name of the holder of the account (first name and last name, defined as a string with a maximum of 50 characters, terminated by hex zero and left-justified with pad character " "), the corresponding ZIP (defined as an unsigned integer with 16 bits), salary, and tax (defined as float with 32 bits). The last part of the container is to be filled with the data of selected customers belonging to the holder of the account. The example for the definition of the CustomerStructure (array for 3 customers using another structure "CustomerInterfaceForCpp") is given later. In order to define the interface, you must define each individual element of the container used by the legacy application. The definition of the interface should be as follows:

```
INTERFACE AccountRepInterfaceForCpp
          LastName:   CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
          FirstName:  CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
          Zip:        UNSIGNED INTEGER 16;
          Salary:     FLOAT 32;
          Tax:        FLOAT 32;
          Customers:  ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
```

You will find a more detailed example under "Simple data structure with default name mapping" on page 220 and the interface definition grammar under "Interface definition" on page 207. If you have an interface element and no corresponding structure element, no mapping will be done by the default mapper. If it is required to have some constants on the legacy application side, each interface element can optionally have a *constant* statement that defines the constant to create for the legacy application. The constant is converted in forward mapping, whether there is a matching structure element or not. If backward mapping occurs, a structure element is set to this constant only if there is an element with the same name or a mapping rule between the structure and the interface with a constant. See "Constants" on page 197 for more information.

Forward/backward mapping definition

The connection between a structure and an interface is done via forward and backward mapping definitions. Forward mapping is used to map a structure into a format accepted by a legacy application, and backward mapping is used to map legacy application data into a structure. Mapping done for structure and interface elements with identical names is called *default mapping*. In addition, it is possible, by using rules, to do *explicit mapping* of elements which have different names. Structures are mapped as a whole into interfaces, and arrays are mapped as a whole if both the structure and interface array have the same size. It is not possible to map array elements individually. If more powerful mappings are required, use container mapping (see *IBM MQSeries Workflow: Getting Started with Buildtime*).

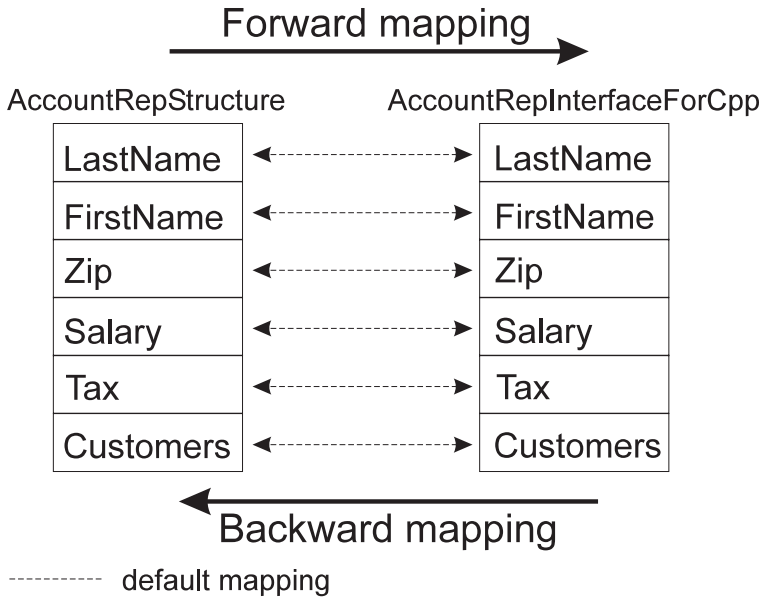


Figure 26. Default forward/backward mapping

To create a forward mapping, you must define which structure is to be mapped to which interface. To create a backward mapping, you must define which interface is to be mapped to which structure. Optionally, you can use rules to map elements with different names. See "Mapping algorithm" on page 194 for more detailed information about default and explicit mapping.

The coding for the mapping according to the diagram would therefore be:

```
/* Mapping from MQSeries Workflow (structure) to legacy appl.(interface) */
FORWARDMAPPING Forward
        FROM AccountRepStructure TO AccountRepInterfaceForCpp
END
/* Mapping from legacy appl. (interface) to MQSeries Workflow (structure) */
BACKWARDMAPPING Backward
        FROM AccountRepInterfaceForCpp TO AccountRepStructure
END
```

Note: All structure and interface elements are mapped because they have identical names (default mapping).

You will find a complete mapping in "Example" on page 198.

Usertype definition

A usertype can be used by the program mapper whenever the interface types provided by a default mapper do not offer the required conversion. In this case the actual data conversion must be done by a usertype exit, which must reside in a DLL. See "Usertype" on page 214.

Example: In order to assign a number to a currency with the corresponding symbol, you need to define a usertype (any mapping type would map the number to the exact number in a different format but not assign it to a special currency). It is also possible to define a usertype that calculates a value of currency A used in the structure to a currency B used in the interface (for example U.S. dollars to British pounds or the new European currency, the euro).

```
USERTYPE SampleUsertype LENGTH(4)
         DLL "SAMPUTY","SampleUsertypeExit"
END
```

Program mapping

```
INTERFACE SampleUsertypeInterface
          DESCRIPTION "Sample Usertype Interface"
          SampleElement: USERTYPE SampleUsertype PARMS "$";
END
```

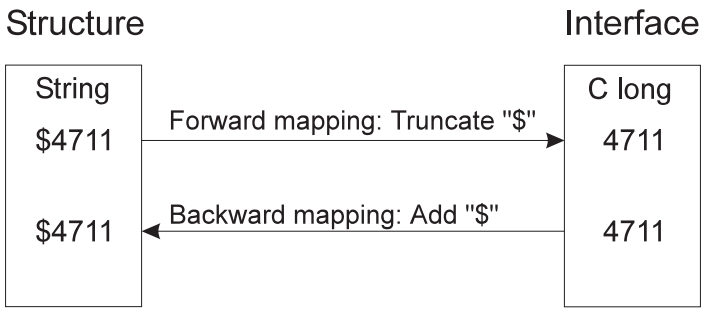


Figure 27. Usertype example

This example shows the functionality of a usertype. An interface element (defined as C long) is mapped to a structure element (defined as a string). In backward mapping, the C long ″4711″ is converted to a string and prefixed with ″$″. In forward mapping the string ″$4711″ is truncated to ″4711″ and then converted to a C long.

Mapping algorithm

All elements in an MQSeries Workflow container have names. The interface elements must also have names. Mapping is done per default on a name-by-name basis if elements have the same name. If element names are different, mapping rules can be used to do explicit mapping.

Structures are mapped as a unit to interfaces if their names are identical. Arrays are also mapped as a unit. It is also possible to define constants that are inserted on the legacy application or container side.

If structure or interface elements are not mapped, the data in the structure element or interface element is not modified.

Note: Each structure element can only be mapped to one interface element and vice versa.

In this example, there are 4 structure and interface elements which are to be mapped by the default mapper.

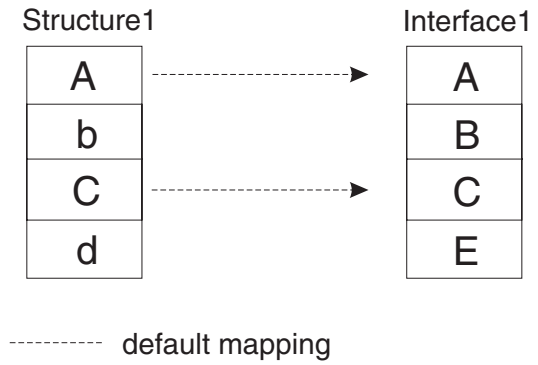


Figure 28. Default forward mapping illustration

```
FORWARDMAPPING Forward
     FROM Structure1 TO Interface1
END
```

The default mapper maps structure element A to interface element A and structure element C to interface element C. It does *not* map structure element b to interface element B or structure element d to interface element E, because of the different names. See also "Simple data structure with default name mapping" on page 220 for a more detailed example.

If the corresponding structure and interface elements do not have identical names, the mapping must be defined explicitly. In this case you must define an additional mapping rule for the mapping. The mapping definition language (MDL) and the corresponding grammar are explained in "Grammar" on page 203. The following graphic displays a simple forward mapping with some non-identical names of the structure and interface elements.

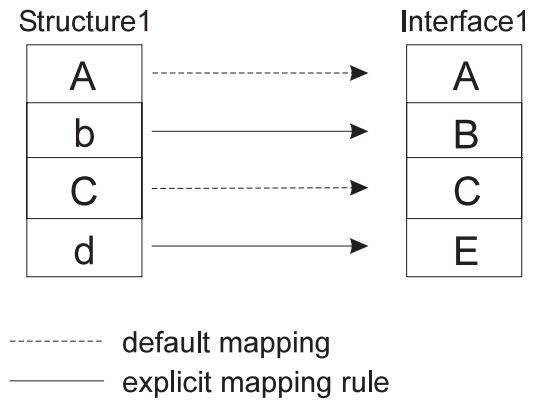


Figure 29. Forward2: Non-default forward mapping illustration

The mapping rules would follow as:

```
FORWARDMAPPING Forward2 FROM Structure1 TO Interface1
        MAP b TO B;
        MAP d TO E;
END
```

Structure elements A and C do not have to be mapped to interface elements A and C explicitly; this will be done be the default mapper automatically. Refer to "Complex data structure with non-default name mapping" on page 221 for a more detailed example.

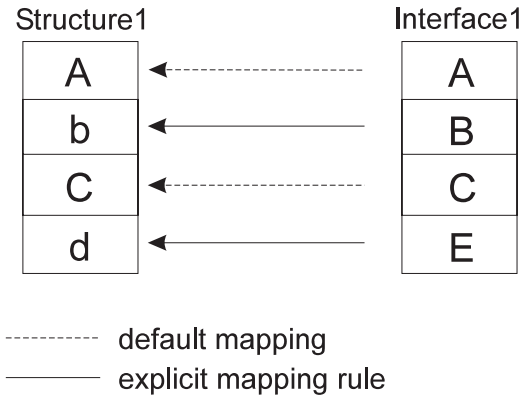


Figure 30. Non-default backward mapping Backward1 illustration

If you would like a backward mapping according to the above diagram the mapping rules would be:

```
BACKWARDMAPPING Backward1 FROM Interface1 TO Structure1
        MAP B TO b;
        MAP E TO d;
END
```

How you map elements to each other only depends on the definition as long as you do not violate any conversion rules (see "Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types" on page 200). For example. it is not permissible to map an integer to a binary.

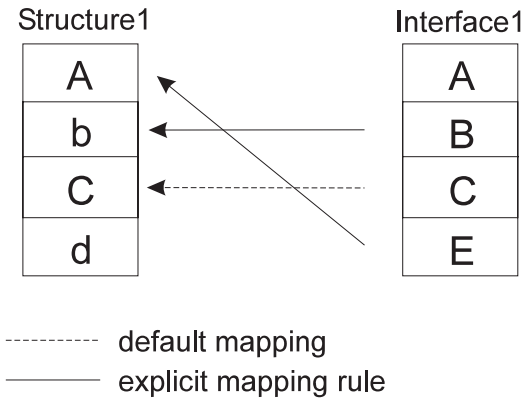


Figure 31. Backward2: Explicit mapping illustration

In this case the interface element A will not be mapped to structure element A because there is a rule from interface element E to structure element A. Interface element B is mapped because of the explicit rule. Interface element C is mapped to structure element C because they have the same name (default mapping) and

because no explicit mapping for interface element C is defined. Interface element E will be mapped to structure element A because of a mapping rule for interface element E.

```
BACKWARDMAPPING Backward2 FROM Interface1 TO Structure1
        MAP B TO b;
        MAP E TO A;
END
```

Table 10. Rule mapping with no constant definition

	BACKWARDMAPPING	FORWARDMAPPING
There is a definition rule for forward/backward mapping	Map interface element to structure element	Map structure element to interface element
There is *no* definition rule for forward/ backward mapping	Interface element not mapped to structure element	Interface element undefined

Notes:

1. If the mapping rules use invalid or nonexistent interface element names, these rules are ignored during actual mapping. Make sure you use the right names in forward mapping and backward mapping. By contrast, structure element names used in definition rules must exist. Otherwise, runtime errors will occur.
2. Do not map structure elements to interface elements which are used in other arrays. The structure element will contain the interface elements with the largest dimension.

Constants

If it is required to have constants on the legacy application or structure side, each *interface* element can optionally have a constant statement that defines the constant to create for the legacy application or structure element. The constant is converted in forward mapping whether there is a matching structure element or not. If a backward mapping occurs, the structure element is set to this constant only if there is an element with the same name or a rule is defined for this structure element.

Table 11. Mapping with constant definition

	Backward mapping	Forward mapping
There is a definition rule for forward/ backward mapping	Structure element set to constant	Interface element set to constant
There is *no* definition rule for forward/ backward mapping	Structure element **not** set to constant	Interface element set to constant

Example for non-default forward mapping with constant definitions:

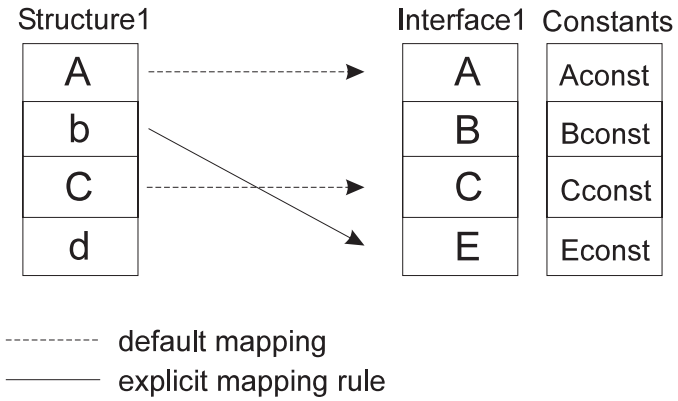


Figure 32. Forward mapping with constants

Because structure elements A, C and b are mapped to interface elements A, C and E (which all have a constant definition), the interface elements A, C and E are set to Aconst, Cconst and Econst, respectively. The interface element B is set to Bconst because no structure element was mapped to B. So all interface elements are set to their corresponding constant values. In forward mapping, all interface elements with a constant definition are set to their constant whether there is a mapping to this element or not. Only if the interface element has *no* constant definition can a mapping change the value. Refer to "Simple data structure with all interface types with CONSTANTS and usertypes" on page 223 for a more complex example.

Example for default backward mapping with constant definitions:

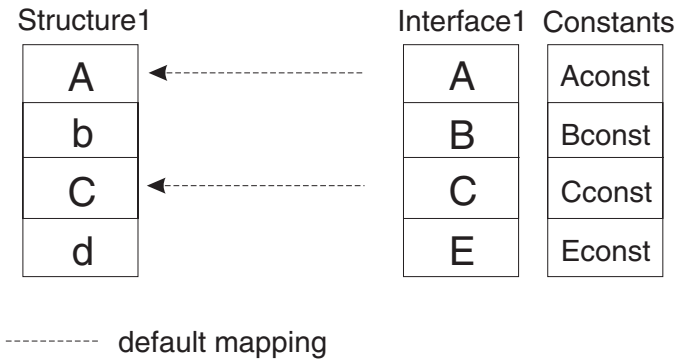


Figure 33. Backward mapping with constants

Because interface elements B and E are not mapped to structure elements b and d, b and d are *not* set to the constant values of Bconst and Econst. Because interface element A is mapped to structure element A and interface element A has a constant Aconst, the structure element A is set to Aconst (same as structure element C is set to Cconst). Assuming there would be no constant definition for C on the legacy application side, interface element C would have been mapped to structure element C as usual.

Example

In this example, the structure elements and interface elements do not have the same names. Therefore they must be mapped explicitly in the forward/backward mapping definition. If they were not mapped explicitly, no mapping would be done at all, because all structure and interface elements have different names.

```
STRUCTURE AccountRepStructure
          LastName:   STRING;
          FirstName:  STRING;
          Zip:        LONG;
          Salary:     FLOAT;
          Tax:        FLOAT;
          Customers:  CustomerStructure(3);
END AccountRepStructure
INTERFACE AccountRepInterfaceForCpp
          L: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
          F: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
          Z: UNSIGNED INTEGER 16;
          S: FLOAT 32;
          T: FLOAT 32;
          C: ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
/* Mapping from MQSeries Workflow (structure) to legacy application (interface) */
FORWARDMAPPING Forward FROM AccountRepStructure TO AccountRepInterfaceForCpp
         MAP LastName TO L;
         MAP FirstName TO F;
         MAP Zip TO Z;
         MAP Salary TO S;
         MAP Tax TO T;
         MAP Customers TO C;
END
/* Mapping from legacy appl. (interface) to MQSeries Workflow (structure) */
BACKWARDMAPPING Backward FROM AccountRepInterfaceForCpp TO AccountRepStructure
         MAP L TO LastName;
         MAP F TO FirstName;
         MAP Z TO Zip;
         MAP S TO Salary;
         MAP T To Tax;
         MAP C TO Customers;
END
```

Supported program mapping definition element types

Program mapping structure definition element types

All MQSeries Workflow element types are supported:

- LONG
- FLOAT
- STRING
- BINARY

Program mapping interface definition element types

Characters: Characters have a size in bytes, an optional termination character (hex 0), and justification and padding.

Integer numbers: Integers are either signed or unsigned and have a size in bits. Supported sizes are 16 and 32 bits.

Float numbers: Floats have a size in bits. Supported sizes are 32 and 64 bits.

Packed numbers: Packed numbers have a size, are either signed, with a character for plus and a character for minus, or unsigned with an unsigned character, and have a scale.

Zoned numbers: Zoned numbers have a size, are either signed, with a character for plus and a character for minus, or unsigned with an unsigned character, and have a scale.

Interface: Interfaces have only a name and define another interface used as an interface element. In this way, it is possible to structure the interface in the same way structures can be defined to contain other structures.

Usertypes: Whenever the previous interface types do not match the required types, it is possible to define a usertype. For details see "Grammar" on page 203. Usertypes have a name and an optional parameter string, which can be used to pass additional information to the usertype exit. In order to use a usertype, it must be defined and a usertype DLL with a usertype exit must be provided. (See "Usertype" on page 214 for more details):

Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types: The following table lists all possible combinations of structure elements and interface elements. If they are arrays, they must have the same size. There is one exception: a character of size 1 can be mapped to an MQSeries Workflow LONG. If an invalid combination is used, a runtime error will be generated (see *MQSeries Workflow for z/OS: Messages and Codes* for detailed information).

Table 12. Mapping combinations

Workflow Type Interface Type	String	Binary	Long	Float
CHAR	*	*	*	
INTEGER	*		*	
FLOAT				*
PACKED	*		*	*
ZONED	*		*	*
USERTYPE	*1	*1	*1	*1
Note: [1]Only available if this type of combination is supported by the *usertype* exit				

Table 13. C/C++ data type mappings (legacy application (C/C++) to FDL types (structure))

C/C++ type	Interface	Structure	Comment
char	CHAR(1) JUSTIFY LEFT PAD " "	STRING	No imbedded x'00'
char [5]	CHAR(5) JUSTIFY LEFT PAD " " or CHAR(5) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD " "	STRING	No imbedded x'00'
char	CHAR(1) JUSTIFY LEFT PAD " "	BINARY	
char [5]	CHAR(5) JUSTIFY LEFT PAD " " or CHAR(5) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD "<h00>"	BINARY	
char	CHAR(1) JUSTIFY LEFT PAD "<h00>"	LONG	
short	SIGNED INTEGER 16	LONG, STRING	
unsigned short	UNSIGNED INTEGER 16	LONG, STRING	
int	SIGNED INTEGER 32	LONG, STRING	
unsigned int	UNSIGNED INTEGER 32	LONG, STRING	
long	SIGNED INTEGER 32	LONG, STRING	
unsigned long	UNSIGNED INTEGER 32	LONG, STRING	
float	FLOAT 32	FLOAT	
double	FLOAT 64	FLOAT	

Table 14. COBOL data type mappings (legacy application (COBOL) to FDL types (structure))

COBOL	Interface	Structure	Comment
PIC X(n)	CHAR(n) JUSTIFY LEFT PAD " "	STRING	No imbedded x'00'
PIC X(n)	CHAR(n) JUSTIFY LEFT PAD "<h00>"	BINARY	
PIC X(1)	CHAR(1) JUSTIFY LEFT PAD "<h00>"	LONG	
PIC S999 PACKED-DECIMAL or COMP-3	PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE 0	LONG, STRING, FLOAT	
PIC S999PP PACKED-DECIMAL or COMP-3	PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE 2	LONG, STRING, FLOAT	
PIC S99V9 PACKED-DECIMAL or COMP-3	PACKED(3) SIGNED MINUS "<h0d>" PLUS "<h0c>" SCALE -1	LONG, STRING, FLOAT	
PIC S9999 BINARY or COMP-4	SIGNED INTEGER 16	LONG, STRING, FLOAT	Up to 4 digits[1]
PIC S9(6) BINARY or COMP-4	SIGNED INTEGER 32	LONG, STRING, FLOAT	Between 5 and 9 digits[1]
COMP-1	FLOAT 32	FLOAT	
COMP-2	FLOAT 64	FLOAT	
PIC S9(n)V9(m) DISPLAY	ZONED(n+m) SIGNED LAST MINUS "-" PLUS "+" SCALE -m	LONG, STRING, FLOAT	
PIC S9(n)V9(m) DISPLAY SIGN LEADING	ZONED(n+m) SIGNED FIRST MINUS "-" PLUS "+" SCALE -m	LONG, STRING, FLOAT	
PIC S9(n)V9(m) DISPLAY SIGN TRAILING SEPARATE	ZONED(n+m) SIGNED LAST MINUS "-" PLUS "+" SEPARATE SCALE -m	LONG, STRING, FLOAT	
PIC 9(n)V9(m) DISPLAY	ZONED(n) UNSIGNED SCALE -m	LONG, STRING, FLOAT	
Note: [1] 8-byte BINARY with 10-18 digits is not supported.			

Grammar

This section describes the program mapper grammar. Base elements of the grammar are tokens, keywords in uppercase, constants and comments. By combining the base elements, you can define mapping elements: forward/backward mapping (FM/BM) and structure/interface definitions (IF, ST). The forward/backward mapping consists of rules (RL) which combine the structure and interface elements (IFE, STE). The following graphic is intended to illustrate the relationship between all these elements.

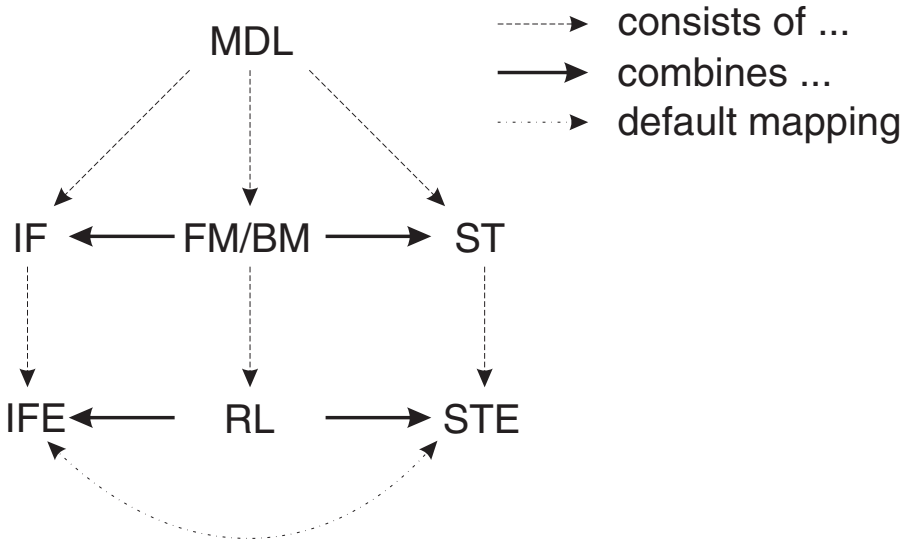


Figure 34. Relationship between mapping elements

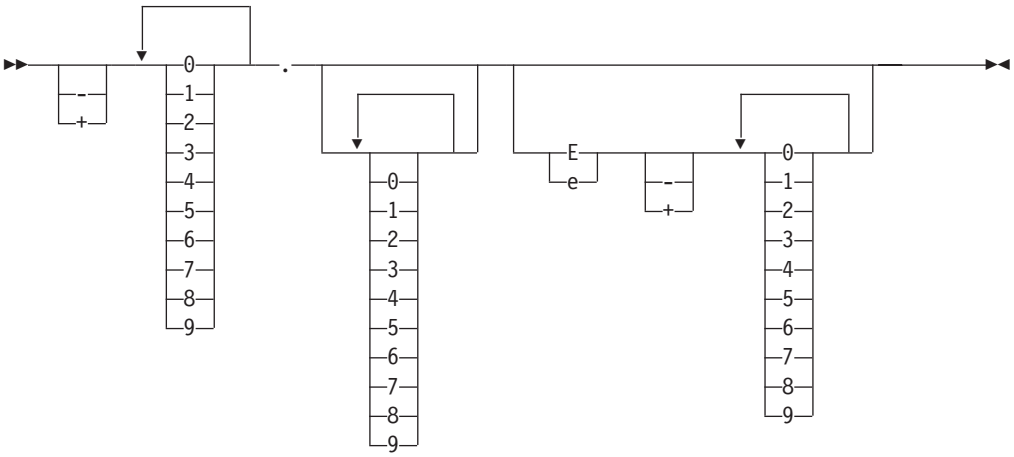
Grammar elements

Comments: Comments should be used to document the mapping definition. There are two types of comments: C style, for example /* 'comment' */ and C++ style, for example // 'comment' at the end of a line.

Comments starting with '/*' and ending with '*/' can be located anywhere between syntax tokens. Comments starting with // can be at the end of any line. Nesting of '/* ... */' is not allowed.

Tokens: Tokens are the base element of the grammar, and each token is therefore explained in detail. For each token, at least a syntax diagram and an example is given.

FLOAT_TOKEN:



Example: -7.42E-4, which equals -0.000742.

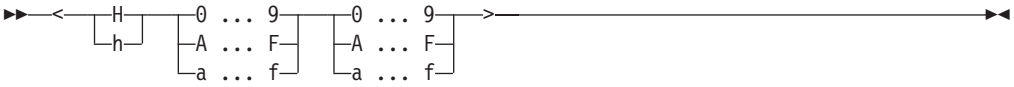
Note: In order to keep the diagrams simple, the possibility of choosing a single number from 0 to 9 will be displayed with

from now on.

Hex_digit:

Examples: 3, F.

Hex_token:



Example: <H4F>

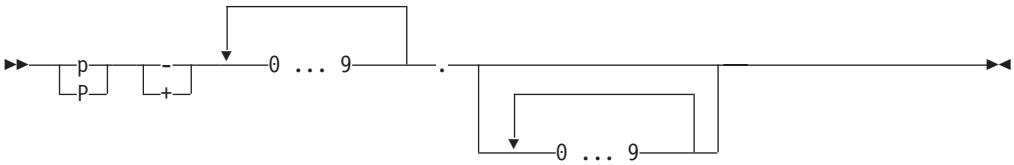
IDENTIFIER:

Examples: a_b4_h4, _Z97_bfsd

INT_TOKEN:

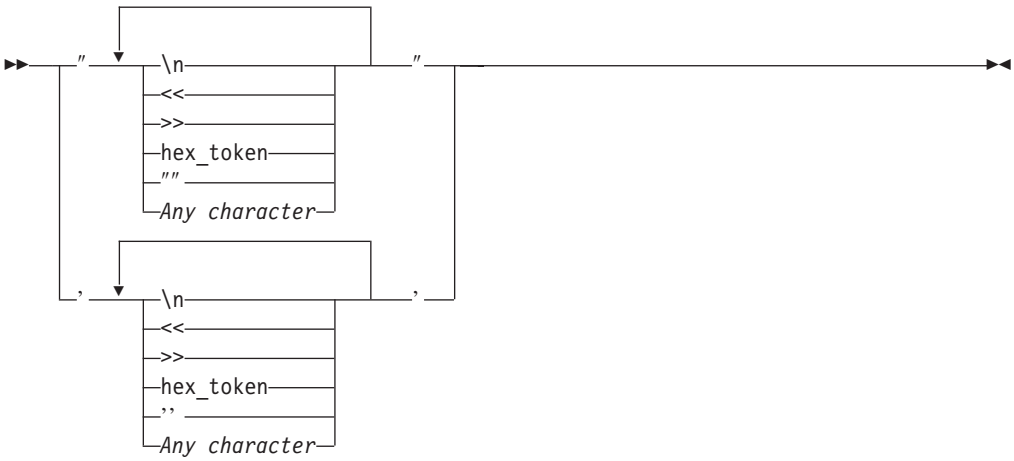
Examples: -89432, 412

PACKED_TOKEN:



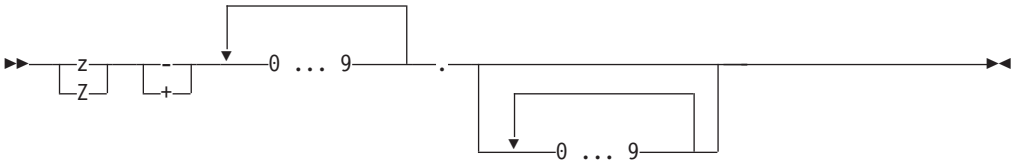
Examples: p+212.2 equals 212.2, p-142.8 equals -142.8

STRING_TOKEN:



Example: "AlbertEinstein", "xyz'a", 'xyz"a', 'other_example<h15><h12>'

ZONED_TOKEN:



Example: z+412.8

Program mapping

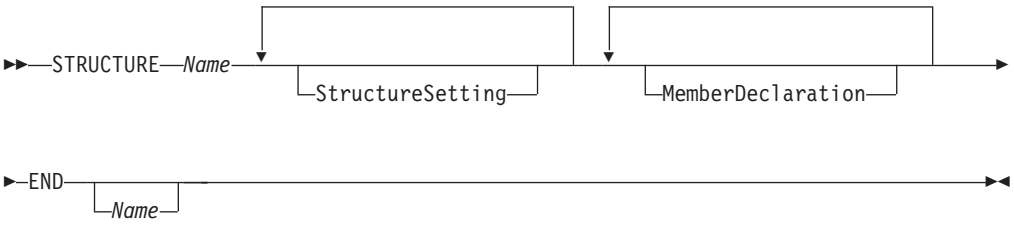
Keywords: Listed below are all available keywords. Do not name variables with these reserved keywords, because this would cause problems when mapping (for example, do not name a structure element 'STRUCTURE').

```
ARRAY           BACKWARDMAPPING BINARY          CHAR
CONSTANT        DESCRIPTION     DLL             DOCUMENTATION
END             FIRST           FLOAT           FORWARDMAPPING
FROM            IGNORE          INTEGER         INTERFACE
JUSTIFY         LAST            LEFT            LENGTH
LONG            MAP             MAPPING         MINUS
PACKED          PAD             PARMS           PLUS
RIGHT           SCALE           SEPARATE        SIGNED
STRING          STRUCTURE       TERMINATEDBY    TO
UNSIGNED        USERTYPE        ZONED
```

Note: All keywords must be in uppercase!

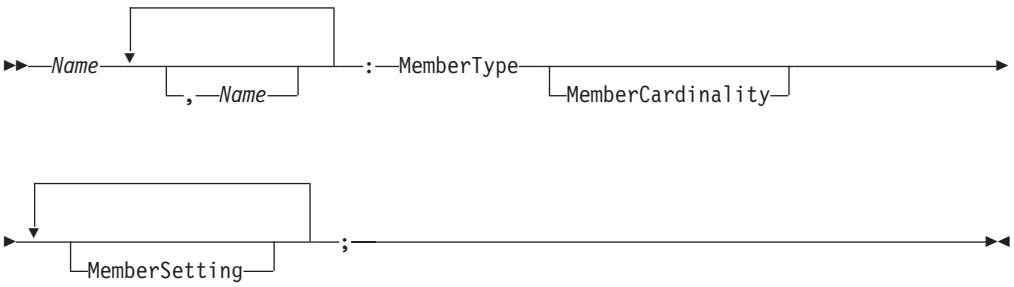
Structure definition:

Structure:



StructureSetting:

MemberDeclaration:



MemberType:

MemberCardinality:

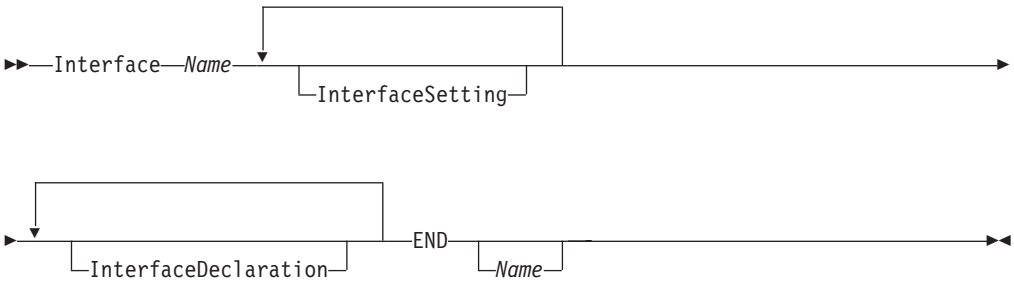
MemberSetting:

Note:

- Same syntax as structure definitions in FDL.

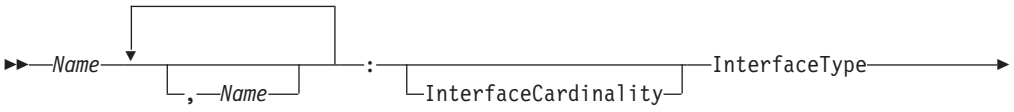
Interface definition:

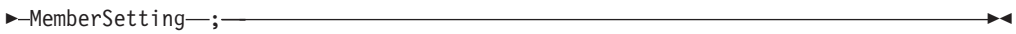
Interface:



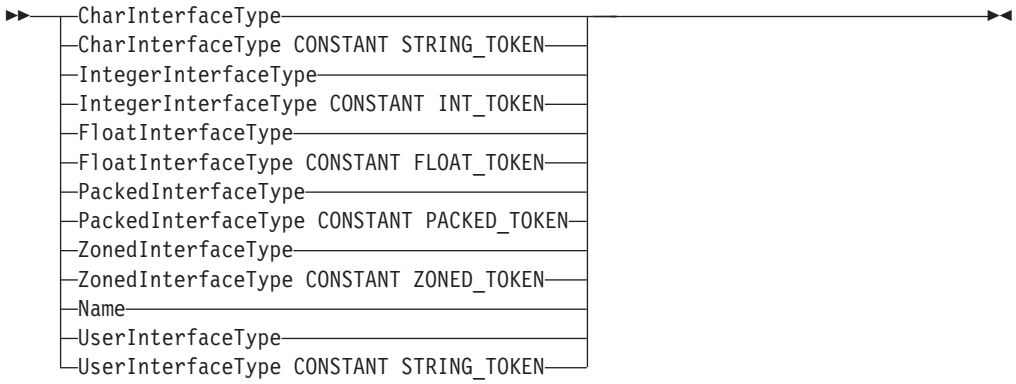
InterfaceSetting:

InterfaceDeclaration:

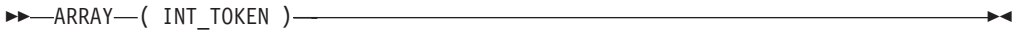




InterfaceType:



InterfaceCardinality:



Note:

- The sequence of elements is significant and defines the sequence of the elements in the data area used for forward and backward mapping. Each element has a fixed offset from the start of the data area. Make sure the interface elements have the size and type of the data the legacy application expects (See "Valid conversions between MQSeries Workflow container program mapping element types and program mapping interface types" on page 200 for size information).
- The sequence of member definitions in the structure is not relevant for the mapping. Mapping is done by name from interface elements to structure elements.

Interface types:

PackedInterfaceType:

PackedAttributeList:

Examples: p+212.2 equals 212.2, p-142.8 equals -142.8

Scale is used to define the decimal point of the packed number and the factor used by conversion. The packed number is multiplied by 10^{scale} in BACKWARDMAPPING and divided by 10^{scale} in FORWARDMAPPING. Of the plus character, minus character and unsigned character, only the lowermost 4 bits are used, which means that the value has to be <= x'0f'.

Example:
Packed number is 4711, scale is 0. Decimal number is 4711.
Packed number is 4711, scale is 2. Decimal number is 471100.
Packed number is 4711, scale is -2. Decimal number is 47.11.

Format: PACKED(5) SIGNED MINUS ″<h0d>″ PLUS ″<h0c>″ SCALE 1;

Byte0 Byte1 Byte2

DD DD DS

where D is a digit 0-9 and S is the positive or negative sign

Note:

- The size in bytes used by the packed number is (packed number size + 1) / 2, rounded up to the next integer.
- Packed numbers can create runtime conversion errors if the digits are > 9 or the sign does not match the sign defined for the interface element.

ZonedInterfaceType:

ZonedAttributeList:

Examples: z+471.1 equals 471.1, z-142.8 equals -142.8.

The size defines the number of significant digits used by the zoned number. Scale is used to define the decimal point of the zoned number and defines the factor used by conversion. The zoned number is multiplied by 10^{scale} in backward mapping and divided by 10^{scale} in forward mapping. FIRST and LAST define where the sign is located in the number. Of the plus character, minus character and unsigned character, only the lowermost 4 bits are used if the sign is not separate, which means the value has to be <= x'0f'. If the sign is separate, all 8 bits of the first character are used, which means the character has to be <= x'ff'. FIRST and LAST define the location of the sign (see examples below).

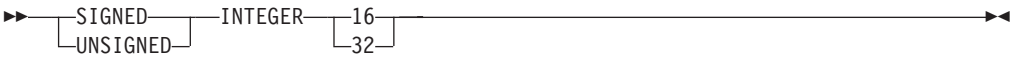
Example:
Zoned number is 4711, scale is 0. Decimal number is 4711.

Zoned number is 4711, scale is 2. Decimal number is 471100.
Zoned number is 4711, scale is -2. Decimal number is 47.11.
Format:
ZONED(3) SIGNED LAST LAST MINUS ″<h0d>″ PLUS ″<h0c>″ SCALE 2
Byte0 Byte1 Byte2
FD FD SD
where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits.
Format:
ZONED(3) SIGNED FIRST LAST MINUS ″<h0d>″ PLUS ″<h0c>″ SCALE 2
Byte0 Byte1 Byte2
SD FD FD
where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits.
Format:
ZONED(3) SIGNED LAST SEPARATE MINUS ″-″ PLUS ″+″ SCALE 2
Byte0 Byte1 Byte2 Byte3
FD FD FD XX
where D is a digit 0-9 and S is the positive, negative or unsigned sign and F are the zoned bits and XX is the sign (either x'4E' or x'60').

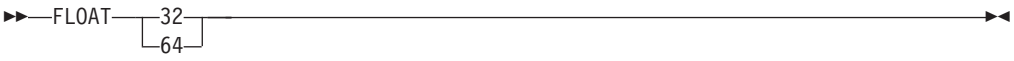
Note:

- The size in bytes used by the zoned number is the zoned number size. If the sign is separate, one additional byte is used.
- Zoned numbers can create runtime conversion errors if the digits are > 9 and the zone does not contain x'f' or the sign does not match the sign defined for the interface element.
- FIRST and LAST define where to append the sign.

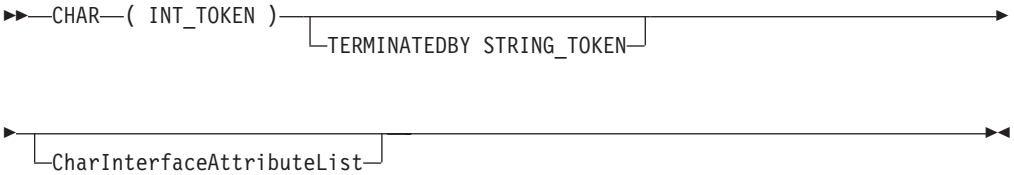
IntegerInterfaceType:



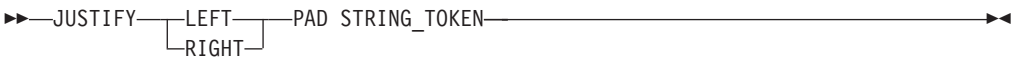
FloatInterfaceType:



CharacterInterfaceType:



CharInterfaceAttributeList:

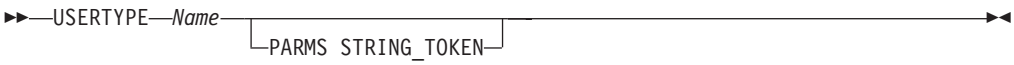


The termination character is inserted in forward mapping and stripped off in backward mapping. Padding, alignment and truncation occurs in forward mapping. The data is not modified in backward mapping. The character size in bytes must include the termination character and the number of bytes converted is one less than the specified character size.

Examples for justification:

Length	Content	Length	JUSTIFY(Left)	JUSTIFY(RIGHT)
3	'ABC'	4	'ABC '	' ABC'
3	'AB '	4	'AB '	' AB '
3	' BC'	4	' BC '	' BC'
4	'ABCD'	4	'ABCD'	'ABCD'
4	' BCD'	4	' BCD'	' BCD'
4	'ABC '	4	'ABC '	'ABC '
5	'ABCDE'	4	'ABCD'	'BCDE'
5	' BCDE'	4	' BCD'	'BCDE'
5	'ABCD '	4	'ABCD'	'BCD '

UserInterfaceType:

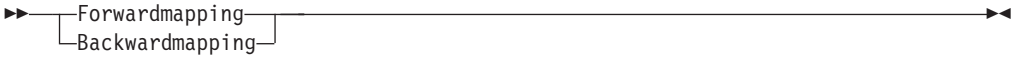


Note: STRING_TOKEN is passed to the usertype exit.

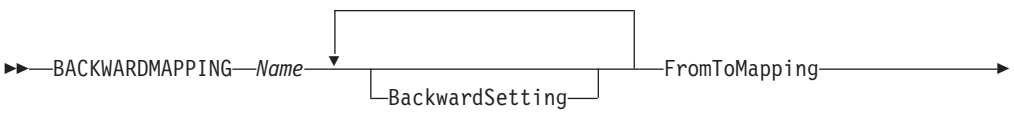
Mapping elements: This section illustrates the formal definition and grammar for the MDL. For each mapping element (structure, interface, forward/backward mapping etc.), there is a syntax diagram which explains how to use the elements correctly. Examples: “Example” on page 198ff and “MDL examples” on page 220ff.

MappingElement:

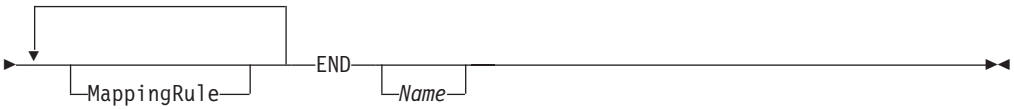
Mapping:



Backward mapping:



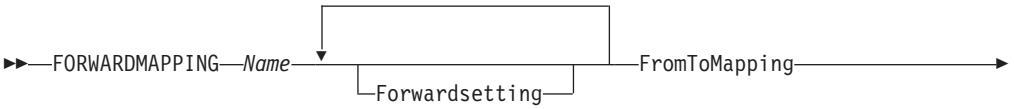
Program mapping

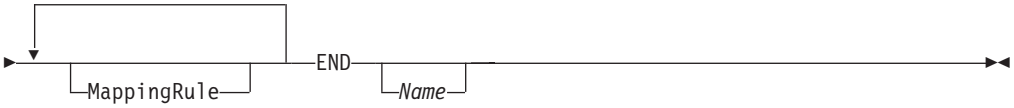


FromToMapping:

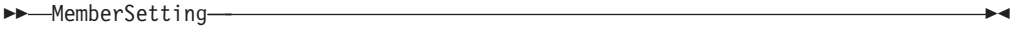
BackwardSetting:

Forward mapping:





ForwardSetting:

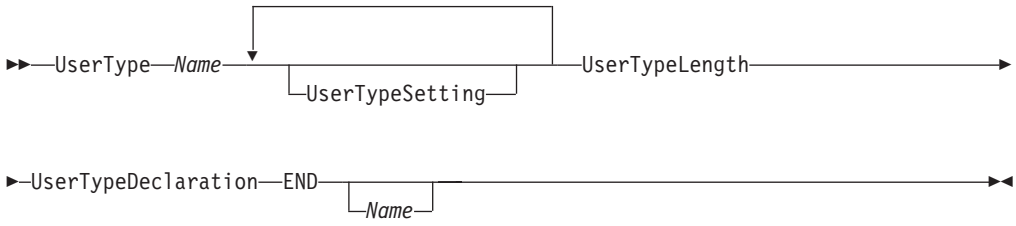


MappingRule:

Note: The first name must be the interface name in backward mapping and the structure element name in forward mapping. The second name must be the interface name in forward mapping and the structure element name in backward mapping (see "Mapping algorithm" on page 194).

Usertype definition:

UserType:



UserTypeLength:

Note: The usertype length defines the size of the usertype in bytes.

UserTypeSetting:

UserTypeDeclaration:

Note: The first string_token in the usertype declaration defines the DLL name, and the second defines the exit entry name. It is therefore possible to use one DLL for multiple usertypes.

Sample MDL for C/C++ and COBOL: In this example, each definition is shown in detail and the variables used have the same name (interface and structure). Therefore the forward and backward mapping definitions are as simple as possible and an explicit mapping as in the previous example is unnecessary.

```
/*
   --- Structure definition ---
*/
STRUCTURE AccountRepStructureBackw
   LastName:  STRING;
   FirstName:  STRING;
   Zip: LONG;
   Salary: FLOAT;
   Tax: FLOAT;
   Customers: CustomerStructure [3];
END AccountRepStructureBackw
STRUCTURE AccountRepStructureForw
   LastName:  STRING;
   FirstName:  STRING;
   Zip: LONG;
   Salary: FLOAT;
   Tax: FLOAT;
END AccountRepStructureForw
/* In this example the CustomerStructure contains 3 elements
   (last name, first name and telephone number which are defined as
```

```
   string) */
STRUCTURE CustomerStructure
   LastName: STRING;
   FirstName: STRING;
   PhoneNumber: STRING;
END CustomerStructure
/*
   --- Interface definition for COBOL ---
*/
INTERFACE AccountRepInterfaceForCOBOL
        LastName: CHAR(50) JUSTIFY LEFT PAD ' ';
        FirstName: CHAR(50) JUSTIFY LEFT PAD ' ';
        Zip: UNSIGNED INTEGER 16;
        Salary: PACKED(8) UNSIGNED '<h0c>' SCALE -2;
        Tax: PACKED(2) UNSIGNED '<h0c>' SCALE -2;
        CustomersOpt: ARRAY(3) CustomerInterfaceForCOBOL;
END AccountRepInterfaceForCOBOL

INTERFACE CustomerInterfaceForCOBOL
        LastName: CHAR(50) JUSTIFY LEFT PAD ' ';
        FirstName: CHAR(50) JUSTIFY LEFT PAD ' ';
        PhoneNumber: CHAR(10) JUSTIFY LEFT PAD ' ';
END CustomerInterfaceForCOBOL
/*
   --- Interface definition for C++ ---
*/
INTERFACE AccountRepInterfaceForCpp
           LastName:  CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
           FirstName: CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
           Zip:       UNSIGNED INTEGER 16;
           Salary:    FLOAT 32;
           Tax:       FLOAT 32;
           Customers: ARRAY(3) CustomerInterfaceForCpp;
END AccountRepInterfaceForCpp
INTERFACE CustomerInterfaceForCpp
           LastName:    CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
           FirstName:   CHAR(50) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
           PhoneNumber: CHAR(10) TERMINATEDBY "<H00>" JUSTIFY LEFT PAD " ";
END CustomerInterfaceForCpp
/*
  -- Forward/backward mapping definition for COBOL ---
*/
FORWARDMAPPING ForwardSampleForCOBOL
    FROM AccountRepStructureForw TO AccountRepInterfaceForCOBOL
END ForwardSampleForCOBOL
BACKWARDMAPPING BackwardSampleForCOBOL
    FROM AccountRepInterfaceForCOBOL TO AccountRepStructureBackw
END BackwardSampleForCOBOL
/*
 --- Forward/backward mapping definition for C++ ---
*/
FORWARDMAPPING ForwardSampleForCpp
    FROM AccountRepStructureForw TO AccountRepInterfaceForCpp
END ForwardSampleForCpp
BACKWARDMAPPING BackwardSampleForCpp
    FROM AccountRepInterfaceForCpp TO AccountRepStructureBackw
END BackwardSampleForCpp
```

Note: This sample is distributed as FMCEMDL in SFMCDATA.

Usertype

A usertype allows converting MQSeries Workflow program mapping structure definition elements if the available interface types do not fulfill the required conversion. The program mapper will call a user exit each time a conversion for a usertype is required. It is possible to pass up to 256 characters to the user exit,

which must be defined where the interface element is mapped to the usertype. This allows using the same usertype for different conversions and controlling the functionality of the exit via passed parameters. In addition, it is possible to define parameters at the same time the forward and backward mapping formats are defined at buildtime. These are designated as 'forward mapping parameters' and 'backward mapping parameters', and the user exit has access to these parameters. Usertypes must have a fixed length.

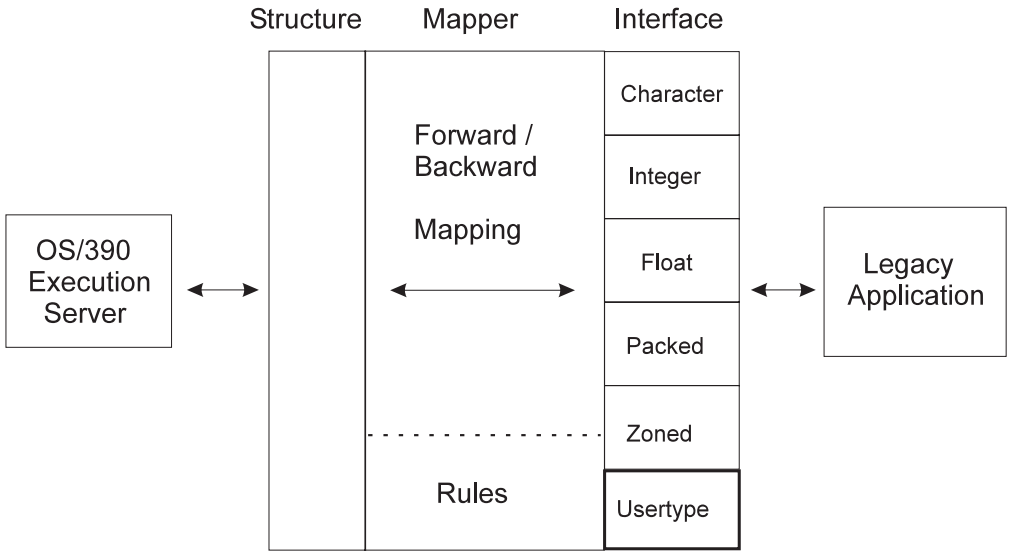


Figure 35. Usertype exit

Exit interface

The exit interface and data structures passed are defined in file fmcxmeut.h.

Note: The exit's entry point must have C linkage.

The following parameters are passed:

1. Direction of mapping required by PES

   Use defined constants to check whether forward or backward mapping should be done.

   ```
   #define FMC_PROGRAMMAPPING_USERTYPE_BACKWARDMAPPING 0
   #define FMC_PROGRAMMAPPING_USERTYPE_FORWARDMAPPING 1
   ```

2. InterfaceDescriptor

   Allows accessing the interface element data via a pointer to the data buffer. In addition the length of the usertype in bytes is passed.

   ```
   typedef struct {
           char* elementData;                  // pointer to raw data
           unsigned long elementDataLength;    // length of usertype in bytes
   } FmcProgrammMappingInterfaceDescriptor;
   ```

   Warning: Make sure that the data written into the data buffer during forward mapping is not longer than the size of the usertype in bytes. Otherwise the results can be unpredictable.

3. StructureDescriptor

Allows accessing the structure element. The element name and length are passed in addition to a handle to the MQSeries Workflow container. The element can be accessed with the MQSeries Workflow Container API by passing the element name.

```
typedef struct {
     const char* elementName;
        //container element name (zero terminated)
     unsigned long elementNameLength;
        // container element name length in bytes
     FmcjContainerHandle containerHandle;
        // container handle
} FmcProgrammMappingStructureDescriptor;
```

The element name contains the qualified element name and can be used to get or set the container element with the container API. The name is zero terminated.

4. BuildTimeParameter, buildTimeParameterLength

   At buildtime, it is possible to insert forward and backward mapping parameters. The length in bytes is passed via buildTimeParameterLength. The name is zero terminated.

5. InterfaceParameter, interfaceParameterLength

   It is possible to insert a parameter in the interface where the usertype is used. The length in bytes is passed via interfaceParameterLength. The name is zero terminated.

   Example:

   ```
   INTERFACE SampleUsertypeInterface
          SampleElement: USERTYPE SampleUsertype PARMS "$";
   END
   ```

6. Return value

   A nonzero return value signals an error to the program mapper and the return code is set for the activity implementation.

   For a sample, see sample usertype exit FMCHSMUT in SFMCDATA.

Creation of DLL

The usertype exit must be available at PES runtime. Any entry name and DLL name can be used, but these names must be identical to the names used in the usertype definition (See below). Sample JCL FMCHJMUT in SFMCDATA builds the sample usertype DLL. It is possible to have multiple usertype exits in one DLL if they use different function names.

Usertype definition

The usertype definition defines the name of a usertype and the length of the usertype in bytes. In addition the DLL name and exit function name have to be specified.

```
USERTYPE SampleUsertype LENGTH(4)
            DLL "SAMPUTY","SampleUsertypeExit"
END
```

See FMCHEMUT in SMFCDATA for a sample usertype definition.

Size of program mapping interface definition elements

The interface definition must match exactly the layout of the data the legacy application expects. If there is a mismatch of even one byte, the results are unpredictable!

The following list summarizes which number of bytes are used by the interface definition element types.

Table 15. Interface element size

Type	Length	Example
Char	Size equals length in bytes.	Char(2) has a length of 2 bytes.
Integer	2 bytes for INTEGER 16 and 4 bytes for INTEGER 32.	-
Float	4 bytes for FLOAT 32 and 8 bytes for FLOAT 64.	-
Packed	Size used to define the (packed number + 1) divided by 2 and rounded up	Packed(2) equals 2 bytes, Packed(3) equals 2 bytes and Packed(4) equals 3 bytes.
Zoned	Size used to define the zoned number if a separate sign is not defined. Otherwise, it is one larger than the size used to define the zoned number.	Zoned(4) equals 4 bytes as Zoned(4) SEPARATE equals 5 bytes.
UserType	Size of usertype.	USERTYPE SampleUsertype LENGTH(4) equals 4 bytes.

Note: If there is any alignment done by the compiler used to compile the legacy application, this alignment also must be done in the interface definition.

Example: A C structure defined as follows:

```
struct S {
          int x;
          char y;
          int z;
         };
```

might be aligned on 4-byte boundary so that

```
x x x x y       z z  z  z
0 1 2 3 4 5 6 7 8 9 10 11 Byte
```

and therefore needs following interface definition

```
INTERFACE i
          x:   SIGNED INTEGER 32;
          y:   CHAR(1) JUSTIFY LEFT PAD " ";
          pad: CHAR(3) JUSTIFY LEFT PAD "<h00>";
          z:   SIGNED INTEGER 32;
END
```

Activation of program mapping definitions

The activation of a program mapping definition is described in detail in "Administering program mapping" in *MQSeries Workflow for z/OS: Customization and Administration*. The following contains a short summary about how to activate a program mapping definition:

1. Copy sample job to FMCHJMPR.
2. Create an MDL.
3. Update control statements for the input utility.
4. Run and compile MDL and insert MDL into mapping database.
5. If existing MDL elements were modified, restart the PES to activate the modifications. For new elements, no PES restart is needed.

Troubleshooting

In order to provide as much help to you as possible, this section lists usual problems and typical solutions. For a detailed list of error messages refer to *MQSeries Workflow for z/OS: Messages and Codes*.

Common errors

Element data mapped is incorrect: Most commonly, this is due to a mismatch between the legacy application data layout and the interface definition.

There is no way for the program mapper to check whether the interface maps correctly to the data format and layout the legacy application expects. Each interface should be carefully created and double checked. Runtime conversion errors (for packed and zoned interface types), are usually caused by this. In addition, reflect alignment on the legacy side in the interface (see "Size of program mapping interface definition elements" on page 216 for more details). If the size of one interface element is incorrect (for example integer 16 instead of integer 32) all the following data will be incorrect.

Elements not mapped: Either the element names are different and no mapping rule was specified or a mapping rule uses the wrong element names.

Note: If a mapping rule is used in both directions, the mapping rule arguments have to be switched.

Modified mapping definition is not activated: Mapping definitions are reloaded whenever the PES is restarted. It is not sufficient to import the definition into the program mapping database. New definitions will be used without a PES restart.

Additional mapping examples

Application examples

CICS C++ Application: This C++ application under CICS displays the data, creates new customers and increases the salary by 8 percent. It corresponds to the forward/backward mapping example in "Forward/backward mapping definition" on page 192.

```
#pragma XOPTS(SP)
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
/* --- CustomerStructure --- */
#pragma pack(1)
struct CustomerStructure {
   char LastName[50];
   char FirstName[50];
   char PhoneNumber[10];
};
/* --- AccountRepStructure --- */
struct AccountRepStructure {
   char LastName[50];
   char FirstName[50];
   short Zip;
   float Salary;
   float Tax;
   struct CustomerStructure Customers[3];
};
#pragma pack(1)
int main()
```

```
{
   struct AccountRepStructure *commarea;
   EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
   if (dfheiptr->eibcalen <= 0) {
      cout << "??? Empty commarea ???" << endl;
      EXEC CICS RETURN;
   }
// Display all data
   cout << "LastName:   " << commarea->LastName << endl;
   cout << "FirstName:  " << commarea->FirstName << endl;
   cout << "Zip:        " << commarea->Zip << endl;
   cout << "Salary:     " << commarea->Salary << endl;
   cout << "Tax:        " << commarea->Tax << endl;
// Create customers
   strcpy(commarea->Customers[0].LastName,"EINSTEIN");
   strcpy(commarea->Customers[0].FirstName,"ALBERT");
   strcpy(commarea->Customers[0].PhoneNumber,"3048");
   strcpy(commarea->Customers[1].LastName,"NEWTON");
   strcpy(commarea->Customers[1].FirstName,"ISAAC");
   strcpy(commarea->Customers[1].PhoneNumber,"4041");
   strcpy(commarea->Customers[2].LastName,"HAWKING");
   strcpy(commarea->Customers[2].FirstName,"STEVEN");
   strcpy(commarea->Customers[2].PhoneNumber,"5154");
   for (int i=0; i<3; i++) {
      cout << "Customer LastName : "
         << commarea->Customers[i].LastName << endl;
      cout << "Customer FirstName : "
         << commarea->Customers[i].FirstName << endl;
      cout << "Customer PhoneNumber : "
         << commarea->Customers[i].PhoneNumber << endl;
   }
// Increase salary by 8%
   commarea->Salary *= 1.08;
   cout << "New Salary: " << commarea->Salary << endl;
   EXEC CICS RETURN;
}
```

CICS COBOL Application: This COBOL application under CICS does the same thing as "CICS C++ Application" on page 218 (displays the data, creates new customers and increases the salary by 8 percent). It corresponds to the forward/backward mapping example in "Forward/backward mapping definition" on page 192.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. "SAMPCBL".
       DATA DIVISION.
         WORKING-STORAGE SECTION.
         01 PRINT-SALARY PIC Z(5)9.9(2).
         01 PRINT-TAX PIC Z9.99.
         LINKAGE SECTION.
         01  DFHCOMMAREA.
      *
      *     AccountRepStructure
      *
              02  LASTNAME PIC X(50).
              02  FIRSTNAME PIC X(50).
              02  ZIP  PIC 9999 COMP-4.
              02  SALARY PIC 9(6)V9(2) COMP-3.
              02  TAX PIC V99 COMP-3.
      *
      *     CustomerStructure
      *
              02  CUSTOMERS OCCURS 3 TIMES
                     INDEXED BY CUSTOMER-INDEX.
                 03 LAST-NAME PIC X(50).
                 03 FIRST-NAME PIC X(50).
                 03 PHONE-NUMBER PIC X(10).
```

```
 PROCEDURE DIVISION.
      IF EIBCALEN <= 0
         DISPLAY "??? EMPTY COMMMAREA ???"
         EXEC CICS RETURN
         END-EXEC
      END-IF
*
*     Display all data
*
      DISPLAY "Lastname:  " LASTNAME
      DISPLAY "Firstname: " FIRSTNAME
      DISPLAY "Zip:       " ZIP
      MOVE TAX TO PRINT-TAX
      MOVE SALARY TO PRINT-SALARY
      DISPLAY "Salary:    " PRINT-SALARY
      DISPLAY "Tax:       " PRINT-TAX
*
*     Create some customers
*
      MOVE "EINSTEIN" TO LAST-NAME(1)
      MOVE "ALBERT" TO FIRST-NAME(1)
      MOVE "3048" TO PHONE-NUMBER(1)
      MOVE "NEWTON" TO LAST-NAME(2)
      MOVE "ISAAC" TO FIRST-NAME(2)
      MOVE "4041" TO PHONE-NUMBER(2)
      MOVE "HAWKING" TO LAST-NAME(3)
      MOVE "STEPHEN" TO FIRST-NAME(3)
      MOVE "5154" TO PHONE-NUMBER(3)
      PERFORM
         VARYING CUSTOMER-INDEX FROM 1 BY 1
            UNTIL CUSTOMER-INDEX > 3
         DISPLAY "Customer LastName : "
            LAST-NAME(CUSTOMER-INDEX)
         DISPLAY "Customer FirstName: "
            FIRST-NAME(CUSTOMER-INDEX)
         DISPLAY "Customer PhoneNumber: "
            PHONE-NUMBER(CUSTOMER-INDEX)
      END-PERFORM
*     Increase salary by 8%
      COMPUTE SALARY = SALARY * 1.08
      MOVE SALARY TO PRINT-SALARY
      DISPLAY "New Salary: " PRINT-SALARY
      EXEC CICS RETURN
      END-EXEC
      GOBACK.
```

MDL examples

This section illustrates some examples of how to use the mapper. There are examples coded in C, COBOL and for simple and complex data structures.

Simple data structure with default name mapping: In this example the mapping is defined for a simple data structure and the mapping used is the default one (which means that each element of a container is mapped to the element with the **same name** in the other container).

```
STRUCTURE SimpleDataStructure
        element1: STRING;
        element2: STRING;
        element3: LONG;
        element4: FLOAT;
        element5: BINARY;
        element6: BINARY;
        element7: LONG(20);
        END SimpleDataStructure
INTERFACE SimpleDataInterface
        DESCRIPTION 'This is an example of a simple interface mapping'
```

```
        element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
        element2: CHAR(20) JUSTIFY LEFT PAD "<h00>";
        element3: SIGNED INTEGER 16;
        element4: FLOAT IBM 32;
        element5: CHAR(500);
        element6: CHAR(200);
        element7: ARRAY (10) SIGNED INTEGER 8;
        END SimpleDataInterface
BACKWARDMAPPING SimpleMapping
        FROM SimpleDataInterface
        TO SimpleDataStructure
        END SimpleMapping
FORWARDMAPPING SimpleMapping
        FROM SimpleDataStructure
        TO SimpleDataInterface
        END SimpleMapping
```

Complex data structure with default name mapping: In this example, the mapping is defined for a complex data structure and the mapping used is the default one (which means that each element of a container is mapped to the element with the **same name** in the other container).

```
STRUCTURE ComplexDataStructure1
        element1: STRING (10);
        element2: FLOAT;
        END ComplexDataStructure1
STRUCTURE ComplexDataStructure2
        element1: STRING (20);
        element2: ComplexDataStructure (5);
        END ComplexDataStructure2
INTERFACE ComplexDataInterface1
        element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
        element2: FLOAT IBM 32";
        END ComplexDataInterface1
INTERFACE ComplexDataInterface2
        element1: CHAR(20) JUSTIFY RIGHT PAD ' ';
        element2: ARRAY(5) ComplexDataInterface1;
        END ComplexDataInterface2
BACKWARDMAPPING ComplexMapping
        FROM ComplexDataInterface2
        TO ComplexDataStructure2
            /*
             * Element1 and element2 are mapped implicitly
             */
        END ComplexMapping
FORWARDMAPPING ComplexMapping
        FROM ComplexDataStructure2
        TO ComplexDataInterface2
            /*
             * element1 and element2 are mapped implicitly
             */
        END ComplexMapping
```

Complex data structure with non-default name mapping: In this example the mapping is defined for a complex data structure and the mapping used is not the default one (which means that the structure elements of ComplexDataStructure1 do not have identical names in the interface ComplexStructure1 and are mapped explicitly).

```
STRUCTURE ComplexDataStructure1
        strs: STRING (10);
        flts: FLOAT;
        END ComplexDataStructure1
INTERFACE ComplexDataInterface1
        stri: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
        flti: FLOAT IBM 32;
```

```
        END ComplexDataInterface1
BACKWARDMAPPING ComplexMapping
        FROM ComplexDataInterface1
        TO ComplexDataStructure1
           MAP stri TO strs;
           MAP flti TO flts;
        END ComplexMapping
FORWARDMAPPING ComplexMapping
        FROM ComplexDataStructure1
        TO ComplexDataInterface1
           MAP strs TO stri;
           MAP flts TO flti;
        END ComplexMapping
```

Complex data structure with non-default name mapping with arrays and structures: In this example, the mapping is defined for a complex data structure and the mapping used is not the default one (which means that the elements (in this case arrays) of structure ComplexDataStructure1 do not have identical names in the Interface ComplexStructure1 and are mapped explicitly).

```
STRUCTURE ComplexDataStructure1
        element1: STRING (10);
        element2: FLOAT;
        END ComplexDataStructure1
STRUCTURE ComplexDataStructure2
        element1: STRING (20);
        element2: ComplexDataStructure1 (5);
        specials: FLOAT;
        END ComplexDataStructure2
STRUCTURE ComplexDataStructure3
        element1: STRING (5);
        element2: ComplexDataStructure2 (4);
        element3: ComplexDataStructure1 (4);
        END ComplexDataStructure3
INTERFACE ComplexDataInterface1
        element1: CHAR(10) TERMINATEDBY "<h00>" JUSTIFY LEFT PAD ' ';
        element2: FLOAT IBM 32;
        END ComplexDataInterface1
INTERFACE ComplexDataInterface2
        element1: CHAR(20) TERMINATEDBY "<h00>" JUSTIFY RIGHT PAD '*';
        element2: ARRAY (5) ComplexDataInterface1;
        speciali: FLOAT IBM 32
        END ComplexDataInterface2
INTERFACE ComplexDataInterface3
        element1: CHAR(5) JUSTIFY RIGHT PAD '*';
        element2: ARRAY (4) ComplexDataInterface1;
        elementx: ARRAY (4) ComplexDataInterface2;
        END ComplexDataInterface3
BACKWARDMAPPING ComplexMapping
        FROM ComplexDataInterface3
        TO ComplexDataStructure3
         /* Interface element elementx is explicitly mapped to structure
            element element2. All structure and interface elements of this
            structure are mapped per default; interface element element2 is
            also explicitly mapped with all its subelements. */
           MAP 'elementx' to 'element2';
           MAP 'element2' to 'element3';
           /* element speciali is not mapped per default and an explicit rule
              for this element is required */
           MAP 'elementx.speciali' to 'element2.specials';
           /* Per default mapping element1 with all subelements is mapped */
        END ComplexMapping
FORWARDMAPPING ComplexMapping
        FROM ComplexDataStructure1
        TO ComplexDataInterface1
         /* Structure element element2 is explicitly mapped to interface
            element elementx. All structure and interface elements of this
```

```
            structure are mapped per default; structure element element3 is
            also explicitly mapped with all its subelements. */
           MAP 'element2' to 'elementx';
           MAP 'element3' to 'element2';
           /* element specials is not mapped per default and an explicit rule
              for this element is required */
           MAP 'element2.specials' to 'elementx.speciali';
           /* Per default mapping element1 with all subelements is mapped */
        END ComplexMapping
```

Simple data structure with all interface types with CONSTANTS and usertypes: In this example the mapping is defined for a simple data structure and the mapping used is not the default one (which means that the elements of structure ComplexDataStructure1 do not have identical names in the interface ComplexStructure1 and are mapped explicitly). Additionally, there is a usertype defined, which converts a 4 byte integer into a Workflow string, separates every three digits by a comma and prefixes the string with a currency symbol, for example $1,234,567.

```
/* A usertype which converts a 4 byte integer into a Workflow string, separates
 * every three digits by a comma and prefixes the string with a currency
 * symbol, for example $1,234,567 */
USERTYPE user1 LENGTH(4)
               DLL "dlluser","user2Inbound"
         END user1
STRUCTURE SimpleDataStructure
        element1: LONG;
        element2: STRING;
        element3: LONG;
        element7: LONG(20);
        element8: STRING;
        END SimpleDataStructure
INTERFACE SimpleDataInterface
        DESCRIPTION 'This is an example of a simple interface mapping'
        /* The following integer constant is inserted in forward mapping
         * and removed in backward mapping */
        insert:   SIGNED INTEGER 16
                  CONSTANT 4711
        element3: SIGNED INTEGER 16;
        element7: ARRAY (100) SIGNED INTEGER 8;
        element1: USERTYPE user1 PARMS 'DM'; /* three digits */
        element8: USERTYPE user1 PARMS '$'; /* for example $12,345 */
        element9: CHAR(5) CONSTANT "This is a string constant with some
                        hex chars <h47><h11>" JUSTIFY RIGHT PAD '*'
        element10: PACKED (10) CONSTANT p47.11
                        SIGNED FIRST MINUS "<h0d>" PLUS "<h0c>" UNSIGNED "<h0c>"
                        SCALE 5
        element11: ZONED (10) CONSTANT z47.11
                        SIGNED FIRST MINUS "<h0d>" PLUS "<h0c>" SCALE 5
        element12: FLOAT IBM 8
                   CONSTANT +47E11
        END SimpleDataInterface
BACKWARDMAPPING SimpleMapping
        FROM SimpleDataInterface
        TO SimpleDataStructure
        END SimpleMapping
FORWARDMAPPING SimpleMapping
        FROM SimpleDataStructure
        TO SimpleDataInterface
        END SimpleMapping
```

Program execution server exits

Introduction

For extensibility of MQSeries Workflow for OS/390, the PES uses exits in the following areas:

- Application invocation
- Legacy application program mapping
- Event notification

Whenever new exits are needed, these exits can be used instead of, or in parallel with, the IBM supplied exits.

The exits have a defined interface, to which every user written exit must conform.

Each exit type must provide an **Init()** function, which is called from the PES when the exit is needed the first time. Later, the exit-specific functions are called (see the specific exit descriptions for more details). A shutdown request for the PES triggers a call to the **Deinit()** function of the exits.

Usually the **Init()** function does all the initialization needed for the exit. If information is needed further on in subsequent calls, a handle can be filled in the initialization call, which is then passed to all subsequent functions (for example, DB handles, connection information, states, etc.). **Deinit()**, which is called last, normally deallocates and frees all resources allocated during Init().

The exit DLL is loaded by the PES when the exit is needed the first time and unloaded when the PES terminates.

The main difference between the exit types is the following:

- **Mapping** exits do a data conversion between MQSeries Workflow for OS/390 containers and data acceptable by legacy applications.
- **Invocation** exits invoke applications on the application side.
- **Notification** exits react to specific events reported by the PES in conjunction with program invocation requests.

Notes:

1. Whenever you modify an exit, you must shutdown and reboot the PES in order to make your changes effective.
2. All PES exits must be reentrant.

Return codes and error messages

All exits use a return code to signal availability of error information from the exit functions to the PES. If the return code is not OK, 4 parameters used in each function contain more detailed error information. The possibility exists to classify errors as recoverable or non-recoverable. The PES no longer distinguishes between these two types of errors. Both are handled the same way: by passing the error to the execution server for a Workflow request or in the case of an Execute Program request to the Workflow client.

Parameters:

char * errorIdBuffer

- 4-character buffer provided by the PES and used to pass an error number. Must be set accordingly if the ID is shorter than 4 characters.

- Input/output parameter

long * errorIdBufferLength

- A pointer to the length of the message number in errorIdBuffer, which must be set by the exit. On input, the maximum available number of characters is passed. On return, valid lengths are between 0 and the maximum value passed in errorIdBufferLength.
- Input/output parameter

char * errorDescriptionBuffer

- Character buffer provided by the PES, which is 512 characters long and is used for an error message. Has to be set accordingly if the message is shorter than 512 characters.
- Input/output parameter

long * errorDescriptionBufferLength

- A pointer to the length of the description in errorDescriptionBuffer, which must be set by the exit. On input, the maximum available number of characters is passed. On return, valid lengths are between 0 and the maximum value passed in errorDescriptionBufferLength.
- Input/output parameter

The error ID can consist of up to 4 digits. The PES prefixes it with a character identifying the exit type (I for invocation, M for mapping, N for notification). In addition the errorDescription is prefixed by the PES with the DLL name of the exit so that each exit can use the message numbers and the DLL name is the identifier of the exit.

Return codes:

FMC_EXIT_OK
The function was successful.

FMC_EXIT_RECOVERABLE_ERROR
The function was unsuccessful but recoverable. The PES will return message FMC32204 (see *MQSeries Workflow for z/OS: Messages and Codes*) with the passed error information. The PES continues processing. errorIdBuffer, errorIdBufferLength, errorDescriptionBuffer, and errorDescriptionBufferLength must be set accordingly.

FMC_EXIT_NONRECOVERABLE_ERROR
The function was unsuccessful and unrecoverable. The PES handles this error similar to recoverable errors.

Interfaces for all exits

Note: All interfaces must have C linkage!

Init()

Header files: FMCXMIF.H (program mapping exit), FMCXIEP.H (invocation exit), FMCXNEIF.H (notification exit).

Function:

- Initialize exit.
- It is called once when the exit is used the first time.

Interface:

```
long Init
   ( void **  exitHandle,
     void *   initializationParameter,
     long     initializationParameterLength,
     char *   errorIdBuffer,
     long *   errorIdBufferLength,
     char *   errorDescriptionBuffer,
     long *   errorDescriptionBufferLength)
```

Parameters:

void ** exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called at a later time. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

void * initializationParameter

- Parameters defined in the PES directory entry for the exit, which can be used to customize the exit initialization. The parameter is terminated by zero.
- Input parameter

long initializationParameterLength

- Length of the initializationParameter in bytes.
- Input parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

Return codes: See "Return codes and error messages" on page 224 for the return codes.

Deinit()

Header files: FMCXMIF.H (program mapping exit), FMCXIEP.H (invocation exit), FMCXNEIF.H (notification exit).

Function:

- Deinitialize exit.
- It is called once when the PES terminates.

Interface:

```
long Deinit
 ( void **  exitHandle,
   char *   errorIdBuffer,
   long *   errorIdBufferLength,
   char *   errorDescriptionBuffer,
   long *   errorDescriptionBufferLength)
```

Parameters:

void ** exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called later on. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

Return codes: See "Return codes and error messages" on page 224 for the return codes.

Special considerations for exit programming

Use of RRS commit and rollback

PES exits are called in the context of an MQ Workflow transaction by the program execution server (PES). Changes made by the exits can be made persistent (RRS commit) or rescinded (RRS rollback). However, these actions also terminate the transaction itself. Since PES exits can be called at any point during the transaction, RRS commit or rollback would terminate the transaction prematurely and lead to unpredictable results. In particular, a "safe application" would no longer be "safe".

Consequently, RRS commit and rollback calls **srrcmit** and **srrback** must never be issued in a PES exit. Furthermore, any functions performing these calls must *not* be used in PES exits; otherwise, units of work are terminated by these calls before the PES can complete them.

Buffer allocation

Both invocation and mapping exits use buffers to receive data from the PES and pass data to the PES. In some situations, the buffers are owned by the PES and must not be deleted by the exit. In other situations, the buffers must be created and deleted by the exit. For specific requirements, see the descriptions of the individual buffer parameters.

Program mapping exit

The IBM-supplied program mapping exit can be used to convert and translate data between legacy applications and MQSeries Workflow for OS/390. Any other mapping exit that conforms to the mapping exit interface can be used in parallel with, or in place of, the IBM supplied mapping exit.

The program mapping exit must be available in a DLL which is loaded by the PES and used to translate the MQSeries Workflow for OS/390 container data into data accepted by a legacy application.

A mapping exit must follow the general rules for PES exits as described in "Interfaces for all exits" on page 225. The DLL must have two exit functions: **Init()** and **Deinit()** and one exit specific function **Translate()**. The latter is used to do the actual conversion and translation for forward and backward mappings every time a legacy application is called. The raw buffer for the legacy application is available, and the container can be accessed via MQSeries Workflow for OS/390 Container API calls.

In forward mapping calls, the program mapping exit must extract the data from the container and translate the data into a raw buffer which is passed to the legacy application. In backward mapping calls the program mapping exit must translate the raw buffer and assign the data to the container.

Additional interfaces specific to the program mapping exit

The program mapping exit must define the **Translate()** function as follows.

Translate()

Header files: FMCXMIF.H (program mapping exit).

Function:

- **Translate()** is called each time by the PES in the direction FMC_PROGRAMMAPPING_BACKWARDMAPPING when backward mapping is to be done and in the direction FMC_PROGRAMMAPPING_FORWARDMAPPING when forward mapping is to be done.

Interface:

```
long Translate
  (void *  exitHandle,
   short   direction,
   char *  mappingName,
   long    mappingNameLength,
   char *  buildTimeParameter,
   long    buildTimeParameterLength,
   void *  containerHandle,
   char ** buffer,
   long *  bufferLength,
   char *  errorIdBuffer,
   long *  errorIdBufferLength,
   char *  errorDescriptionBuffer,
   long *  errorDescriptionBufferLength)
```

Parameters:

void * exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called at a later time. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

short direction

- Used to identify the translation direction. Either FMC_PROGRAMMAPPING_BACKWARDMAPPING or FMC_PROGRAMMAPPING_FORWARDMAPPING
- Input parameter

char * mappingName

- Forward mapping format or backward mapping format defined in the OS/390 program properties. The name is zero terminated.
- Input parameter

long mappingNameLength

- Length of the mappingNameLength in bytes.
- Input parameter

char * buildTimeParameter

- Forward mapping parameter or backward mapping parameter defined in the OS/390 program properties. Can be used to customize the translation process. The parameter is zero terminated.
- Input parameter

long buildTimeParameterLength

- Length of the buildTimeParameter in bytes.
- Input parameter

void * containerHandle

- Handle to the MQSeries Workflow for OS/390 container used for translation. Argument in MQSeries Workflow for OS/390 Container API calls to access the container.
- Input parameter

char ** buffer

- Pointer to a valid buffer address for forward mapping calls. It must be set by the program mapping exit to point to a valid buffer address in backward mapping calls. This buffer is passed by the PES to the legacy application in forward mapping calls and to the program mapping exit for data conversion and setting of container elements in backward mapping calls.
- Input parameter for forward mapping. The buffer is created and owned by the exit and passed to the PES.
- Output parameter for backward mapping. The buffer is created and owned by the PES and passed to the exit.

long * bufferLength

- Pointer to the length of the buffer in bytes. This is already set in forward mapping calls and must be set in backward mapping calls by the program mapping exit.
- Input parameter for forward mapping
- Output parameter for backward mapping

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

Return codes: See "Return codes and error messages" on page 224 for the return codes.

See the program mapping sample "Program mapping exit example" for more details.

Enabling the PES to use a program mapping exit

In order to use a program mapping exit, the exit must reside in a DLL which must be available in a link library used by the PES. Customize the sample JCL FMCHJMEX in *InstHLQ*.SFMCCNTL and submit the job.

You must then notify the PES of the program mapping exit by defining it in the PES directory. See *MQSeries Workflow for z/OS: Customization and Administration* for more information.

Program mapping exit example

A C sample program mapping exit (FMCHSMEX in *InstHLQ*.SFMCSRC) with a corresponding JCL to compile and link the exit (FMCHJMEX in *InstHLQ*.SFMCCNTL).

The sample exit gives an example how a program mapping exit should work in general. It also shows how the exit can use all the different parameters passed (exit initialization parameter, forward/backward mapping format and forward/backward mapping parameters), how error messages can be created and how the container elements can be accessed.

The exit can convert MQSeries Workflow for OS/390 container elements of type LONG and FLOAT into C types long and double. The elements are converted in the same sequence that the elements are defined in the container structure.

The type of the element is derived from the first character of the element name and must be defined in the PES directory entry for the mapping exit (exitParameters).

ExitParameter syntax: LONG=x FLOAT=y

where x is the character used to identify LONG container elements and y is the character to identify FLOAT container elements. All elements starting with an undefined character are ignored.

In addition TRACE=YES can be defined during initialization to enable trace printouts.

Mapping example:

```
STRUCTURE S
   LE1: LONG(2);
   FE1: FLOAT;
   LE2: LONG;
END;
```

will be converted so that the following structure is filled with conversion data:

```
struct S {
   long LE1[2];
   double FE1;
   long LE2;
};
```

The following three different mapping formats are available:

1. DEFAULT
2. INCREASE_INCOME
3. DECREASE_INCOME

The first format does normal conversion of the values. The second one increases all values by 8% (default), while the third format decreases all values by 8% (default). If other percentages are needed, they can be defined as forward mapping parameters or backward mapping parameters using the Buildtime tool when the program properties for OS/390 are defined.

To use the mapping sample, you must

1. Compile and link the mapping sample (See JCL FMCHJMEX in *InstHLQ*SFMCCNTL).
2. Update the PES directory with following definition (see "Importing a PES directory source file" in *MQSeries Workflow for z/OS: Customization and Administration*).

   (KEYTOMAPPING2)

   ```
   type           =SAMPLE
   exitName       =SAMPEXT
   exitParameters =LONG=L FLOAT=F
   ```
3. Define a program in BuildTime that uses the sample mapping type SAMPLE and optionally provides a different increase/decrease amount for forward/backward mapping parameters.

Program invocation exit

A program invocation exit is used by the program execution server to run a requested application on a service system like CICS or IMS. The corresponding invocation request is issued to the program execution server by an MQSeries Workflow for OS/390 execution server.

To handle an invocation request, the program execution server uses an exit containing the implementation of an invocation type such as *EXCI*. This allows application developers using MQSeries Workflow for OS/390 to attach their own invocation types to MQSeries Workflow for OS/390. An invocation exit must follow the general rules for program execution server exits as described in “Program execution server exits” on page 224 in this book. As for all program execution server exits, invocation exits are available as dynamic link libraries providing entry points.

Synchronous and asynchronous invocation exits

MQSeries Workflow for OS/390 supports synchronous and asynchronous invocation exits. They are characterized by the following. Asynchronous invocations must be based on MQSeries queues.

Synchronous invocation exit: A synchronous invocation exit

- establishes the connection to the service where the request should run.
- runs the requested application on that service by use of the invocation protocol.
- removes the connection to the service after the application has terminated.
- passes back the output data from the application, or an error message if the invocation failed, to the program execution server.
- if, depending on the invocation protocol, connections can be reused by subsequent calls, the connections are cached after being used a first time and removed at deinitialization of the invocation exit.

The program execution server ″waits″ for the output from a synchronous invocation exit.

Asynchronous invocation exit: When processing a request, the program execution server calls an asynchronous invocation exit twice. First to handle a request, and second to handle a reply. If an asynchronous invocation exit is called by the program execution server with parameters describing a request, it does not execute a request from the program execution server directly on the target service system. It only creates a message consisting of an invocation specific header followed by the data for the requested application and passes it back to the program execution server. It also creates a message descriptor (which is an MQSeries message descriptor MQMD since only MQSeries based invocations are supported) and passes it back also.

The program execution server then uses the message descriptor and message together with the connection parameters to put the message to the input queue as specified by the connection parameters. The program execution server does not wait until the requested application has finished but rather continues with the next request.

When the program execution server gets the reply message consisting of message descriptor and message data, it calls the respective invocation exit to recognize the reply message. It is recommended that only asynchronous invocation protocols be used for which the message contains protocol-specific data at the beginning, for example CICS bridge header MQCIH or IMS bridge header MQIIH, so that the reply can be correctly handled by the invocation exit. Also, it is recommended to reflect that format in the message descriptor (by the Format field in the MQMD structure).

If an exit recognizes the reply, it is called to handle it and passes the application output data, or an error message, back to the program execution server.

Note: An asynchronous invocation exit does **not** connect itself to a service. The MQSeries queues to and from the service are always served by the PES.

Additional interfaces specific to the invocation exit

Each asynchronous and synchronous invocation exit must provide the following additional entry points beside the interfaces provided by all program execution server exits:

HdlRequ()

Header files: FMCXIEP.H (invocation exit).

Function:

- For a synchronous invocation, this method executes an application program passed as **executableName** on a service to which it connects as defined by *connectionParameters*. If no error occurs, this function returns the output from the application in a buffer passed back by the **programOutput** parameter.
- For an asynchronous invocation this method either creates a request message or treats the passed parameters as a reply message. The parameters represent a request message if an application name is passed as **executableName**. If zero is passed as **executableName**, the parameters represent a reply message whose data is passed as **executableParameters** and **executableParametersLength** together with a message descriptor MQMD as **messageDescriptor** and **messageDescriptorLength**.

Interface:

```
long HdlRequ
   (void *  exitHandle,
    void *  invocationContext,
    char *  serviceName,
    long    serviceNameLength,
    char *  connectionParameters,
    long    connectionParametersLength,
    char *  executableName,
    long    executableNameLength,
    char *  executableType,
    long    executableTypeLength,
    char *  executionParameters,
    long    executionParametersLength,
    char ** programOutput,
    long *  programOutputLength,
    char ** messageDescriptor,
    char *  messageDescriptorLength,
    char *  errorIdBuffer,
    long *  errorIdBufferLength,
    char *  errorDescriptionBuffer,
    long *  errorDescriptionBufferLength)
```

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit
- Input parameter

void * invocationContext

- Address of data describing the context in which the invocation of the request is to be done.
- For the context information provided by MQSeries Workflow for OS/390, see "Invocation context" on page 237.
- Input parameter

char * serviceName

- Name of service as known to MQSeries Workflow for OS/390 where the requested application is to be performed.
- Input parameter

long serviceNameLength

- Length of serviceName character string
- Input parameter

char * connectionParameters

- Character string providing the parameters needed by the invocation to connect to the service where the requested program should run. These parameters are defined for the respective service in the PES directory.
- Input parameter
- See also "Connection parameters" on page 238

long connectionParametersLength

- Length of connectionParameters
- Input parameter

char * executableName

- Address of buffer containing the name of the executable of the request
- For asynchronous invocations only: the program execution server passes zero to indicate that this function is called to handle a reply message from an asynchronously invoked program execution request.
- Input parameter

long executableNameLength

- Length of executableName
- Input parameter

char * executableType

- Type of the program specified by the executableName
- Input parameter

long executableTypeLength

- Length of executableType
- Input parameter

char * executionParameters

- Parameters for the program specified by the executable name
- Reply from an asynchronously performed execution request
- Input parameter

long executionParametersLength

- Length of executionParameters or the asynchronous reply
- Input parameter

char ** programOutput

- Pointer to the address of return buffer containing the output of the executable
- Asynchronous invocations also use this parameter to return the protocol-conforming message to the program execution server corresponding to the execution request passed in **executableName** and **executionParameters**.
- This buffer is created and owned by the exit and passed to the PES.
- Input/output parameter

long * programOutputLength

- Pointer to length of output data or the protocol-conforming message

- Input/output parameter

char ** messageDescriptor

- Pointer to the address of a buffer containing a message descriptor
- For asynchronous invocations only
- If the executable name is passed (request message has been constructed), the buffer is created and owned by the exit and passed to the PES. If a reply message is being handled, the buffer is created and owned by the PES and passed to the exit.
- Input/output parameter

long * messageDescriptorLength

- Pointer to length of the message descriptor
- For asynchronous invocations only
- Input/output parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

Return codes:

FMC_EXIT_OK

- See "Return codes and error messages" on page 224 for detailed information.

FMC_EXIT_RECOVERABLE_ERROR

- See "Return codes and error messages" on page 224 for detailed information.

FMC_EXIT_NONRECOVERABLE_ERROR

- See "Return codes and error messages" on page 224 for detailed information.

Recogn()

Header files: FMCXIEP.H (invocation exit).

Function:

- Checks whether the reply message passed as message descriptor and message data is recognized by the invocation type this exit represents.
- This method applies to exits for asynchronous invocations. However, this entry point must also be provided by synchronous invocation exits. For these exits, FMC_INV_NOT_RECOGNIZED must always be returned.

- This function is called for all asynchronous invocation exits when an invocation reply message is received. It is assumed that there are no ambiguities, so that the first invocation exit where this function recognizes this message is the correct one to handle it.

Interface:

```
long Recogn
   (void * exitHandle
    char * messageDescriptor,
    long   messageDescriptorLength,
    char * messageData,
    long   messageDataLength)
```

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit
- Input parameter

char * messageDescriptor

- Descriptor of the reply message
- Input parameter

long messageDescriptorLength

- Length of messageDescriptor
- Input parameter

char * messageData

- The data of the reply message
- Input parameter

long messageDataLength

- Length of messageData
- Input parameter

Return codes:

FMC_INV_NOT_RECOGNIZED
The message is not recognized

FMC_INV_RECOGNIZED
The message is recognized

IsAsync()

Header files: FMCXIEP.H (invocation exit).

Function:

- Returns whether the exit represents an asynchronous invocation.

Interface:

```
long IsAsync
   (void* exitHandle)
```

Parameters:

void * exitHandle

- Reference to the invocation environment for this exit

- Input parameter

Return codes:

FMC_INV_SYNCHRONOUS
The invocation is synchronous.

FMC_INV_ASYNCHRONOUS
The invocation is asynchronous.

Invocation context

The PES passes an invocation context to the invocation exit for **HdlRequ()**. It contains information about the context in which the request should be executed. There are four different context types: Workflow, Security, Transaction, and Performance. The context is created by the PES. The content depends on server and program settings. The workflow context is passed for internal reasons and is not intended to be used by an invocation exit. The security context is either set to the PES user ID or the execution user ID resolved for the request. The transaction context is set to a nonzero value if the request is to be executed as a safe application. The performance context is set to the WLM enclave token if the PES is running WLM-managed.

An invocation accesses the context by using a C-type interface defined in the included file fmcxiinv.h.

GetContext()

Header files: FMCXIINV.H (invocation context).

Function:

- Extracts the value of the context type as specified by a passed context name.

Interface:

```
int GetContext
   (void *  invContext,
    char *  invContextName,
    char ** invContextValue,
    long *  invContextValueLength)
```

Parameters:

void * invContext

- Address of the complete invocation context containing all context types
- Input parameter

char * invContextName

- Name of the requested context type
- Can be one of the character string constants as listed in the *Name* column in the *Context types* table below.
- Input parameter

char ** invContextValue

- Address to which this function puts the pointer to the value of the requested context type
- Input/output parameter

long * invContextValueLength

- Address to which this function puts the length of the value of the requested context type
- Input/output parameter

Return codes:

FMC_INV_CTX_ON
Context successfully retrieved

FMC_INV_CTX_NOT_SET
Context has not been set

FMC_INV_CTX_NOT_DEFINED
Context not found

Table 16. Context types

Type	Meaning	Used by	Name
Workflow	Contains the Workflow context	Internal use only	FMC_WORKFLOW_CTX
Security	Contains the user identification the request should be executed for	Invocation exit	FMC_SECURITY_CTX
Transaction	Contains an indication that the request should be executed in a transactional context	Invocation exit	FMC_TRANSACT_CTX
Performance	Contains the WLM enclave token, if the PES is running WLM managed	Invocation exit	FMC_PERFMGMT_CTX

To link your exit correctly, you must include definition side deck FMCH0XIC from *InstHLQ*.SFMCDSD.

Connection parameters

The parameter *connectionParameters* of the entry point **HdlRequ()** represents a character string of up to 254 printable characters. This string contains the parameters needed by the invocation to connect to the service system in order to execute an application there.

Connection parameters for synchronous invocations: For synchronous invocations, these parameters and the syntax how to specify them must be defined by the developer of an invocation exit. In most cases it is recommended to use the following syntax:

```
<keyword_1>=<value_1>;<keyword_2>=<value_2>;...;<keyword_n>=<value_n>
```

where the parameters are represented by key-value pairs separated by semicolons. Nevertheless, you can define the syntax of the connection parameters for your invocation exit. Connection parameters and the syntax how to specify them must be part of the documentation of your invocation exit, since an MQSeries Workflow for OS/390 administrator must specify this string when defining this exit to the PES directory.

Connection parameters for asynchronous invocation: Because only asynchronous invocations based on MQSeries queues are supported, the connection parameters for asynchronous invocations and the syntax how to specify them must be

```
[QUEUEMANAGER=<queuemanagername>;]INPUTQUEUE=<inputqueuename>
```

where <queuemanagername> and <inputqueuename> represent the MQSeries queue manager or MQSeries queue, respectively, and must follow the corresponding MQSeries naming rules. These parameters define the MQSeries queue to which the request message created by an asynchronous invocation exit must be put. QUEUEMANAGER is optional and should be omitted when the target queue manager is part of a queue manager cluster. When work distribution is required, INPUTQUEUE should specify a cluster queue defined with the attribute DEFBIND=<NOTFIXED>. At least two queues with this name should exist within the cluster.

Enabling MQSeries Workflow for OS/390 to use an invocation exit

In order to use a program invocation exit, the exit must reside in a DLL which must be available in a link library used by the PES.

Note:
If more than one asynchronous invocation exit recognizes the same kind of reply messages, it is unpredictable which of the exits will handle a reply message. This may lead to incorrect results, the cause of which may be difficult to determine.

See also "Program execution" in *MQSeries Workflow for z/OS: Customization and Administration*.

Invocation exit coding example

Since an invocation exit assumes there is a special kind of service system available where an application should run, there are no compiling sources provided as samples but rather a skeleton for synchronous invocations FMCHSIVS and one for asynchronous invocations FMCHSIVA, both in *InstHLQ*.SFMCSRC and written in C.

Notification exit

The PES supports an exit that provides notification whenever one of the following events happens in the PES:

- The PES is about to invoke a program
- The PES has successfully invoked a program
- A failure occurs when the PES tries to invoke a program.

The exit can be used for various purposes, such as to write journal records. It can perform its operation in either transaction or non-transaction mode. If in transaction mode, the exit must participate in the global transactions that are controlled by the PES. In particular, this means that the exit must not issue commit or rollback requests and that it must use resource managers that are coordinated by OS390/RRS (see “Use of RRS commit and rollback” on page 227).
Non-transaction mode implies that the exit does not record its persistent data via resource managers but rather directly in data sets, for example.

It is strongly recommended that the exit work in transaction mode, since it is otherwise not possible to synchronize PES actions with those of the exit.

The notification exit must be available in a DLL that is loaded by the PES and used to call the exit. This DLL must have the name FMCXNEXT. IBM does not supply

an executable for this exit, and it must therefore be provided by the customer. To activate the notification exit, the variable **PesNotificationExit** must be set to 1 in the configuration profile. If this variable is set, the PES tries to load and initialize the notification exit at PES startup time. If the variable is set and the exit cannot be loaded or initialized correctly, the PES will not start properly.

The notification exit must have the two exit functions **Init()** and **Deinit()** common to all PES exits (see "Interfaces for all exits" on page 225).

When the **Init()** function is called, the `initializationParameters` buffer has two key=value pairs in the following format:

```
ASID=<asid>;TCBID=<tcbid>
```

where `<asid>` and `<tcbid>` are the address space and task control block IDs, respectively, of the corresponding PES instance.

Additional interfaces specific to the notification exit

The entry points specific to the notification exit are:

- Notification of a 'before invocation' event (**EVB4INV()**)
- Notification of a 'successful invocation' event (**EVIVSUCC()**)
- Notification of an 'invocation failure' event (**EVIVFAIL()**)

When the PES receives a request to start a program, it performs the following steps:

1. Phase Ia processing
   - Validate message parameters
   - Look up connection parameters in directory
   - Retrieve local user ID and check user authorization if required
2. Phase Ib processing
   - Retrieve invocation exit information
   - Load and initialize invocation exit if not already done
   - Retrieve mapping exit information
   - Load and initialize mapping exit if not already done
   - Map forward if desired
3. Phase II processing
   - Invoke the program
   - Map backward if desired
   - Send reply message to originator of the request

The notification exit function **EVB4INV()** is called immediately after Phase Ia processing is complete. If an error occurs during Phase Ia, an error message is returned to the originator of the request, but the notification exit is not called.

If an error occurs during Phases Ib or II, the notification exit function **EVIVFAIL()** is called, and an error message is returned to the originator of the request.

The PES passes to the notification exit functions all data required to identify the workflow request that is currently being processed. In particular, this means that the program's containers are passed to the exit. The exit functions can use the Container API to access the contents of the containers.

In order to enable correlation between **EVB4INV()** and **EVIVSUCC()/EVIVFAIL()** calls, the PES passes the MQ MessageID of the received InvokeProgram request message to all notification entry points. This ID can be used as a correlation ID that is unique for each program invocation request processed by the PES (called program invocation instance).

The exit behavior differs depending on whether the program invocation is synchronous or asynchronous:

Synchronous invocation: When the PES cannot invoke a program correctly (that is, as soon as it detects an error after calling **EVB4INV()**, which can be a mapping, invocation, or directory error), it calls **EVIVFAIL()**. It passes **EVIVFAIL()** the same MQ Message ID buffer that was passed to **EVB4INV()** as well as the available error information.

If the invocation is successful, the PES calls **EVIVSUCC()** as soon as the output containers of the program are available, that is, after calling the mapping exit for BACKWARD_MAPPING if MAPPING is specified in the definition of the external program, or after the invocation exit returns and the PES has constructed the output container if NO MAPPING is specified in the definition of the external program.

The PES passes **EVIVSUCC()** the same MQ Message ID buffer that was passed to **EVB4INV()**, as well as the program's output container.

Asynchronous invocation: If the invocation is asynchronous, the processing of the program's reply data takes place in a second transaction. If program invocation is successful, the PES calls **EVIVSUCC()** in this second transaction after **HdlRequ()** returns from handling the program's reply and the output containers are available (either by direct action in the PES or by calling the mapping exit for BACKWARD_MAPPING).

If an error occurs in the process (e.g. **Recogn()**, **HdlRequ()**, or the mapping exit returns an error), the PES will call **EVIVFAIL()** as soon as it detects the error condition.

Regardless of the invocation type, there are two calls to the notification exit for each program invocation: **EVB4INV()** and **EVIVSUCC()** if the invocation is successful, or **EVB4INV()** and **EVIVFAIL()** otherwise.

EVB4INV()

Header file: FMCXNEIF.H (notification exit).

Function:

- This exit function is called when the PES is about to invoke a program. It is called by the PES before it prepares the program's input buffer. If forward mapping is desired, **EVB4INV()** is called before the mapping exit for forward mapping is called.

Interface:

```
long EVB4INV
    (void * exitHandle,
     void * inputContainerHandle,
     void * invocationContext,
     char * userName,
     long   userNameLength,
```

```
void * workFlowCorrelId,
long   workFlowCorrelIdLength,
char * serviceName,
long   serviceNameLength,
char * connectionParameters,
long   connectionParametersLength,
char * executableName,
long   executableNameLength,
char * executableType,
long   executableTypeLength,
void * rqMQMessageID,
long   rqMQMessageIDLength,
char * errorIdBuffer,
long * errorIdBufferLength,
char * errorDescriptionBuffer,
long * errorDescriptionBufferLength)
```

Parameters:

void * exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called subsequently. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

void * inputContainerHandle

- Handle of the MQSeries Workflow for OS/390 input container. Argument in MQSeries Workflow for OS/390 Container API calls to access the container.
- Input parameter

void * invocationContext

- Address of the data describing the context in which the invocation of the request is to be done.
- For the context information provided by MQSeries Workflow for OS/390, see "Invocation context" on page 237.
- Input parameter

char * userName

- String containing the name of the user on whose behalf the program is started.
- Input parameter

long userNameLength

- Length of the userName string.
- Input parameter

void * workFlowCorrelId

- Binary buffer containing the workflow correlation ID. This field can be used to correlate the current program invocation instance with the state of the workflow's process instance. The field contains either an ActImplCorrelID structure or an FmcCorrelationID structure, depending on the originator of the start program request:
  - an FmcActImplCorrelID structure if the program start request comes from MQ Workflow's execution server
  - an FmcCorrelationID structure if the program start request comes from a client application that issued an ExecuteProgram API call (**FmcjProgramTemplateExecute()** function).

The field can also be empty, that is, the parameter can have the value NULL or the length can be 0.

- Input parameter

long workFlowCorrelIdLength

- Length of the workFlowCorrelId buffer. If 0, the buffer is considered empty.
- Input parameter

char * serviceName

- Name of the service, as known to MQSeries Workflow for OS/390, on which the application is to be performed.
- Input parameter

long serviceNameLength

- Length of the serviceName character string.
- Input parameter

char * connectionParameters

- Character string providing the parameters needed by the invocation to connect to the service under which the requested program is to run. These parameters are defined for the respective service in the PES directory.
- Input parameter
- See also "Connection parameters" on page 238.

long connectionParametersLength

- Length of connectionParameters
- Input parameter

char * executableName

- Address of buffer containing the name of the executable of the request
- Input parameter

long executableNameLength

- Length of executableName
- Input parameter

char * executableType

- Type of the program specified by the executableName
- Input parameter

long executableTypeLength

- Length of executableType
- Input parameter

void * rqMQMessageID

- Buffer that contains the MQ Message ID of the InvokeProgram request. This ID uniquely identifies the current program invocation instance. The same ID will be passed to **EVIVSUCC()/EVIVFAIL()** in order to enable correlation.
- Input parameter

long rqMQMessageIDLength

- Length of the rqMQMessageID buffer

- Input parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

EVIVSUCC()

Header files: FMCXNEIF.H (notification exit).

Function:

- Notifies the exit that a program invocation has been successfully completed.

Invocations: **EVIVSUCC()** is invoked:

- For synchronous invocations: after the invocation exit function **HdlRequ()** returns to the PES with a return code of 0 and the output container is available (either by calling the mapping exit or by internal actions).
- For asynchronous invocations: when the PES has received the program's reply data from the MQ bridge, successfully called **Recogn()** and the **HdlRequ()** function of the appropriate invocation exit, and constructed the output container (either by calling the mapping exit or by internal actions).

Interface:

```
long EVIVSUCC
    (void * exitHandle,
     void * outputContainerHandle,
     void * rqMQMessageID,
     long   rqMQMessageIDLength,
     char * errorIdBuffer,
     long * errorIdBufferLength,
     char * errorDescriptionBuffer,
     long * errorDescriptionBufferLength)
```

Parameters:

void * exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called subsequently. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

void * outputContainerHandle

- Handle of the MQSeries Workflow for OS/390 output container. Argument in MQSeries Workflow for OS/390 Container API calls to access the container.
- Input parameter

void * rqMQMessageID

- Buffer that contains the MQ Message ID of the InvokeProgram request. This ID uniquely identifies the current program invocation instance. The same ID was passed to **EVB4INV()**.
- Input parameter

long rqMQMessageIDLength

- Length of the rqMQMessageID buffer
- Input parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

EVIVFAIL()

Header files: FMCXNEIF.H (notification exit).

Function:

- Notifies the exit that an attempt to invoke a program has failed.

Invocations: The PES will call **EVIVFAIL()** as soon as it detects an error after calling **EVB4INV()**.

- For synchronous invocations, this can be one of the following situations:
  - **EVB4INV()** returns an error to the PES.
  - The PES cannot retrieve the information for the invocation exit from the directory.
  - The invocation exit cannot be loaded and initialized correctly.
  - The PES cannot retrieve the information for the mapping exit from the directory.
  - The mapping exit cannot be loaded and initialized correctly.
  - The PES is unable to prepare the program's input buffer.
  - The mapping exit for FORWARD_MAPPING returns an error.
  - The **HdlRequ()** call of the invocation exit returns an error.
  - The mapping exit for BACKWARD_MAPPING returns an error.
  - The PES is not able to construct the output container of the program.
- For asynchronous invocations, this can be one of the following situations:
  - **EVB4INV()** returns an error to the PES.
  - The PES cannot retrieve the information for the invocation exit from the directory.
  - The invocation exit cannot be loaded and initialized correctly.

- The PES cannot retrieve the information for the mapping exit from the directory.
- The mapping exit cannot be loaded and initialized correctly.
- The PES is unable to prepare the program's input buffer.
- The mapping exit for FORWARD_MAPPING returns an error.
- The **HdlRequ()** call of the PES transaction that processes the program invocation returns an error.
- The program is invoked, but an error occurs in the transaction that processes the program's reply data. Possible errors are:
  - **Recogn()** returns an error.
  - **HdlRequ()** returns an error.
  - The mapping exit for BACKWARD_MAPPING returns an error.
  - The PES is not able to construct the output container of the program.

Interface:

```
long EVIVFAIL
    (void * exitHandle,
     void * rqMQMessageID,
     long   rqMQMessageIDLength,
     char * processingErrorId,
     long   processingErrorIdLength,
     char * processingErrorDescription,
     long   processingErrorDescriptionLength,
     char * errorIdBuffer,
     long * errorIdBufferLength,
     char * errorDescriptionBuffer,
     long * errorDescriptionBufferLength)
```

Parameters:

void * exitHandle

- Pointer passed to the **Init()** function, which is not used by the PES but passed to any function called subsequently. Used to pass exit environment data between the PES and all exit functions.
- Input/output parameter

void * rqMQMessageID

- Buffer that contains the MQ Message ID of the InvokeProgram request. This ID uniquely identifies the current program invocation instance. The same ID was passed to **EVB4INV()**.
- Input parameter

long rqMQMessageIDLength

- Length of thg rqMQMessageID buffer
- Input parameter

char * processingErrorId

- Character buffer that contains the ID of the error that occurred in the processing of the current program invocation instance since the call to **EVB4INV()**. This error ID may have been set by the directory, invocation, or mapping exit, or by the PES.
- Input parameter

long processingErrorIdLength

- Length of the processingErrorId buffer.
- Input parameter

char * processingErrorDescription

- Character buffer that contains the description of the error that occurred in the processing of the current program invocation instance since the call to **EVB4INV()**. This error description may have been set by the directory, invocation, or mapping exit, or by the PES.
- Input parameter

long processingErrorDescriptionLength

- Length of the processingErrorDescription buffer.
- Input parameter

char * errorIdBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorIdBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

char * errorDescriptionBuffer

- See "Return codes and error messages" on page 224 for detailed information.

long * errorDescriptionBufferLength

- See "Return codes and error messages" on page 224 for detailed information.

Return codes: See "Return codes and error messages" on page 224 for the return codes. If **EVIVFAIL()** returns FMC_EXIT_OK, the PES sends the original error back to the originator of the request, that is, in processingErrorID/processingErrorDescription. If **EVIVFAIL()** returns FMC_EXIT_RECOVERABLE_ERROR or FMC_EXIT_NONRECOVERABLE_ERROR, the PES sends the error from **EVIVFAIL()** back to the originator of the request, that is, the error that is returned in errorIDBuffer and errorDescriptionBuffer.

Chapter 4. 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 import packages take this fact into account. IMS programs can only use the "Container API".

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

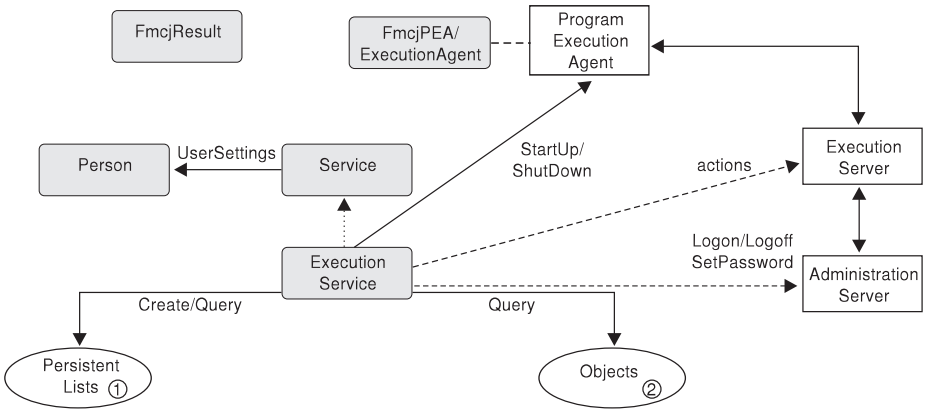


Figure 36. Setting up client/server communication. Legend: --➔ Inheritance (C++); ⟶ provides for access; — ⟶ sends messages to

As a first step, an ExecutionService object must be obtained (constructed/allocated/located). An ExecutionService object represents a session between a user and an MQ Workflow execution server. It essentially provides the basic API calls to set up a communication path to the specified MQ Workflow execution server and to establish the user session (Logon() respectively Passthrough()), and finish it (Logoff()). To log on, not only the execution server but also the administration server must be up and running so that authentication can be done. This is, however, transparent to you.

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

- Query objects for which you are authorized: process templates, process instances, items (work items, activity instance notifications, process instance notifications), or lists containing such objects.

- Create persistent lists, that is, persistent views on objects contained in the MQ Workflow database.
- Query information about the logged-on user or change that user's password.

In C and C++, all API call calls update a so-called result object. Detailed information about an erroneous request can be obtained from there. See "Handling errors" on page 4 for more information.

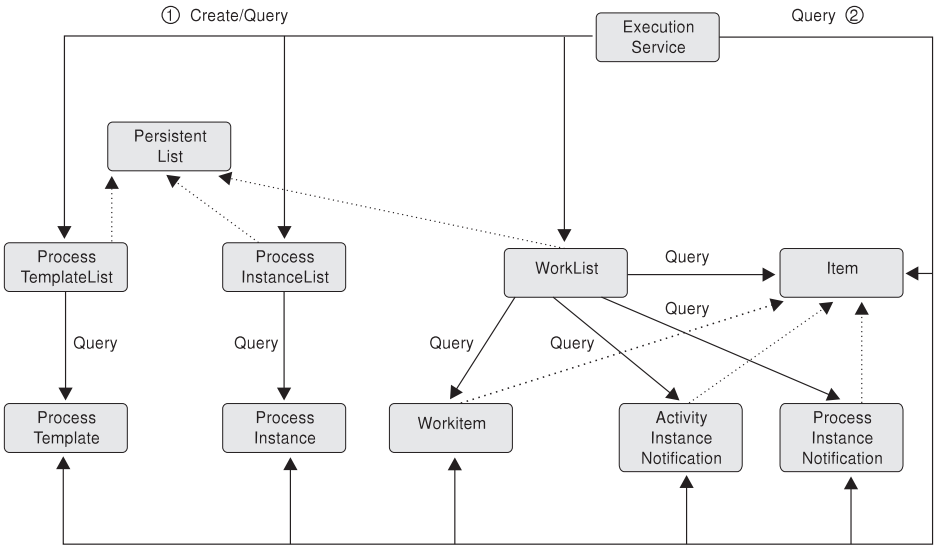


Figure 37. Querying objects. Legend: --► Inheritance (C++); ⟶ provides for access

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.

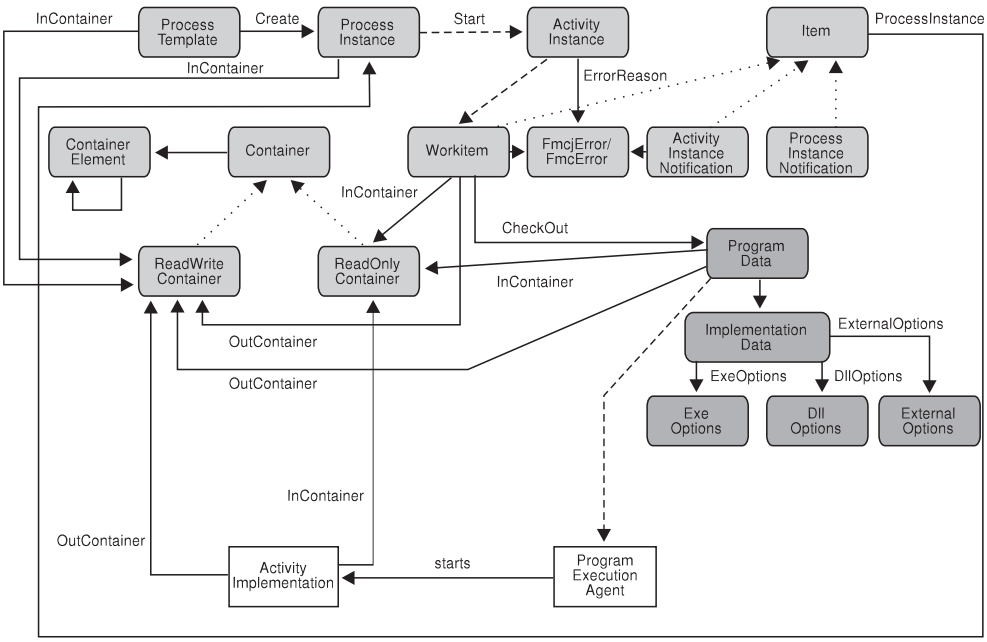


Figure 38. Dealing with process instances and (work) items. Legend: --➔ Inheritance (C++); ⟶ provides for access; — ⟶ data is passed to or results in

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

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

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

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

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

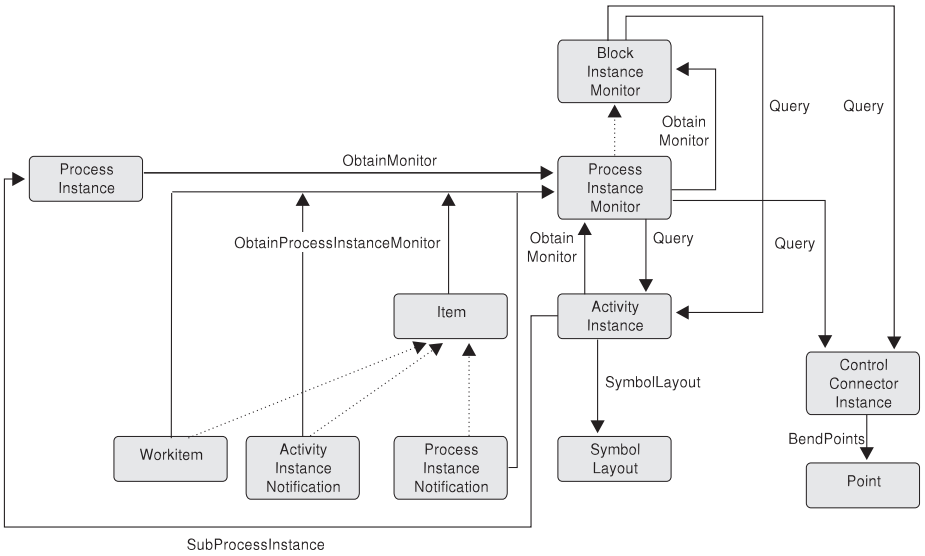


Figure 39. Monitoring a process instance. Legend: --➤ Inheritance (C++); ⟶ provides for access

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

Once a process instance monitor has been obtained, you can iterate into the process model by obtaining block instance monitors for activities of type Block or process instance monitors for activities of type Process, that is, for subprocess instances. See “Monitoring a process instance” on page 67 for more information.

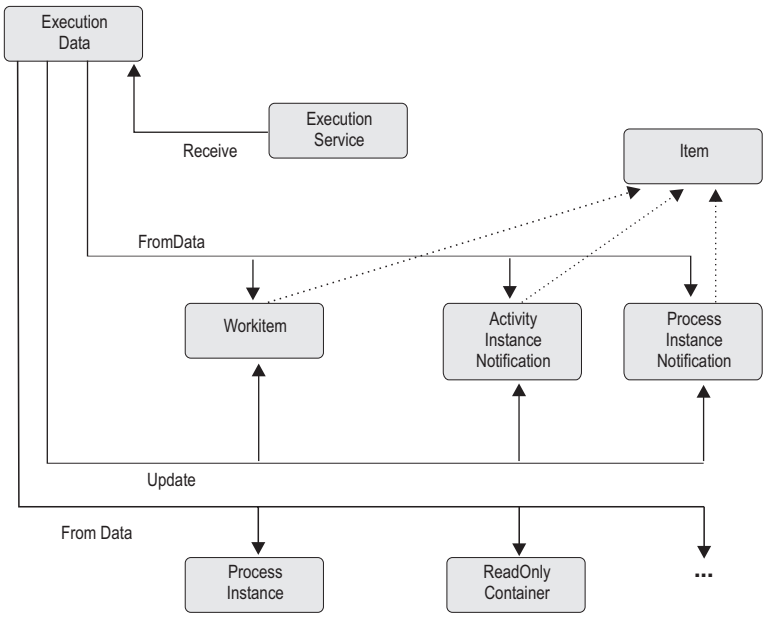


Figure 40. Handling data sent by an MQ Workflow server. Legend: -➤ Inheritance (C++); ⟶ provides for access

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 "Client/server communication and data access models" on page 12 for more information.

API classes

An alphabetical list of API classes or function prefixes follows. Unless otherwise stated, all indicated names are valid Java classes. To become valid C++ classes, a prefix of Fmcj has to be added. To become valid C-language or COBOL calls, the class name must be prefixed with Fmcj and extended by the actual function name. For example, if your Java class is Workitem, then your C++ class is named FmcjWorkitem and all your functions in the C-language or COBOLstart with FmcjWorkitem; FmcjWorkitemStart, for example, is a valid COBOL subprogram name or C-language function.

Note: In the following discussions, use of the listed class names should be understood as a generic designation that also applies to C, C++, and COBOL.

To adhere to language requirements, the paragraph names in copybook fmcperf.cpy are usually abbreviated versions of the C function names. For more information concerning COBOL naming conventions, see "COBOL interface" on page 148.

Class/Object	Description
ActivityInstance	An instance of a workflow process template activity.
ActivityInstanceNotification	A notification associated with an activity instance.

API classes

Class/Object	Description
ActivityInstanceNotificationVector	The C-language result of a query for activity instance notifications.
ActivityInstanceVector	The C-language result of a query for activity instances.
Agent	An agent in the Java API to access an MQ Workflow domain.
Container	The data container of a work item or a process instance.
ContainerElement	An element of a data container.
ContainerElementVector	The C-language result of a query for container elements.
ControlConnectorInstance	The instance of a control connector between two activity instances.
ControlConnectorInstanceVector	The C-language result of a query for control connector instances.
DateAndTime	Representation of date and time values. FmcjCDateTime is the C-language equivalent structure. The C++ class is called FmcjDateTime. Java uses the Calendar object.
DllOptions	The program implementation definitions for a dynamic link library.
ExecutionData	Information pushed by an MQ Workflow execution server or the response to an asynchronous request.
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.
ExeOptions	The program implementation definitions for an executable.
ExternalOptions	The program implementation definitions for an external service.
FmcError	Describes the cause of a state InError in Java. The C++ class is called FmcjError.
FmcException	The Java representation of an exception.
Global	A means to group API calls which are global API API calls in C and C++.
ImplementationData	The program implementation definitions.
InstanceMonitor	The monitor for a process instance or an activity instance of type *Block* or *Process*.
Item	An item associated to a user; can be a work item or notification.
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++.

Class/Object	Description
PersistentList	A list definition stored persistently.
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.
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.
ProcessInstanceListVector	The C-language result of a query for process instance lists.
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.
ProcessTemplateListVector	The C-language result of a query for process template lists.
ProcessTemplateVector	The C-language result of a query for process templates.
ProgramData	The program definitions of an activity implementation.
ProgramTemplate	A program definition contained in a process template.
ReadOnlyContainer	A data container that can only be read.
ReadOnlyContainerHolder	In Java, a data container that contains the output container of a process instance; returned by executeProcessInstance().
ReadWriteContainer	A data container that can be read and written to.
Result	The detailed result of a request; only C-language and C++.
Service	Provides for common aspects of MQ Workflow services.
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.
WorkitemVector	The C-language result of a query for work items.
Worklist	A list to group work items or notifications.
WorklistVector	The C-language result of a query for worklists.

API calls per class

Note: In the following descriptions, the basic methods listed are not differentiated by programming language. Not all of these methods are available in each language.

ActivityInstance

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an activity instance object.	77
Copy()	Allocates and initializes the storage for an activity instance object by copying.	81
Deallocate()	Deallocates the storage for an activity instance object.	82
destructor()	Destructs an activity instance object.	82
Equal()	Compares two activity instances.	79
IsComplete()	Indicates whether the complete activity instance information is available.	82
IsEmpty()	Indicates whether no activity instance information is available.	83
Kind()	States the kind of the activity instance, whether it is a program, a process, or a block.	83, 102
operator=()	Assigns an activity instance to this one.	79
operator==()	Compares two activity instances.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache. Note that the activity instances returned by an instance monitor fetched from the server contain both primary and secondary values.

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

Accessor methods	Set/ Type	Description	Page
ActivationTime()	P/D	Returns the activation time of the activity instance.	95

Accessor methods	Set/ Type	Description	Page
ActivationTimeIsNull()	P/B	Indicates whether an activation time is set.	121
Category()	P/C	Returns the process category of the activity instance.	117
CategoryIsNull()	P/B	Indicates whether a category is set.	121
Description()	P/C	Returns the description of the activity instance.	117
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the activity instance.	117
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the ending time of the activity instance.	95
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
ExitCondition()	S/C	Returns the exit condition of the activity instance.	117
ExpirationTime()	S/D	Returns the expiration time of the activity instance.	95
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	121
FirstNotificationTime()	S/D	Returns the time the first notification for the activity instance is to occur or has occurred.	95
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).	117
Icon()	P/C	Returns the icon associated with the activity instance.	117
Implementation()	P/C	Returns the name of the implementing program of the activity instance.	117
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	121
InContainerName()	S/C	Returns the name of the input container of the activity instance.	117

Activity instance

Accessor methods	Set/ Type	Description	Page
LastModificationTime()	P/D	Returns the last time a primary attribute of the activity instance was changed.	95
LastStateChangeTime()	P/D	Returns the last time the state of the activity instance changed.	95
ManualExitMode()	S/B	Returns whether the exit mode of the activity instance is manual.	94
ManualStartMode()	S/B	Returns whether the start mode of the activity instance is manual.	94
Name()	P/C	Returns the name of the activity instance.	117
OutContainerName()	S/C	Returns the name of the output container of the activity instance.	117
PersistentOid()	P/C	Returns a representation of the object identification of the activity instance.	117
Priority()	P/I	Returns the priority of the activity instance.	116
PriorityIsNull()	P/B	Indicates whether a priority is set.	121
ProcessAdmin()	S/C	Returns the process administrator of the activity instance.	117
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.	117
ProcessInstanceState()	P/E	Returns the state of the process instance the activity instance is part of.	96, 114
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	117
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the activity instance is part of.	117
SecondNotificationTime()	S/D	Returns the time the second notification for the activity instance is to occur or has occurred.	95
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 is assigned to.	118
StartCondition()	S/C	Returns the start condition of the activity instance.	117

Accessor methods	Set/ Type	Description	Page
Starter()	P/C	Returns the starter of the activity instance or the requestor of a ForceFinish action.	117
StarterIsNull()	P/B	Indicates whether a starter is set.	121
StartTime()	P/D	Returns the start time of the activity instance.	95
StartTimeIsNull()	P/B	Indicates whether a start time is set.	121
State	P/E	Returns the state of the activity instance.	96, 99
StateOfNotification()	S/E	Returns the notification state of the activity instance.	96, 99
SupportTools()	P/M	Returns the support tools associated with the activity instance.	118
SymbolLayout()	S/O	Returns the symbol layout of the activity instance.	119

Refer to “Action API calls” on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
ForceFinish()	Force finishes the activity instance.	317
ForceFinish2()	In Java, force finishes the activity instance and passes a container.	317
ForceRestart()	Forces the restart of the activity instance.	320
ForceRestart2()	In Java, force restarts the activity instance and passes a container.	320
InContainer()	Retrieves the input container of the activity instance.	322
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance the activity instance is part of.	324
OutContainer()	Retrieves the output container of the activity instance.	326
Refresh()	Refreshes the specified activity instance from the server.	328
SubProcessInstance()	Retrieves the process instance implementing the activity instance of type Process.	330
Terminate()	Terminates the specified activity instance.	332

ActivityInstanceNotification

An activity instance notification represents a notification for an activity instance. **All API calls of FmcjItem are also applicable to activity instance notifications.**

Activity instance notification

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an activity instance notification object.	77
Copy()	Allocates and initializes the storage for an activity instance notification ocopybject by copying.	81
Deallocate()	Deallocates the storage for an activity instance notification object.	82
destructor()	Destructs an activity instance notification object.	82
Kind()	In the C++ language, states that the object is an activity instance notification.	83, 112
operator=()	Assigns an activity instance notification to this one.	79
operator==()	Compares two activity instance notifications.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when activity instance notifications are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific activity instance notification. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
ActivityKind()	P/E	Returns the kind of the associated activity instance, whether it is a program or process and so on.	83, 102
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.	117
ExpirationTime()	S/D	Returns the expiration time of the associated activity instance.	95
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	121
FirstNotificationTime()	S/D	Returns the first notification time of the activity instance, that is, the time when this notification has been created.	95

Accessor methods	Set/ Type	Description	Page
Implementation()	P/C	Returns the implementing program or process name of the associated activity instance.	117
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.	94
ManualStartMode()	S/B	Returns whether the start mode of the associated activity instance is manual.	94
PersistentOidOfActivityInstance()	P/C	Returns the object ID of the associated activity instance.	117
Priority()	P/I	Returns the priority of the associated activity instance.	116
SecondNotificationTime()	S/D	Returns the second notification time of the associated activity instance.	95
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.	117
State	P/E	Returns the state of the associated activity instance.	96, 99
StateOfNotification()	S/E	Returns the notification state of the associated activity instance.	96, 99
SupportTools()	P/M	Returns the support tools associated with the activity instance.	118
SupportToolsIsNull()	P/B	Indicates whether support tools are set.	121

Refer to “Action API calls” on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
ActivityInstance()	Retrieves the associated activity instance.	335
StartTool()	Starts the specified support tool.	337
ObtainProcessMonitor()	Returns the instance monitor for the associated process instance.	429

ActivityInstanceNotificationVector

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

Activity instance notification vector

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector functions.

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

ActivityInstanceVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

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

Agent

An agent object represents an MQ Workflow instance in Java.

Refer to "Basic API calls" on page 76 for detailed descriptions of basicAPI calls.

Basic methods	Description	Page
constructor()	Constructs an agent object. Initially an agent has no context, locator policy, or name.	77

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
addPropertyChangeListener()	O	Adds the specified listener to the set of listeners to be notified of property changes.	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.	117
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.	116
getName()	C	Returns the name of the agent. If the agent is not bound, an empty string is returned.	117
isBound()	B	Indicates whether the agent bean is bound to a Java CORBA agent.	94
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 by the agent. A locator policy of LOC_LOCATOR is automatically set. Such, a java.beans.PropertyVetoException is thrown when the policy has already been set using the setLocator() method. Only one configuration can be used per application process. Configuration IDs for other than the LOC_LOCATOR policies are to be passed as command line parameters to the agent program.	117
setContext()	O	Sets the context of the agent. This call must precede a setName() call. An applet must set the context by issuing an *agent.setContext(this,null);*	119

Accessor methods	Type	Description	Page
setLocator()	I	Sets the locator policy; can be COS_LOCATOR, IOR_LOCATOR, LOC_LOCATOR, OSA_LOCATOR, RMI_LOCATOR. If LOC_LOCATOR is set, the default configuration ID for profile access is automatically used. Such, a java.beans.PropertyVetoException is thrown when the policy has already been set using the setConfigurationID() method. Configuration IDs for other than the LOC_LOCATOR policies are to be passed as command line parameters to the agent program. This call must precede a setName() call. **Note:** Java RMI agents should only be used for prototyping. They are currently not suited for production purposes.	121
setName()	C	Sets the name of the agent to be contacted as the result of this call.	117
toString()	C	Returns the name of the agent.	117
versionInfo()	C	Returns the version of the agent, that is, only useful formation is returned when the agent is bound.	117

Container

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

Refer to “Basic API calls” on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
IsEmpty()	Indicates whether no container information is available.	83

Refer to “Accessor/mutator API calls” on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
AllLeafCount()	I	Returns the number of leaf elements of the container including the MQ Workflow predefined members.	38

Accessor methods	Type	Description	Page
AllLeaves()	M	Returns all leaf elements of the container including the MQ Workflow predefined members.	38
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.	50
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.	50
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.	50
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.	50
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.	51
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.	51
AsStream()	C	Returns the container as a binary stream of data.	117
BinaryLength()	I	Returns the length of the value of the specified container leaf element. The leaf is of type BINARY.	50,51
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.	50
FloatValue()	F	Returns the value of the specified container leaf element in the C-language. The leaf is of type FLOAT.	50
getBuffer()	C	Returns the value of the specified container leaf element in Java. The leaf is of type BINARY.	52
getBuffer2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type BINARY.	52
getDouble()	F	Returns the value of the specified container leaf element in Java. The leaf is of type FLOAT.	52
getDouble2()	F	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type FLOAT.	52
GetElement()	O	Provides access to a container element.	48
getLong()	I	Returns the value of the specified container leaf element in Java. The leaf is of type LONG.	52

Container

Accessor methods	Type	Description	Page
getLong2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type LONG.	52
getString()	C	Returns the value of the specified container leaf element in Java. The leaf is of type STRING.	52
getString2()	C	Returns the value of the specified container leaf element in Java. The leaf is part of an array and of type STRING.	52
LeafCount()	I	Returns the number of user-defined leaf elements of the container.	38
Leaves()	M	Returns all user-defined leaf elements of the container.	38
LongValue()	I	Returns the value of the specified container leaf element in the C-language. The leaf is of type LONG.	50
MemberCount()	I	Returns the number of structural members in the container.	39
SetStringCcsid()	C	Sets the CCSID to be used for reading or writing strings in the container.	117
StreamLength()	I	In the C, C++, and COBOL languages, returns the length of the buffer needed to hold the container in stream format.	116
StringLength()	I	Returns the length of the value of the specified container leaf element in the C-language. The leaf is of type STRING.	51
StringValue()	C	Returns the value of the specified container leaf element in the C-language. The leaf is of type STRING.	51
StructMembers()	M	Returns the structural members of the container.	39
Type()	C	Returns the type of the container, that is, the data structure name.	40
Value()	C/I/F/N	Returns the value of the specified container leaf element in the C++ language.	51

Refer to “Activity implementation API calls” on page 133 for detailed descriptions of activity implementation API calls.

Activity implementation methods	Description	Page
InContainer()	Accesses the input container from within an activity implementation; for Java see the ExecutionAgent.	340
OutContainer()	Accesses the output container from within an activity implementation; for Java see the ExecutionAgent.	341
SetOutContainer()	Sets the output container from within an activity implementation; for Java see the ExecutionAgent.	343

ContainerElement

A container element represents an arbitrary element of a container.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a container element object.	77
Copy()	Allocates and initializes the storage for a container element object by copying.	81
Deallocate()	Deallocates the storage for a container element object.	82
destructor()	Destructs a container element object.	82
Equal()	Compares two container elements.	79
operator=()	Assigns a container element to another one.	79
operator==()	Compares two container elements.	79
IsEmpty()	Indicates whether no container element information is available.	83

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties because a container element describes a part of a container.

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

Accessor methods	Type	Description	Page
ArrayBinaryLength()	I	Returns the length of the value of the specified container element leaf element in the C-language. The leaf is part of an array and of type BINARY.	56
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.	56
ArrayElements()	M	Returns the array elements of the container element.	47
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.	56
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.	56
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.	57
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.	57

Container element

Accessor methods	Type	Description	Page
BinaryLength()	I	Returns the length of the value of the specified container element leaf element. The leaf is of type BINARY.	56,57
BinaryValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is of type BINARY.	56
Cardinality()	I	Returns the number of array elements of the container element.	47
FloatValue()	F	Returns the value of the specified container element leaf element in the C-language. The leaf is of type FLOAT.	56
FullName()	C	Returns the fully-qualified dotted name of the container element.	41
getBuffer()	C	Returns the value of the specified container element leaf element in Java. The leaf is of type BINARY.	58
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.	58
getDouble()	F	Returns the value of the specified container element leaf element in Java. The leaf is of type FLOAT.	58
getDouble2()	F	Returns the value of the specified container element leaf element in Java. The leaf is part of an array and of type FLOAT.	58
GetElement()	O	Provides access to an element of the container element.	48
getLong()	I	Returns the value of the specified container element leaf element in Java. The leaf is of type LONG.	58
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.	58
getString()	C	Returns the value of the specified container element leaf element in Java. The leaf is of type STRING.	58
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.	58
IsArray()	B	Indicates whether the container element is an array.	43
IsLeaf()	B	Indicates whether the container element is a leaf.	43
IsStruct()	B	Indicates whether the container element is a structure itself.	43
LeafCount()	I	Returns the number of leaf elements of the container element.	44
Leaves()	M	Returns all leaf elements of the container element.	44

Accessor methods	Type	Description	Page
LongValue()	I	Returns the value of the specified container element leaf element in the C-language. The leaf is of type LONG.	56
MemberCount()	I	Returns the number of structural members in the container element.	46
Name()	C	Returns the name of the container element.	41
StringLength()	I	Returns the length of the value of the specified container element leaf element in the C-language. The leaf is of type STRING.	57
StringValue()	C	Returns the value of the specified container element leaf element in the C-language. The leaf is of type STRING..	57
StructMembers()	M	Returns the structural members of the container element.	46
Type()	C	Returns the type of the container element, that is, the data structure name.	41
Value()	C/I/F/N	Returns the value of the specified container element leaf element in the C++ language.	57

ContainerElementVector

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

Refer to “C-language and COBOL vectors” on page 21 for detailed descriptions of vector functions.

Vector methods	Description
Deallocate()	Deallocates a container element vector object.
FirstElement()	Returns the first element of the container element vector.
NextElement()	Returns the next element of the container element vector.
Size()	Returns the number of elements in the container element vector.

ControlConnectorInstance

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

Refer to “Basic API calls” on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a control connector instance object.	77
Copy()	Allocates and initializes the storage for a control connector instance object by copying.	81
Deallocate()	Deallocates the storage for a control connector instance object.	82
destructor()	Destructs a control connector instance object.	82

Control connector instance

Basic methods	Description	Page
Equal()	Compares two control connector instance objects on the basis of their source and target activity instances.	79
IsEmpty()	Indicates whether no control connector instance information is available.	83
Kind()	States the kind of the control connector instance, whether it is a transition condition or the "otherwise" connector.	83, 103
operator=()	Assigns a control connector instance object to this one.	79
operator==()	Compares two control connector instance objects on the basis of their source and target activity instances.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
BendPoints()	M	Returns the bend points of the control connector instance.	118
Name()	C	Returns the name associated with the control connector instance.	117
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.	117
PersistentOidOfTargetActivity()	C	Returns the object ID of the activity instance which is the target of this control connector instance.	117
State()	E	Returns the state of the control connector instance, whether it is evaluated, and the result of evaluation.	96, 103
TransitionCondition()	C	Returns the transition condition of the control connector instance.	117
TransitionConditionIsNull()	B	Indicates whether a transition condition is set.	121

ControlConnectorInstanceVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates a control connector instance vector object.

Accessor methods	Description
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

An FmcjDateTime object represents date and time values in the C++ language. An FmcjCDateTime structure represents date and time values in the C-language.

Date/time values are expressed in local time.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls. Following methods are only available in the C++ language.

Basic methods	Description	Page
constructor()	Constructs a date/time object.	77
destructor()	Destructs a date/time object.	82
operator=()	Assigns a date/time object to another one.	79
operator==()	Compares two date/time objects.	79
operator string()	Returns the string representation of the date/time object.	117
IsEmpty()	Indicates whether no date/time information is available.	83

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. Because a date/time object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

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

Following methods are only available in the C++ language.

Accessor methods	Type	Description	Page
Day()	I	Returns the day of the date/time object.	116
Hour()	I	Returns the hours of the date/time object.	116
Minute()	I	Returns the minutes of the date/time object.	116
Month()	I	Returns the month of the date/time object.	116
Second()	I	Returns the sceonds of the date/time object.	116
Year()	I	Returns the year of the date/time object.	116

Date/time

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.	117
FmcjDateTimeCurrentTime	D	Returns the current date/time.	95
FmcjDateTimeIsValid	B	Indicates whether the passed date/time is a valid date/time; must be greater or equal 1970-1-1 12:00 (yyyy-mm-dd hh:mm).	94

DllOptions

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a DLL options object.	77
Copy()	Allocates and initializes the storage for a DLL options object by copying.	81
Deallocate()	Deallocates the storage for a DLL options object.	82
destructor()	Destructs a DLL options object.	82
IsEmpty()	Indicates whether no DLL options information is available.	83
operator=()	Assigns a DLL options object to this one.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
EntryPointName()	C	Returns the name of the entry point of the DLL.	117
ExecuteFenced()	B	States whether the DLL should run in a separate address space.	94
ExecuteFencedIsNull()	B	Indicates whether execute fended is set.	121
KeepLoaded()	B	States whether the DLL should stay loaded.	94
KeepLoadedIsNull()	B	Indicates whether keep loaded is set.	121
PathAndFileName()	C	Returns the path and file name of the DLL.	117

ExecutionAgent/FmcjPEA

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

Refer to "Activity implementation API calls" on page 133 for detailed descriptions of activity implementation API calls.

Activity implementation methods	Description	Page
InContainer()	Accesses the input container from within an activity implementation in Java; for non-Java see the Container.	340
OutContainer()	Accesses the output container from within an activity implementation in Java; for non-Java see the Container.	341
PersistentOidOfActivityInstance()	Returns the object ID of the associated activity instance.	134
UserID()	Returns the user identification on whose behalf the activity implementation was started.	134
SetOutContainer()	Sets the output container from within an activity implementation in Java; for non-Java see the Container.	343

ExecutionData

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an execution data object.	77
Copy()	Allocates and initializes the storage for an execution data object by copying.	81
Deallocate()	Deallocates the storage for an execution data object.	82
destructor()	Destructs an execution data object.	82
IsEmpty()	Indicates whether no execution data information is available.	83
Kind()	Returns the kind of the data, whether it is describing a work item creation, deletion, and so on.	83, 104
operator=()	Assigns an execution data object to this one.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
ActivityInstanceNotificationFromData()	P	Creates an activity instance notification from the execution data.	120
ErrorFromData()	P	Creates an error description object from the execution data.	120

Execution data

Accessor methods	Type	Description	Page
IsError()	B	States whether the execution data describes an erroneous situation.	94
PersistentOid()	C	Returns a representation of the object ID of the object described by the execution data.	117
ProcessInstanceFromData()	P	Creates a process instance from the execution data.	120
ProcessInstanceNotificationFromData()	P	Creates a process instance notificationfrom 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.	117
UserContextIsNull()	B	States whether a user context had been specified.	121

ExecutionService

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
Allocate()	Allocates the storage for an execution service object. The execution server to connect to is taken from the MQ Workflow user's or configuration profile in the currently set configuration.	77
AllocateForSystem()	Allocates the storage for the specified execution service object. The execution server to connect to is taken from the specified parameters in the currently set configuration.	77
AllocateForGroup()	Allocates the storage for the specified execution service object. The execution server to connect to can be any system within the specified system group in the currently set configuration.	77
constructor()	Constructs an execution service object.	77
Copy()	Allocates and initializes the storage for an execution service object by copying.	81
Deallocate()	Deallocates the storage for an execution service object.	82
destructor()	Destructs an execution service object.	82
Equal()	Compares two execution service objects if they represent the same session.	79
operator=()	Assigns an execution service object to this one.	79

Basic methods	Description	Page
operator==()	Compares two execution service objects if they represent the same session.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessorAPI calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
PersistentActivityInstance()	P	Constructs a transient activity instance object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentActivityInstanceNotification()	P	Constructs a transient activity instance notification object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentInstanceMonitor()	P	Constructs a transient instance monitor object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentPerson()	P	Constructs a transient person object representing the persistent object identified by the passed user identification; does not contact the server.	120
PersistentProcessInstance()	P	Constructs a transient process instance object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentProcessInstanceList()	P	Constructs a transient process instance list object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentProcessInstanceNotification()	P	Constructs a transient process instance notification object representing the persistent object identified by the passed object identification; does not contact the server.	120

Execution service

Accessor methods	Type	Description	Page
PersistentProcessTemplate()	P	Constructs a transient process template object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentProcessTemplateList()	P	Constructs a transient process template list object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentWorkitem()	P	Constructs a transient work item object representing the persistent object identified by the passed object identification; does not contact the server.	120
PersistentWorklist()	P	Constructs a transient worklist object representing the persistent object identified by the passed object identification; does not contact the server.	120
ProgramDataFromStream()	P	Constructs a transient program data object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	120
ProgramTemplateFromStream()	P	Constructs a transient program template object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	120
ReadOnlyContainerFromStream()	P	Constructs a transient read-only container object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	120
ReadWriteContainerFromStream()	P	Constructs a transient read/write container object from the stream passed. The specified stream length must be equal or greater to the length returned by the StreamLength() API call when the stream was retrieved. Otherwise, the result of calling this method is undetermined.	120

Accessor methods	Type	Description	Page
SessionID()	C	Returns a string representation of the session.	117
SetSessionContext()	C	Associates the execution service object with the session identified by the passed session identification.	123

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
CreateProcessInstanceList()	Creates a new process instance list on the execution server.	346
CreateProcessTemplateList()	Creates a new process template list on the execution server.	352
CreateWorklist()	Creates a new worklist on the execution server.	357
Logoff()	Logs off from the connected execution server.	365
Logon(), logon()	Logs on to the execution server.	367
LogonWithCredentials()	Logs on to the execution server in C and COBOL by passing user credentials.	367
logon2()	Logs on to the execution server in Java and provides additional parameters.	367
logon3()	Logs on to the execution server in Java by passing user credentials.	367
logon4()	Logs on to the execution server in Java by passing user credentials and additional parameters.	367
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications the logged-on user has access to.	375
QueryItems()	Retrieves the work items or notifications the logged-on user has access to.	381
QueryProcessInstanceLists()	Retrieves the process instance lists the logged-on user has access to.	388
QueryProcessInstanceNotifications()	Retrieves the process instance notifications the logged-on user has access to.	390
QueryProcessInstances()	Retrieves the process instances the logged-on user has access to.	395
QueryProcessTemplateLists()	Retrieves the process template lists the logged-on user has access to.	400
QueryProcessTemplates()	Retrieves the process templates the logged-on user has access to.	402
QueryWorkitems()	Retrieves the work items the logged-on user has access to.	407
QueryWorklists()	Retrieves the worklists the logged-on user has access to.	413

Execution service

Action methods	Description	Page
Receive()	Receives execution data sent by an MQ Workflow execution server.	415
SetPersonAbsent()	Sets the specified person absent or not absent.	418
TerminateReceive()	Places information in the client input queue to indicate that receiving execution data sent by an MQ Workflow execution server can end.	420

Refer to "Activity implementation API calls" on page 133 for detailed descriptions of activity implementation API calls.

Activity implementation methods	Description	Page
Passthrough()	Establishes a session between an activity implementation and an execution server.	373

ExeOptions

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an EXE options object.	77
Copy()	Allocates and initializes the storage for an EXE options object by copying.	81
Deallocate()	Deallocates the storage for an EXE options object.	82
destructor()	Destructs an EXE options object.	82
operator=()	Assigns an EXE options object to this one.	79
IsEmpty()	Indicates whether no EXE options information is available.	83

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
AutomaticClose()	B	States whether the window in which the EXE starts should close when the EXE ends.	94
AutomaticCloseIsNull()	B	Indicates whether automatic close is set.	121
Environment()	C	States the environment settings for the EXE.	117

Accessor methods	Type	Description	Page
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.	94
PathAndFileName()	C	Returns the path and file name of the EXE.	117
RunInXTerm()	B	States whether the EXE should start in a separate xterm.	94
RunInXTermIsNull()	B	Indicates whether run in xterm is set.	121
StartInForeGround()	B	States whether the EXE should start in the foreground.	94
StartInForeGroundIsNull()	B	Indicates whether start in foreground is set.	121
WindowStyle()	E	States the initial window style.	96, 106
WindowStyleIsNull()	B	Indicates whether a window style is set.	121
WorkingDirectoryName()	C	States the working directory for the EXE.	117
WorkingDirectoryNameIsNull()	B	Indicates whether a working directory is set.	121

ExternalOptions

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

Refer to “Basic API calls” on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an External options object.	77
Copy()	Allocates and initializes the storage for an External options object by copying.	81
Deallocate()	Deallocates the storage for an External options object.	82
destructor()	Destructs an External options object.	82
operator=()	Assigns an External options object to this one.	79
IsEmpty()	Indicates whether no External options information is available.	83

Refer to “Accessor/mutator API calls” on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

External options

Accessor methods	Type	Description	Page
BackwardMappingFormat()	C	Specifies the format of the mapping from the structure the executable uses to an MQ Workflow container.	117
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.	116
CodePageIsNull()	B	Indicates whether a code page is set.	121
ExecutableName()	C	Specifies the executable to be invoked by the invocation type and service.	117
ExecutableType()	C	Identifes the type of the executable.	117
ForwardMappingFormat()	C	Specifies the format for the mapping from an MQ Workflow container to the structure the executable uses.	117
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
InvocationType()	C	Specifies the invocation mechanism to invoke the executable on the service.	117
IsLocalUser()	B	Returns whether a local user is to be resolved instead of using the MQ Workflow user ID.	94
IsMappingRoutineCall()	B	Specifies whether forward and backward mapping routines are to be called.	94
IsSecurityRoutineCall()	B	Specifies whether a security routine is to be called.	94
MappingType()	C	Identifies the type of mapping that should occur.	117
MappingTypeIsNull()	B	Indicates whether a mapping type is set.	121
ServiceName()	C	Identifies the service that is to be called.	117
ServiceType()	C	Identifies the type of service to be called, for example, CICS(R) or IMS(TM).	117

Accessor methods	Type	Description	Page
TimeoutPeriod()	E	Specifies how long the program execution agent should wait for a response from the started service, forever, a time period, or never.	96, 107
TimeoutInterval()	I	Specifies the timeout interval in seconds.	116
TimeoutIntervalIsNull()	B	Indicates whether a timeout interval is set.	121

FmcError/FmcjError

An FmcError or FmcjError object represents a description of the reason why a work item or activity instance is in state InError. It also describes an error returned as an asynchronous response.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an Error object.	77
Copy()	Allocates and initializes the storage for an Error object by copying.	81
Deallocate()	Deallocates the storage for an Error object.	82
destructor()	Destructs an Error object.	82
Equal()	Compares two Error objects on the basis of their return codes and parameters.	79
IsEmpty()	Indicates whether no Error information is available.	83
operator=()	Assigns an Error object to this one.	79
operator==()	Compares two Error objects on the basis of their return codes and parameters.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
MessageText()	C	Returns the error as an NLS regarding formatted message.	117
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.	116

FmcException

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

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
MessageText()	C	Returns the exception as an NLS regarding formatted message.	117
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.	117
origin()	C	Returns the module that threw the exception.	117
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.	116

Global

An API global object serves to group global MQ Workflow API API calls.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description
Connect()	Initializes the API in the current thread; not supported in Java.
Disconnect()	Deinitializes the API in the current thread; not supported in Java.

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
ConfigurationID()	C	Returns the configuration ID to be used for profile access.	117
SetConfigurationID()	C	Sets the configuration ID to be used for profile access. Can only be set before the first profile usage; normally the Logon(). Only one configuration can be used per application process.	117

ImplementationData

An implementation data object represents the program implementation definitions.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an implementation data object.	77
Copy()	Allocates and initializes the storage for an implementation data object by copying.	81
Deallocate()	Deallocates the storage for an implementation data object.	82
destructor()	Destructs an implementation data object.	82
operator=()	Assigns an implementation data object to this one.	79
IsEmpty()	Indicates whether no implementation data information is available.	83
Kind()	States the actual kind of the implementation data, whether it is a DLL or an EXE.	83, 109

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
CommandLineParameters()	C	Returns the command line parameters to be passed to the invoked program.	117
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
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

Implementation data

Accessor methods	Type	Description	Page
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.	96, 107

ImplementationDataVector

An implementation data vector represents the result of a query for implementation data in the C-language.

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

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

InstanceMonitor

An instance monitor object represents a monitor in the API. It can be the monitor of a process instance, a monitor of an activity instance of type *Block*, or a monitor of an activity instance of type *Process*.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an instance monitor object.	77
Copy()	Allocates and initializes the storage for an instance monitor object by copying.	81
Deallocate()	Deallocates the storage for an instance monitor object.	82
destructor()	Destructs an instance monitor object.	82
Equal()	Compares two instance monitor objects.	79
operator=()	Assigns an instance monitor object to this one.	79
operator==()	Compares two instance monitor objects.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary.

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

Accessor methods	Type	Description	Page
ActivityInstances()	M	Returns the activity instances which are represented by the instance monitor. The activity instances contain both primary and secondary values.	118
ControlConnectorInstances()	M	Returns the control connector instances which are represented by the instance monitor.	118
PersistentOid()	P/C	Returns a representation of the object identification of the instance monitor.	117

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
ObtainBlockMonitor()	Returns the instance monitor for an activity instance of type *Block*. The activity instance is part of the set of activity instances represented by the instance monitor.	422
ObtainProcessMonitor()	Returns the instance monitor for an activity instance of type *Process*. The activity instance is part of the set of activity instances represented by the instance monitor.	422
Refresh()	Refreshes the instance monitor from the MQ Workflow execution server.	425

Item

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs an item object.	77
Copy()	Allocates and initializes the storage for an item object by copying.	81
Deallocate()	Deallocates the storage for an item object.	82
destructor()	Destructs an item object.	82
Equal()	Compares two items.	79
IsComplete()	Indicates whether the complete item information is available.	82
IsEmpty()	Indicates whether no item information is available.	83
Kind()	States the actual kind of the item, whether it is a work item or some kind of notification.	83, 112
operator=()	Assigns an item to this one.	79
operator==()	Compares two items.	79

Item

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor and mutator API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific item. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
Category()	P/C	Returns the process category of the item.	117
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	P/D	Returns the creation time of the item.	95
Description()	P/C	Returns the description of the item.	117
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the item.	117
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the ending time of the item.	95
EndTimeIsNull()	S/B	Indicates whether an end time is set.	121
Icon()	P/C	Returns the icon associated with the item.	117
InContainerName()	S/C	Returns the name of the input container of the item.	117
LastModificationTime()	P/D	Returns the last time a primary attribute of the item was changed.	95
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.	117
OutContainerName()	S/C	Returns the name of the output container of the item.	117
Owner()	P/C	Returns the user ID of the owner of the item.	117

Accessor methods	Set/ Type	Description	Page
PersistentOid()	P/C	Returns a representation of the object identification of the item.	117
PersistentOidOfProcessInstance()	P/C	Returns the object ID of the associated process instance.	117
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the item.	117
ProcessInstanceName()	P/C	Returns the name of the process instance the item is part of.	117
ProcessInstanceState()	P/E	Returns the state of the process instance the item is part of.	96, 114
ProcessInstanceSystemGroupName()	S/C	Returns the name of the system group of the process instance the item is part of.	117
ProcessInstanceSystemName()	S/C	Returns the name of the system of the process instance the item is part of.	117
ReceivedAs()	P/E	Returns the reason why the item was received.	96, 97
ReceivedTime()	P/D	Returns the time when the item was received by the current owner.	95
StartTime()	S/D	Returns the start time of the item.	95
StartTimeIsNull()	S/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 API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes an item.	427
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance the item is part of.	429
ProcessInstance()	Retrieves the process instance the item is part of.	432
Refresh()	Retrieves the complete information of the item.	434
SetDescription()	Sets the description of the item.	436
SetName()	Sets the name of the item.	438
Transfer()	Transfers an item to the specified user.	440

ItemVector

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

Refer to “C-language and COBOL vectors” on page 21 for detailed descriptions of vector access functions.

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

Message

A message object serves to access the MQ Workflow provided message catalog.

Refer to “Accessor/mutator API calls” on page 92 for detailed descriptions of accessor API calls.

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

Accessor methods	Type	Description	Page
MessageText()	C	Returns an NLS regarding formatted message based on the message ID. Any parameters passed will be incorporated.	117

PersistentList

A persistent list represents a persistent list definition. **All API calls of a persistent list are also applicable to process instance lists, process template lists, and worklists.**

Refer to “Basic API calls” on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
IsEmpty()	Indicates whether no persistent list information is available.	83

Refer to “Accessor/mutator API calls” on page 92 for detailed descriptions of accessorAPI calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
Description()	C	Returns the description of the persistent list.	117

Accessor methods	Type	Description	Page
DescriptionIsNull()	B	Indicates whether a description is set.	121
Filter()	C	Returns the filter of the persistent list.	117
FilterIsNull()	B	Indicates whether a filter is set.	121
Name()	C	Returns the name of the persistent list.	117
OwnerOfList()	C	Returns the user ID of the owner of the persistent list.	117
OwnerOfListIsNull()	B	Indicates whether an owner is set; a public list does not have an owner.	121
PersistentOid()	C	Returns a representation of the object identification of the persistent list.	117
SortCriteria()	C	Returns the sort criteria of the persistent list.	117
SortCriteriaIsNull()	B	Indicates whether sort criteria are set.	121
Threshold()	I	Returns the threshold of the persistent list.	116
ThresholdIsNull()	B	Indicates whether a threshold is set.	121
Type()	E	Returns the type of the persistent list, whether it is a public or private list.	96, 113

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes the persistent list.	443
Refresh()	Refreshes the persistent list.	445
SetDescription()	Sets the description of the persistent list.	447
SetFilter()	Sets the filter of the persistent list.	449
SetSortCriteria()	Sets the sort criteria of the persistent list.	451
SetThreshold()	Sets the threshold of the persistent list.	453

Person

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a person object.	77
Copy()	Allocates and initializes the storage for a person object by copying.	81
Deallocate()	Deallocates the storage for a person object.	82
destructor()	Destructs a person object.	82
Equal()	Compares two persons.	79
operator=()	Assigns a person to this one.	79
operator==()	Compares two persons.	79
IsComplete()	Indicates whether the complete person information is available.	82

Person

Basic methods	Description	Page
IsEmpty()	Indicates whether no person information is available.	83

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when persons are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific person. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
CategoriesAuthorizedFor()	P/M	Returns the categories the person is authorized for with basic or with administration rights. If the person is authorized for all categories as administrator, no category is returned here. If the person is authorized for all categories with basic rights, categories authorized with administration rights are returned here.	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.	117
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
FirstName()	P/C	Returns the first name of the person.	117
FirstNameIsNull()	P/B	Indicates whether a first name is set.	121
IsAbsent()	P/B	Indicates whether the person is absent.	94

Accessor methods	Set/ Type	Description	Page
IsAdminForCategory()	P/B	Indicates whether the person has administrator rights for the specified category. Returns false if the category does not exist. If the person is authorized for all categories as administrator, then true is returned independent of the category existence.	94
IsAdministrator()	S/B	Indicates whether the person is an administrator.	94
IsAuthorizedForAllCategories()	P/B	Indicates whether the person is said to be authorized for all categories either with basic and/or administration rights.	94
IsAuthorizedForAllCategoriesAsAdmin()	P/B	Indicates whether the person is said to be authorized for all categories as administrator.	94
IsAuthorizedForAllPersons()	P/B	Indicates whether the person is authorized to see the items of all persons.	94
IsAuthorizedForAuthorizationDefinition()	P/B	Indicates whether the person is authorized to define authorizations.	94
IsAuthorizedForOperationAdministration()	P/B	Indicates whether the person is authorized for operational administrations.	94
IsAuthorizedForProcessDefinition()	P/B	Indicates whether the person is authorized to define process models.	94
IsAuthorizedForStaffDefinition()	P/B	Indicates whether the person is authorized to define persons.	94
IsAuthorizedForTopologyDefinition()	P/B	Indicates whether the person is authorized to define topological data.	94
IsManager()	S/B	Indicates whether the person is a manager.	94
IsResetAbsence()	P/B	Indicates whether the absence flag should be reset when the person logs on.	94
LastName()	P/C	Returns the last name of the person.	117
LastNameIsNull()	P/B	Indicates whether a last name is set.	121
Level()	P/I	Returns the level of the person.	116

Person

Accessor methods	Set/ Type	Description	Page
Manager()	S/C	Returns the user identification of the person's manager.	117
ManagerIsNull()	S/B	Indicates whether the person's manager is set.	121
MiddleName()	P/C	Returns the middle name of the person.	117
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.	117
OrganizationNameIsNull()	P/B	Indicates whether an organization name is set.	121
PersonID()	P/C	Returns the person ID of the person.	117
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.	117
PhoneIsNull()	P/B	Indicates whether a phone is set.	121
SecondPhone()	P/C	Returns the alternate phone number of the person.	117
SecondPhoneIsNull()	P/B	Indicates whether an alternate phone is set.	121
Substitute()	P/C	Returns the substitute of the person.	117
SubstituteIsNull()	P/B	Indicates whether a substitute is set.	121

Accessor methods	Set/ Type	Description	Page
SystemName()	P/C	Returns the home system of the person.	117
UserID()	P/C	Returns the user identification of the person.	117

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Refresh()	Retrieves the complete person information from the server.	455
SetAbsence()	Sets the absent flag of the logged-on user to the specified value.	457
SetSubstitute()	Sets the substitute of the logged-on user to the specified value.	459

Point

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a point object.	77
Copy()	Allocates and initializes the storage for a point object by copying.	81
Deallocate()	Deallocates the storage for a point object.	82
destructor()	Destructs a point object.	82
Equal()	Compares two point objects on the basis of their contents.	79
IsEmpty()	Indicates whether no point information is available.	83
operator=()	Assigns a point object to this one.	79
operator==()	Compares two point objects on the basis of their contents.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
XPosition()	I	Returns the x-coordinate of the point.	116
YPosition()	I	Returns the y-coordinate of the point.	116

PointVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

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

ProcessInstance

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance object.	77
Copy()	Allocates and initializes the storage for a process instance object by copying.	81
Deallocate()	Deallocates the storage for a process instance object.	82
destructor()	Destructs a process instance object.	82
Equal()	Compares two process instances.	79
IsComplete()	Indicates whether the complete process instance information is available.	82
IsEmpty()	Indicates whether no process instance information is available.	83
operator=()	Assigns a process instance to this one.	79
operator==()	Compares two process instances.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific process instance. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
AuditMode()	S/E	Returns the audit mode of the process instance.	96, 98
Category()	P/C	Returns the category of the process instance.	117
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	S/D	Returns the creation time of the process instance.	95
Creator()	S/C	Returns the creator of the process instance.	117
Description()	P/C	Returns the description of the process instance.	117
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the process instance.	117
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
EndTime()	S/D	Returns the end time of the process instance.	95
EndTimeIsNull()	S/B	Indicates whether an end time is set.	121
Icon()	P/C	Returns the icon associated with the process instance.	117
InContainerName()	S/C	Returns the name of the input container of the process instance.	117
InContainerNeeded()	P/B	Indicates whether an input container is needed to start the process instance. An input container is needed when • There is a mapping to some other container. • Staff assignment data is taken from it. • Notification related data is taken from it. • A transition or exit condition refers to a container element. • A description refers to a container element. • *Prompt for data at process start* is set for the process model.	94
LastModificationTime()	P/D	Returns the last time a primary attribute of the process instance was changed.	95
LastStateChangeTime()	P/D	Returns the last time the state of the process instance was changed.	95
Name()	P/C	Returns the name of the process instance.	117

Process instance

Accessor methods	Set/ Type	Description	Page
NotificationTime()	S/D	Returns the notification time of the process instance.	95
NotificationTimeIsNull()	S/B	Indicates whether a notification time is set.	121
NotifiedPerson()	S/C	Returns the person who received the notification.	117
NotifiedPersonIsNull()	S/B	Indicates whether a notified person is set.	121
OrganizationName()	S/C	Returns the name of the organization of the process instance.	117
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.	117
ParentName()	P/C	Returns the name of the parent process instance of this process instance.	117
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.	117
PersistentOidOfProcessTemplate()	P/C	Returns a representation of the object identification of the process template the process instance is derived from.	117
ProcessAdmin()	S/C	Returns the user ID of the process administrator of the process instance.	117
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.	117
RoleName()	S/C	Returns the name of the role of the process instance.	117
RoleNameIsNull()	S/B	Indicates whether a role is set.	121
Starter()	S/C	Returns the starter of the process instance.	117
StarterIsNull()	S/B	Indicates whether a starter is set.	121
StartTime()	S/D	Returns the start time of the process instance.	95
StartTimeIsNull()	S/B	Indicates whether a start time is set.	121
State()	P/E	Returns the state of the process instance.	96, 114

Accessor methods	Set/ Type	Description	Page
StateOfNotification()	S/E	Returns the notification state of the process instance.	96, 113
SuspensionExpirationTime()	P/D	Returns the suspension expiration time of the process instance.	95
SuspensionExpirationTimeIsNull()	P/B	Indicates whether the suspension expiration time is set.	121
SuspensionTime()	P/D	Returns the time the process instance was suspended.	95
SuspensionTimeIsNull()	P/B	Indicates whether the suspension time is set.	121
SystemGroupName()	P/C	Returns the name of the system group where the process instance runs.	117
SystemName()	P/C	Returns the name of the system where the process instance runs.	117
TopLevelName()	P/C	Returns the name of the top level process instance of this process instance.	117

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Delete()	Deletes the process instance.	462
InContainer()	Retrieves the input container of the process instance.	464
ObtainProcessMonitor()	Retrieves the instance monitor for the process instance.	466
OutContainer()	Retrieves the output container of the process instance.	468
Refresh()	Retrieves the complete information of the process instance.	471
Restart()	Restarts the process instance.	473
Resume()	Resumes the execution of a suspended process instance.	474
SetDescription()	Sets the description of the process instance.	476
SetName()	Sets the name of the process instance.	478
Start()	Starts the process instance.	481
Start2()	Starts the process instance in Java and provides an input container.	481
Suspend()	Suspends the process instance.	483
Suspend2()	Suspends the process instance in Java until the specified calendar date.	483
SuspendUntil()	Suspends the process instance until the specified time.	483
Terminate()	Terminates the process instance.	485

ProcessInstanceList

A process instance list represents a group of process instances. **All API calls of a persistent list are also applicable to process instance lists.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance list object.	77
Copy()	Allocates and initializes the storage for a process instance list object by copying.	81
Deallocate()	Deallocates the storage for a process instance list object.	82
destructor()	Destructs a process instance list object.	82
Equal()	Compares two process instance lists.	79
operator=()	Assigns a process instance list to this one.	79
operator==()	Compares two process instance lists.	79

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryProcessInstances()	Retrieves the process instances qualifying via the process instance list.	488

ProcessInstanceListVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

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

ProcessInstanceNotification

A process instance notification represents a notification raised for a process instance. **All API calls of an FmcjItem are also applicable to process instance notifications.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process instance notification object.	77

Basic methods	Description	Page
Copy()	Allocates and initializes the storage for a process instance notification object by copying.	81
Deallocate()	Deallocates the storage for a process instance notification object.	82
destructor()	Destructs a process instance notification object.	82
Kind()	In the C++ language, states that the object is a process instance notification.	83, 112
operator=()	Assigns a process instance notification to this one.	79
operator==()	Compares two process instance notifications.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process instances are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific process instance. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
NotificationTime()	S/D	Returns the notification time of the process instance.	95

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page

ProcessInstanceNotificationVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates a process instance notification vector object.
FirstElement()	Returns the first element of the process instance notification vector.

Process instance notification vector

Accessor methods	Description
NextElement()	Returns the next element of the process instance notification vector.
Size()	Returns the number of elements in the process instance notification vector.

ProcessInstanceVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates the storage for a process instance vector object.
FirstElement()	Returns the first element of the process instance vector.
NextElement()	Returns the next element of the process instance vector.
Size()	Returns the number of elements in the process instance vector.

ProcessTemplate

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process template object.	77
Copy()	Allocates and initializes the storage for a process template object by copying.	81
Deallocate()	Deallocates the storage for a process template object.	82
destructor()	Destructs a process template object.	82
Equal()	Compares two process templates.	79
IsComplete()	Indicates whether the complete process template information is available.	82
IsEmpty()	Indicates whether no process template information is available.	83
operator=()	Assigns a process template to this one.	79
operator==()	Compares two process templates.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when process templates are queried or if this attribute

is a secondary attribute (S) and set only after the refresh of a specific process template. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
AuditMode()	S/E	Returns the audit mode of the process template.	96, 98
Category()	P/C	Returns the category of the process template.	117
CategoryIsNull()	P/B	Indicates whether a category is set.	121
CreationTime()	P/D	Returns the creation time of the process template.	95
Description()	P/C	Returns the description of the process template.	117
DescriptionIsNull()	P/B	Indicates whether a description is set.	121
Documentation()	S/C	Returns the documentation of the process template.	117
DocumentationIsNull()	S/B	Indicates whether a documentation is set.	121
Icon()	P/C	Returns the icon associated with the process template.	117
InContainerName()	S/C	Returns the name of the input container of the process template.	117
InContainerNeeded()	P/B	Indicates whether an input container is needed to start an instance of the process template. An input container is needed when • There is a mapping to some other container. • Staff assignment data is taken from it. • Notification related data is taken from it. • A transaction or exit condition refers to a container element. • A description refers to a container element. • *Prompt for data at process start* is set for the process model.	94
LastModificationTime()	P/D	Returns the last time a primary attribute of the process template was changed.	95
Name()	P/C	Returns the name of the process template.	117
OrganizationName()	S/C	Returns the name of the organization of the process template.	117
OrganizationNameIsNull()	S/B	Indicates whether an organization name is set.	121

Process template

Accessor methods	Set/ Type	Description	Page
OutContainerName()	S/C	Returns the name of the output container of the process template.	117
PersistentOid()	P/C	Returns a representation of the object identification of the process template.	117
ProcessAdmin()	S/C	Returns the user ID of the process administrator of an instance of the process template.	117
ProcessAdminIsNull()	S/B	Indicates whether a process administrator is set.	121
RoleName()	S/C	Returns the name of the role of the process template.	117
RoleNameIsNull()	S/B	Indicates whether a role is set.	121
ValidFromTime()	P/D	Returns the time when the process template becomes valid.	95

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
CreateAndStartInstance()	Creates and starts an instance of the process template.	491
CreateAndStartInstance2()	Creates and starts an instance of the process template in Java and provides an input container.	491
CreateInstance()	Creates an instance of the process template.	496
Delete()	Deletes the specified process template.	499
Delete2()	Deletes the specified process template versions in Java.	499
ExecuteProcessInstance()	Creates and executes an instance from the specified process template.	502
ExecuteProcessInstanceAsync()	Creates and executes an instance from the specified process template; an answer is not waited for.	502
InitialInContainer()	Retrieves the initially defined input container of the process template.	509
ProgramTemplate()	Retrieves the program template specified by the passed name.	511
Refresh()	Retrieves the complete information of the process template.	514

ProcessTemplateList

A process template list represents a group of process templates. **All API calls of a persistent list are also applicable to process template lists.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a process template list object.	77
Copy()	Allocates and initializes the storage for a process template list object by copying.	81
Deallocate()	Deallocates the storage for a process template list object.	82
Equal()	Compares two process template lists.	79
destructor()	Destructs a process template list object.	82
operator=()	Assigns a process template list to this one.	79
operator==()	Compares two process template lists.	79

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryProcessTemplates()	Retrieves the process templates qualifying via the process template list.	516

ProcessTemplateListVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates a process template list vector object.
FirstElement()	Returns the first element of the process template list vector.
NextElement()	Returns the next element of the process template list vector.
Size()	Returns the number of elements in the process template list vector.

ProcessTemplateVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates the storage for a process template vector object.
FirstElement()	Returns the first element of the process template vector.
NextElement()	Returns the next element of the process template vector.

Process template vector

Accessor methods	Description
Size()	Returns the number of elements in the process template vector.

ProgramData

A program data object represents the program implementation definitions. In C++, it privately inherits from program template.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a program data object.	77
Copy()	Allocates and initializes the storage for a program data object by copying.	81
Deallocate()	Deallocates the storage for a program data object.	82
destructor()	Destructs a program data object.	82
Equal()	Compares two program data objects if they belong to the same work item.	79
IsEmpty()	Indicates whether no program data information is available yet.	83
operator=()	Assigns a program data object to this one.	79
operator==()	Compares two program data objects if they belong to the same work item.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
AsStream()	C	Returns the program data as a binary stream.	117
Description()	C	Returns the description of the implementing program.	117
DescriptionIsNull()	B	Indicates whether a description is set.	121
ExecutionMode()	E	States whether the program can participate in global transactions or not.	96, 105
ExecutionUser()	E	Returns the user on whose behalf the program is to be executed.	96, 106
Icon()	C	Returns the icon associated with the implementing program.	117
Implementations()	M	Returns the implementation definitions of the program.	118
InContainer()	P	Returns the input container of the program.	120

Accessor methods	Type	Description	Page
IsUnattended()	B	States whether the program can run unattended.	94
OutContainer()	P	Returns the output container of the program.	120
ProgramTrusted()	B	States whether the program can be trusted. Only a trusted program can receive its program ID.	94
StreamLength()	C	In the C, C++, and COBOL languages, returns the length of the buffer needed to hold the program data in stream format.	117

ProgramTemplate

A program template object represents the program implementation definitions.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a program template object.	77
Copy()	Allocates and initializes the storage for a program template object by copying.	81
Deallocate()	Deallocates the storage for a program template object.	82
destructor()	Destructs a program template object.	82
Equal()	Compares two program template objects on the basis of their names and the process template they belong to.	79
IsEmpty()	Indicates whether no program template information is available yet.	83
operator=()	Assigns a program template object to this one.	79
operator==()	Compares two program template objects on the basis of their names and the process template they belong to.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
AsStream()	C	Returns the program template as a binary stream.	117
Description()	C	Returns the description of the implementing program.	117
DescriptionIsNull()	B	Indicates whether a description is set.	121
ExecutionMode()	E	States whether the program can participate in global transactions or not.	96, 105

Program template

Accessor methods	Type	Description	Page
ExecutionUser()	E	Returns the user on whose behalf the program is to be executed.	117, 106
Icon()	C	Returns the icon associated with the implementing program.	117
Implementations()	M	Returns the implementation definitions of the program.	118
InitialInContainer()	P	Returns the initially defined input container of the program.	120
InContainerAccess()	B	States whether the input container is accessed by the program.	94
IsUnattended()	B	States whether the program can run unattended.	94
InitialOutContainer()	P	Returns the initially defined output container of the program.	120
OutContainerAccess()	B	States whether the output container is accessed by the program.	94
ProgramTrusted()	B	States whether the program can be trusted. Only a trusted program can receive its program ID.	94
StreamLength()	C	In the C, C++, and COBOL languages, returns the length of the buffer needed to hold the program template in stream format.	117
StructuresFromActivity()	B	States whether the program can handle any container passed to it.	94
ValidFromTime()	P/D	Returns the time when the process template and thus the program template becomes valid.	95

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Execute()	Requests the execution of the program by the system's program execution server.	519
Execute2()	Requests the execution of the program by the system's program execution server.	519
ExecuteAsync()	Requests the execution of the program by the system's program execution server; an answer is not waited for.	519
ExecuteAsync2()	Requests the execution of the program by the system's program execution server; an answer is not waited for.	519
ExecuteAsyncWithOptions()	Requests the execution of the program by the system's program execution server; an answer is not waited for. The priority of the program can be specified.	519
ExecuteWithOptions()	Requests the execution of the program by the system's program execution server. The priority of the program can be specified.	519

ReadOnlyContainer

A read-only container represents a container that can only be read, for example, an input data container of a work item or an output container of a process instance. **All API calls of a container are applicable to read-only containers.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
AsReadWriteContainer()	In C, COBOL, and Java, converts the read-only container into a read/write container.	80
constructor()	Constructs a read-only container object.	77
Copy()	Allocates and initializes the storage for a read-only container object by copying.	81
Deallocate()	Deallocates the storage for a read-only container object.	82
Equal()	Compares two read-only containers.	79
destructor()	Destructs a read-only container object.	82
operator=()	Assigns a read-only container to this one.	79
operator==()	Compares two read-only containers.	79
operator FmcjReadWriteContainer()	In C++, converts the read-only container into a read/write container.	80

ReadOnlyContainerHolder

A read-only container holder represents an object in Java that can contain a read-only container, for example, an output container of an executed process instance.

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a read-only container holder object. Optionally, an already existing read-only container can be passed.	77

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

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

Accessor methods	Type	Description	Page
value()	O	Returns the read-only container contained in the holder object.	119

ReadWriteContainer

A read/write container represents a container that can be read and written, for example, an input container of a process instance or an output container of a work item. **All API calls of a container are applicable to read/write containers.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
AsReadOnlyContainer()	In C, COBOL, and Java, converts the read/write container into a read-only container.	80
constructor()	Constructs a read/write container object.	77
Copy()	Allocates and initializes the storage for a read/write container object by copying.	81
Deallocate()	Deallocates the storage for a read/write container object.	82
Equal()	Compares two read/write containers.	79
destructor()	Destructs a read/write container object.	82
operator=()	Assigns a read/write container to another one.	79
operator==()	Compares two read/write containers.	79
operator FmcjReadOnlyContainer()	In C++, converts the read/write container into a read-only container.	80

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

The value in the **Type** column states the type of the property set, whether it is a binary (N), a character string (C), a float (F), or an integer (I). The API call declaration can be found at the indicated page.

Accessor methods	Type	Description	Page
SetArrayBinaryValue()	N	Sets the value of the specified container leaf element in the C-language. The leaf element is part of an array and of type BINARY.	62
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.	62
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.	62
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.	63
SetBinaryValue()	N	Sets the value of the specified container leaf element in the C-language. The leaf element is of type BINARY.	62
SetBuffer()	N	Sets the value of the specified container leaf element in Java. The leaf element is of type BINARY.	64

Accessor methods	Type	Description	Page
SetBuffer2()	N	Sets the value of the specified container leaf element in Java. The leaf element is part of an array and of type BINARY.	64
SetDouble()	F	Sets the value of the specified container leaf element in Java. The leaf element is of type FLOAT.	64
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.	62
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.	62
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.	63
SetValue()	N/F/C/I	Sets the value of the specified container leaf element in the C++ language.	63

Result

A result object represents the result of a API call call in the C++ and C-language.

Refer to “Basic API calls” on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
destructor	Destructs the C++ representation of the result object.	82

Refer to “Accessor/mutator API calls” on page 92 for detailed descriptions of accessor API calls. Because a result object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

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

Result

Accessor methods	Type	Description	Page
MessageText()	C	Returns the result as an NLS regarding formatted message.	117
ObjectOfCurrentThread()	P	Returns the result object associated with the thread from where this API call is called.	119
Origin()	C	Returns the origin of the result, that is, file, line, function.	117
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.	116

Service

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

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. Because a service object represents a supporting object on the client only, the distinction between primary and secondary attributes is not applicable.

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

Accessor methods	Type	Description	Page
IsLoggedOn()	B	Indicates whether a user logged on via this service object. This API call tells you the logon status known by the client. When issuing an action call, the session may, however, be not found because it expired or because you reconstructed a session from a different system group.	94
SetTimeout()	I	Sets the time the client will wait for a server to answer. The time is to be specified in milliseconds.	121
SystemGroupName()	C	Returns the name of the system group where the server resides.	117
SystemName()	C	Returns the name of the system where the server resides.	117
Timeout()	I	Returns the time the client will wait for a server to answer.	116
UserID()	C	Returns the user identification of the logged-on user.	117

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
Refresh()	Refreshes information from the server, especially the logged-on status.	524
SetPassword()	Sets the password of the logged-on user.	526
UserSettings()	Retrieves the user settings of the logged-on user.	528

StringVector

In the C-language, a string vector serves to represents a set of string information. For example, a string vector is returned to show the categories the logged-on user is authorized for. Or, a string vector must be used to specify the persons to stand in for.

Refer to “C-language and COBOL vectors” on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
AddElement()	Adds a string to the string vector.
Allocate()	Allocates the storage for a string vector.
Deallocate()	Deallocates the storage for a string vector.
FirstElement()	Returns the first element of the string vector.
FirstResultParmElement()	Returns the first element of a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read.
NextElement()	Returns the next element of the string vector.
NextResultParmElement()	Returns the next element of a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read.
RemoveElement()	Removes a string from the string vector.
ResultParmDeallocate()	Deallocates the storage for a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read.
ResultParmSize()	Returns the number of elements in a string vector representing the parameters of a result object; calling this function does not change the result object and thus allows for a consistent read.
Size()	Returns the number of elements in the string vector.

SymbolLayout

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

Symbol layout

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a symbol layout object.	77
Copy()	Allocates and initializes the storage for a symbol layout object by copying.	81
Deallocate()	Deallocates the storage for a symbol layout object.	82
destructor()	Destructs a symbol layout object.	82
Equal()	Compares two symbol layout objects on the basis of their contents.	79
IsEmpty()	Indicates whether no symbol layout information is available.	83
operator=()	Assigns a symbol layout object to this one.	79
operator==()	Compares two symbol layout objects on the basis of their contents.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
XPosition()	I	Returns the x-coordinate of the named icon.	116
XPositionOfName()	I	Returns the x-coordinate of the name associated to the icon.	116
YPosition()	I	Returns the y-coordinate of the named icon.	116
YPositionOfName()	I	Returns the y-coordinate of the name associated to the icon.	116

Workitem

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

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a work item object.	77
Copy()	Allocates and initializes the storage for a work item object by copying.	81
Deallocate()	Deallocates the storage for a work item object.	82
destructor()	Destructs a work item object.	82
Kind()	In the C++ language, states that the object is a work item.	83, 112
operator=()	Assigns a work item to this one.	79
operator==()	Compares two work items.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls.

Note: The value in the **Set** column shows if this attribute is a primary attribute (P) and set immediately when work items are queried or if this attribute is a secondary attribute (S) and set only after the refresh of a specific work item. Refreshing can be done explicitly by issuing the Refresh() API call but is also done automatically when a secondary (or primary) attribute is accessed and not yet available in the API cache.

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

Accessor methods	Set/ Type	Description	Page
ActivityKind()	P/E	Returns the kind of the associated activity instance, whether it is a program or process and so on.	83, 102
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.	117
ExpirationTime()	S/D	Returns the expiration time of the work item.	95
ExpirationTimeIsNull()	S/B	Indicates whether an expiration time is set.	121
FirstNotificationTime()	S/D	Returns the time the first notification for the work item is to occur or has occurred.	95
FirstNotificationTimeIsNull()	S/B	Indicates whether a first notification time is set.	121
Implementation()	P/C	Returns the name of the implementing program of the associated activity instance.	117
ImplementationIsNull()	P/B	Indicates whether an implementation is set.	121
ManualExitMode()	S/B	Returns whether the exit mode of the work item is manual.	94
ManualStartMode()	S/B	Returns whether the start mode of the work item is manual.	94
PersistentOidOfActivityInstance()	P/C	Returns the object ID of the associated activity instance.	117
Priority()	P/I	Returns the priority of the work item.	116
SecondNotificationTime()	S/D	Returns the time the second notification for the work item is to occur or has occurred.	95

Work item

Accessor methods	Set/ Type	Description	Page
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.	117
State	P/E	Returns the state of the work item.	96, 109
StateOfNotification()	S/E	Returns the notification state of the work item.	96, 99
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 API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
ActivityInstance()	Retrieves the associated activity instance.	532
CancelCheckOut()	Cancels the check out of the work item.	534
CheckIn()	Checks in the work item.	536
CheckOut()	Checks out the work item.	539
Finish()	Finishes a manual exit work item.	543
ForceFinish()	Force finishes the work item.	545
ForceFinishWithContainer()	Force finishes the work item and passes a container.	317
ForceFinish2()	In Java, force finishes the work item and passes a container.	317
ForceRestart()	Force restarts the work item.	548
ForceRestartWithContainer()	Force restarts the work item and passes a container.	320
ForceRestart2()	In Java, force restarts the work item and passes a container.	320
InContainer()	Retrieves the input container of the work item.	550
OutContainer()	Retrieves the output container of the work item.	552
Restart()	Restarts the work item.	554
Start()	Starts the work item.	556
StartTool()	Starts the specified support tool.	558
Terminate()	Terminates the work item.	560

WorkitemVector

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

Refer to "C-language and COBOL vectors" on page 21 for detailed descriptions of vector access functions.

Accessor methods	Description
Deallocate()	Deallocates the storage for a workitem vector object.
FirstElement()	Returns the first element of the workitem vector.
NextElement()	Returns the next element of the workitem vector.
Size()	Returns the number of elements in the workitem vector.

Worklist

A worklist represents a group of items. **All API calls of a persistent list are also applicable to worklists.**

Refer to "Basic API calls" on page 76 for detailed descriptions of basic API calls.

Basic methods	Description	Page
constructor()	Constructs a worklist object.	77
Copy()	Allocates and initializes the storage for a worklist object by copying.	81
Deallocate()	Deallocates the storage for a worklist object.	82
destructor()	Destructs a worklist object.	82
Equal()	Compares two worklists.	79
operator=()	Assigns a worklist to another one.	79
operator==()	Compares two worklists.	79

Refer to "Accessor/mutator API calls" on page 92 for detailed descriptions of accessor API calls. All properties are primary properties.

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

Accessor methods	Type	Description	Page
BeepOption()	B	Indicates whether a beep should sound when the contents of the worklist changes.	94

Refer to "Action API calls" on page 133 for detailed descriptions of action API calls.

Action methods	Description	Page
QueryActivityInstanceNotifications()	Retrieves the activity instance notifications qualifying via the worklist.	563
QueryItems()	Retrieves all items qualifying via the worklist.	565
QueryProcessInstanceNotifications()	Retrieves the process instance notifications qualifying via the worklist.	568
QueryWorkitems()	Retrieves the work items qualifying via the worklist.	571

WorklistVector

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

Refer to “C-language and COBOL vectors” on page 21 for detailed descriptions of vector access functions.

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

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

Chapter 5. API action and activity implementation calls

The following chapter describes the MQ Workflow action and activity implementation API calls in alphabetical order by class.

Each entry contains a functional description of the API call followed by subsections:

Usage note 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 declarations
Shows the required file declarations and calling sequences.
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.
COBOL signature
Shows the COBOL syntax of the API call.
Parameters Describes each of the parameters together with an indication of whether the parameter is an input or output parameter.
Return type Describes the type of value returned by the call.
Return codes/ FmcException
Lists all possible return codes or exceptions which may issued or raised by this call.
Examples Points to an example of the call.

Activity instance actions

An FmcjActivityInstance or an ActivityInstance object represents an instance of an activity of a process instance. An activity instance is uniquely identified by its object identifier or by its fully qualified name within the process instance. The fully qualified name of an activity instance is a name in dot notation where the hierarchy of nested activities of type *Block* is presented from left to right, and their names are separated by a dot.

The following sections describe the actions which can be applied on an activity instance. See “ActivityInstance” on page 256 for a complete list of API calls.

ForceFinish()

This API call ends the execution of the specified activity instance because it is known to have completed (action call).

An activity instance implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. An activity instance implemented by a process must be in the states *Ready, Executed, InError*, or *Terminated*. The associated process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

Optionally, an output container can be specified to denote the result of processing. If none is specified, the output container available at the execution server is taken. For example, the output container defined with initial values.

The activity instance and, if it exists, the single non-disabled work item are then put into the *ForceFinished* state. The starter is set to the logged-on user. The exit condition is considered to be true and navigation proceeds.

Depending on the "delete finished items" option, associated work items are deleted.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceForceFinish( FmcjActivityInstanceHandle hdlInstance,
                                 FmcjContainerHandle        outputContainer)
```

C++ language signatures

```
APIRET ForceFinish()

APIRET ForceFinish( FmcjContainer const & outputContainer )
```

Java signature

```
public abstract
void   forceFinish() throws FmcException

public abstract
void   forceFinish2( Container outputContainer ) throws FmcException
```

COBOL

```
FmcjAIForceFinish.
   CALL    "FmcjActivityInstanceForceFinish"
                  USING
                  BY VALUE
                     hdlInstance
                     outputContainer
                  RETURNING
                     intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

outputContainer
Input. The output container to become the result of the activity instance execution. A 0 handle can be passed in the C-language.

Return type

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

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

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

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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

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

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.

FMC_ERROR_WRONG_STATE(120)
The activity instance or process instance is in the wrong state.

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

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

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

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

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceRestart()

This API call forces MQ Workflow to enable the restart of the specified activity instance (action call).

An activity instance implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. An activity instance implemented by a process must be in the states *Ready, Executed, InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

Optionally, an input container can be specified to denote the input to be used when the activity instance or its associated work item is (re)started. If none is specified, the input container available at the execution server is taken.

The activity instance and the logged-on user's work item are then reset into the *Ready* state. If there is no work item for the logged-on user, it is created. All other work items associated with the activity instance are set into the *Disabled* state. Note that an automatic activity instance must now be started manually.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceForceRestart(
       FmcjActivityInstanceHandle hdlInstance,
       FmcjContainerHandle        inputContainer   )
```

C++ language signature

```
APIRET ForceRestart()

APIRET ForceRestart( FmcjContainer const & inputContainer )
```

Java signature

```
public abstract
void   forceRestart() throws FmcException

public abstract
void   forceRestart2( Container inputContainer ) throws FmcException
```

COBOL

```
        FmcjAIForceRestart.
           CALL    "FmcjActivityInstanceForceRestart"
                         USING
                         BY VALUE
                            hdlInstance
                            inputContainer
                         RETURNING
                            intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

inputContainer
Input. The input container to be used for restarting the activity instance. A 0 handle can be passed in the C-language.

Return type

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

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

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

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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

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

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.

FMC_ERROR_WRONG_STATE(120)
The activity instance or process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the activity instance from the MQ Workflow execution server (action call).

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceInContainer( FmcjActivityInstanceHandle    hdlInstance
                                 FmcjReadOnlyContainerHandle * input     )
```

C++ language signature

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java signature

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

COBOL

```
        FmcjAIInCtnr.
           CALL    "FmcjActivityInstanceInContainer"
                           USING
                           BY VALUE
                              hdlInstance
                           BY REFERENCE
                              inputValue
                           RETURNING
                              intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

input Input/Output. The input container.

Return type

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

ReadOnlyContainer

The input container.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

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

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)

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

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()

This API call retrieves a monitor for the process instance the activity instance is part of from the MQ Workflow execution server (action call).

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

Note: Deep is currently not supported.

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

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceObtainProcessMonitor(
       FmcjActivityInstanceHandle       hdlInstance,
       bool                             deep,
       FmcjInstanceMonitorHandle *      monitor)
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false ) const
```

Java signature

```
public abstract
InstanceMonitor obtainProcessMonitor( boolean deep )
throws FmcException
```

COBOL

```
        FmcjAIObtainProcMon.
           CALL    "FmcjActivityInstanceObtainProcessMonitor"
                         USING
                         BY VALUE
                            hdlInstance
                            deep
                         BY REFERENCE
                            monitor
                         RETURNING
                            intReturnValue.
```

Parameters

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

hdlInstance Input. The activity instance whose instance monitor for the containing process instance is to be retrieved.

monitor Input/Output. The address of the handle to the instance monitor respectively the instance monitor object to be set.

returnCode Input/Output. The result of calling this method - see return codes below.

Return type

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

InstanceMonitor*/InstanceMonitor

A pointer to the instance monitor respectively the instance monitor for the process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

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

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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

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

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

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

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

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

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

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the activity instance from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceOutContainer( FmcjActivityInstanceHandle    hdlInstance,
                                  FmcjReadOnlyContainerHandle * output     )
```

C++ language signature

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java signature

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

COBOL

```
        FmcjAIOutCtnr.
           CALL    "FmcjActivityInstanceOutContainer"
                         USING
                         BY VALUE
                            hdlInstance
                         BY REFERENCE
                            outputValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance to be dealt with.

output Input/Output. The output container.

Return type

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

ReadOnlyContainer
The output container.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

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

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

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

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

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

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

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

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

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

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the activity instance from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceRefresh( FmcjActivityInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

COBOL

```
        FmcjAIRefresh.
           CALL    "FmcjActivityInstanceRefresh"
                          USING
                          BY VALUE
                             hdlInstance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance object to be refreshed.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The activity instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SubProcessInstance()

This API call retrieves the process instance which is implementing the activity instance from the MQ Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

In C++, when the process instance object to be initialized is not empty, then that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance handle already points to some object.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjAISubProcInst.
           CALL    "FmcjActivityInstanceSubProcessInstance"
                          USING
                          BY VALUE
                             hdlInstance
                          BY REFERENCE
                             instance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance object to be queried.

instance Input/Output. The subprocess instance object to be retrieved (initialized).

returnCode Input/Output. The result of calling this method - see return codes below.

Return type

APIRET

The result of calling this method - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the subprocess instance respectively the subprocess instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The activity instance does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This API call terminates an activity instance implemented by a program or process (action call).

If the activity instance is implemented by a program, it must be in the states *CheckedOut* or *Running* and the process instance must be in the states *Running, Suspending, Suspended*, or *Terminating*. If the activity instance is implemented by a process, it must be in the states *Running, Suspending*, or *Suspended* and the process instance must be in the states *Running, Suspending, Suspended*, or *Terminating*.

When the activity instance is implemented by a program and processed under the control of a program execution agent or user-defined program execution server, a message is sent to inform about the termination request. The program execution agent tries to kill fenced activity implementations.

An activity instance implemented by a process is terminated together with all its non-autonomous subprocesses with respect to control autonomy.

The activity instance is then put into the *Terminating* or *Terminated* state.

When the *Terminated* state has been reached, the exit condition is considered to be false, the output container and especially the return code (_RC) are not set, and navigation ends. Navigation can be explicitly continued by a user with process administration rights, that is, ForceFinish() or ForceRestart() repair actions can be called.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process administration authorization
- Be the starter of the associated work item
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjActivityInstanceTerminate( FmcjActivityInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

COBOL

```
        FmcjAITerminate.
           CALL    "FmcjActivityInstanceTerminate"
                          USING
                          BY VALUE
                             hdlInstance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlInstance Input. The handle of the activity instance to be terminated.

Return type

long/ APIRET The return code of calling this method - see below.

Action and activity implementation calls

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The activity instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_ACT_IMPL_KIND(406)
The activity instance is not implemented by a program or process.

FMC_ERROR_WRONG_STATE(120)
The activity instance or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common

implementations of functions are taken from FmcjItem. That is, common functions start with the prefix FmcjItem; they are also defined starting with the prefix FmcjActivityInstanceNotification.

An activity instance notification is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on an activity instance notification. See "ActivityInstanceNotification" on page 259 for a complete list of API calls.

ActivityInstance()

This API call retrieves the activity instance the activity instance notification is associated to from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

In C++, when the activity instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the activity instance handle already points to some object.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstanceNotification
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationActivityInstance(
       FmcjActivityInstanceNotificationHandle hdlItem,
       FmcjActivityInstanceHandle *           instance )
```

C++ language signature

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java signature

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

COBOL

```
        FmcjAINActivityInstance.
           CALL    "FmcjActivityInstanceNotificationActivityInstance"
                         USING
                         BY VALUE
                            hdlItem
                         BY REFERENCE
                            instance
                         RETURNING
                            intReturnValue.
```

Parameters

hdlItem Input. The handle of the activity instance notification object to be queried.

instance Input/Output. The activity instance object to be retrieved (initialized).

returnCode Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET The return code of calling this API call - see return codes below.

ActivityInstance*/ ActivityInstance

A pointer to the activity instance or the activity instance the activity instance notification is associated to.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The activity instance notification does no longer exist. Or, the transient activity instance notification object recreated from its OID is not an activity instance notification; it is a process instance notification.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

StartTool()

This API call starts the specified support tool (action call).

The support tool must be one of the tools associated to the activity instance the notification has been created for. It is then started on the program execution agent associated to the logged-on user.

Note: A support tool can be started only via a program execution agent in the LAN environment; starting via a program execution server (in either environment) is currently not supported. The PES will simply ignore such an attempt.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Be the activity instance notification owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstanceNotification
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjAINStartTool.
           CALL    "FmcjActivityInstanceNotificationStartTool"
                         USING
                         BY VALUE
                            hdlItem
                            toolName
                         RETURNING
                            intReturnValue.
```

Parameters

hdlItem Input. The handle of the activity instance notification to be dealt with.

toolName Input. The support tool to be started.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_INVALID_TOOL(129)
No tool name is provided or the specified tool is not defined for the activity instance notification.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_KIND(501)
The transient activity instance notification object recreated from its OID is not an activity instance notification; it is a process instance notification.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Container activity implementation API calls

An FmcjContainer or Container object represents a data container of a process template, process instance, work item, or activity implementation. A container can be a read-only input container or a read/write input or output container.

The API calls defined 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 or COBOL, common implementations of functions are taken from FmcjContainer. That is, common functions start with the prefix FmcjContainer; they are also defined starting with the prefixes FmcjReadOnlyContainer and FmcjReadWriteContainer.

The following sections describe the activity implementation functions which are used for communication between an activity implementation or support tool and a program execution agent. See "Container" on page 264 for a complete list of API calls on containers.

InContainer()

This API call retrieves the input container from the CICS COMMAREA or IMS I/O Area (activity implementation call).

It can be used from within an activity implementation.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer().

Usage note

- See "Activity implementation API calls" on page 133 for general information.

Authorization

Be an activity implementation

Required connection

None but active MQ Workflow program execution server.

API interface declarations

C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx resepctively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent
COBOL	fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy)

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
```

COBOL

```
         FmcjCInCtnr.
            CALL    "FmcjContainerInContainer"
                           USING
                           BY REFERENCE
                              inputValue
                           RETURNING
                              intReturnValue.
```

Parameters

input Input/Output. The address of the input container handle respectively the input container of the activity implementation or support tool to be set.

Return type

long/ APIRET

The return code of calling this API call - see return codes below.

ReadOnlyContainer

The input container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)

The program does not have an input container.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)

The API call was not called from within an activity implementation or the program execution server is not active.

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 623
- For a C++ example see “Programming an executable (C++)” on page 624

OutContainer()

This API call retrieves the output container from the CICS COMMAREA or the IMS I/O Area (activity implementation call).

It can be used from within an activity implementation.

Note: This call will fail if the COMMAREA or I/O Area has been changed with SetOutContainer().

Usage note

- See “Activity implementation API calls” on page 133 for general information.

Authorization

Be an activity implementation

Required connection

None but active MQ Workflow program execution server.

API interface declarations

C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent
COBOL	fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy)

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
```

COBOL

```
         FmcjCOutCtnr.
            CALL    "FmcjContainerOutContainer"
                           USING
                           BY REFERENCE
                              outputValue
                           RETURNING
                              intReturnValue.
```

Parameters

output Input/Output. The address of the output container handle respectively the output container of the activity implementation to be set.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

ReadWriteContainer

The output container of the activity implementation.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)
The API call was not called from within an activity implementation or the program execution server is not active.

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 623
- For a C++ example see "Programming an executable (C++)" on page 624

SetOutContainer()

This API call returns the output container to the MQ Workflow program execution server (activity implementation call).

It can be used from within an activity implementation as often as required. Note, however, that the output container is not returned to the MQ Workflow execution server until the activity implementation ends. It is kept transiently in the CICS COMMAREA or IMS I/O Area.

Note: The calls InContainer(), OutContainer(), and Passthrough() will fail after this function is called, due to an altered COMMAREA or I/O Area.

Usage note
- See "Activity implementation API calls" on page 133 for general information.

Authorization

Be an activity implementation

Required connection

None but active MQ Workflow program execution server.

API interface declarations

C-language	fmcjccon.h respectively fmcjcrun.h
C++	fmcjpcon.hxx respectively fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent
COBOL	fmcvars.cpy, fmcperf.cpy (or fmcperfl.cpy)

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
```

COBOL

```
        FmcjCSetOutCtnr.
           CALL    "FmcjContainerSetOutContainer"
                         USING
                         BY VALUE
                            outputValue
                         RETURNING
                            intReturnValue.
```

Parameters

output Input. The output container handle respectively the output container of the activity implementation to be passed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_CONTAINER(509)
The container passed is not a valid output container for the activity implementation; wrong schema or version.

FMC_ERROR_NO_CTNR_ACCESS(1021)
The program does not have an output container.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_PROGRAM_EXECUTION(126)
The API call was not called from within an activity implementation or the program execution server is not active.

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 623
- For a C++ example see "Programming an executable (C++)" on page 624

Execution service actions

An FmcjExecutionService or ExecutionService object represents a session between a user and an MQ Workflow execution server so that Runtime services may be asked for.

The execution service object essentially provides for the basic API calls to set up a communication path to the specified MQ Workflow execution server and to establish the user session (log on), and finish it (log off).

At FmcjExecutionService or ExecutionService construction or allocation time the name of the MQ Workflow system and system group where the execution server resides can be specified. Default values are taken from the current user's profile or from the configuration profile, in this sequence, when logging on. The configuration where to search for the profiles can also be specified.

When the session to an execution server has been established, you can query objects for which you are authorized; for example, you can query process templates, process instances, or work items. The attributes of the queried objects can then be read and further actions can be requested. For example, once a process template has been queried, creation of a process instance can be asked for.

When the execution service object is destructed or deallocated and still represents an active session, logoff is automatically called (provided that there is no other object referencing this session). It is, however, recommended that logon and logoff calls are paired before the execution service object is deallocated.

FmcjService or Service represents common properties of services.

In the C++ language, FmcjExecutionService is thus a subclass of the FmcjService class and inherits all properties and methods. In the Java language, ExecutionService is thus a subclass of the Service class and inherits all properties and methods. Similarly, in the C-language or COBOL, common implementations of functions are taken from FmcjService. That is, common functions start with the prefix FmcjService; they are also defined starting with the prefix FmcjExecutionService.

The following sections describe the actions which can be applied on an execution service. See "ExecutionService" on page 274 for a complete list of API calls.

CreateProcessInstanceList()

This API call creates a process instance list on the MQ Workflow execution server so that process instances can be grouped to one's own taste or for a group of users (action call).

A process instance list is identified by:
- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the process instance list is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A process instance list groups a set of process instances which have the same characteristics. These characteristics are defined via search filters. The number of process instances in the list can be restricted via a threshold which specifies the maximum number of process instances to be returned to the client. That threshold is applied after the process instance list has been sorted according to sort criteria specified. Note that process instances are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a process instance list name:
- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:

  ```
  * ? " ; : .
  ```
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

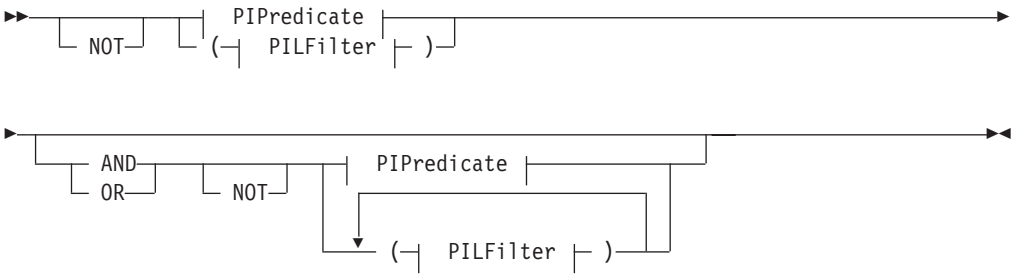
A process instance list filter is specified as a character string containing a filter on process instances (refer to "How to read the syntax diagrams" on page xii).

Notes:

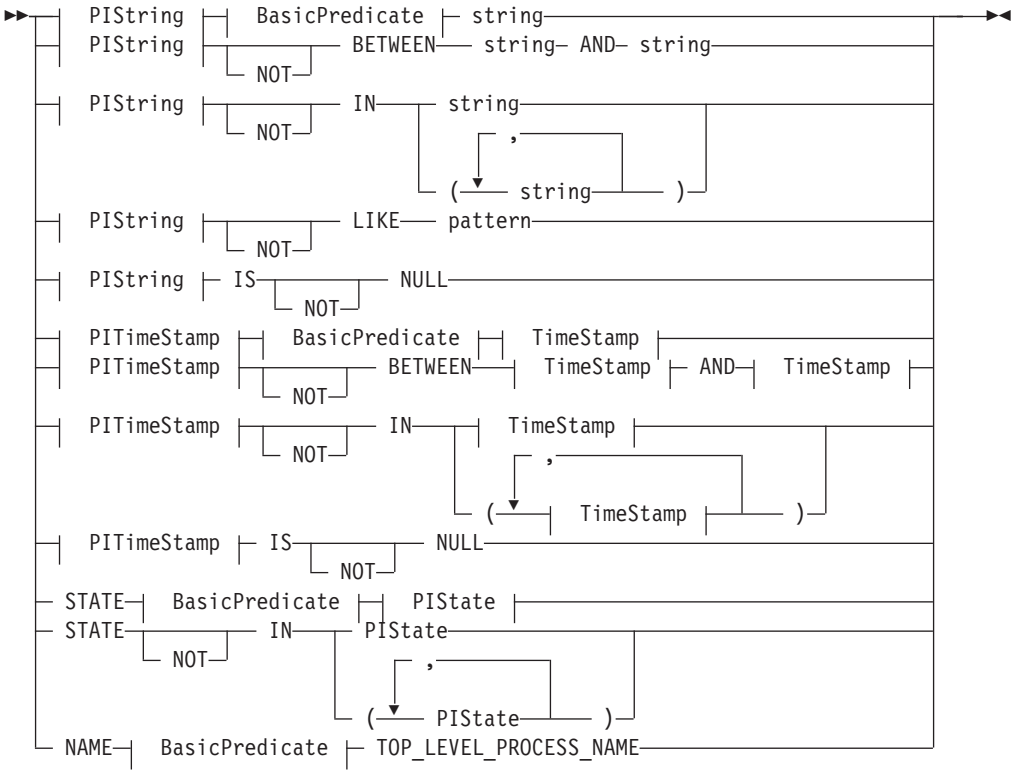
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.

5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

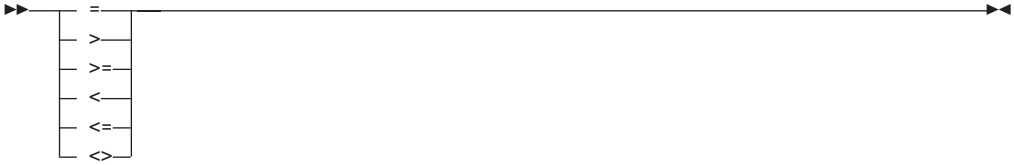
PILFilter



PIPredicate



BasicPredicate



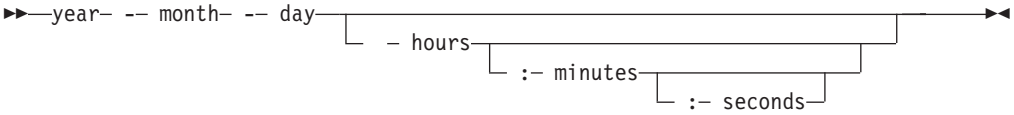
Action and activity implementation calls

PIState

PIString

PITimeStamp

TimeStamp

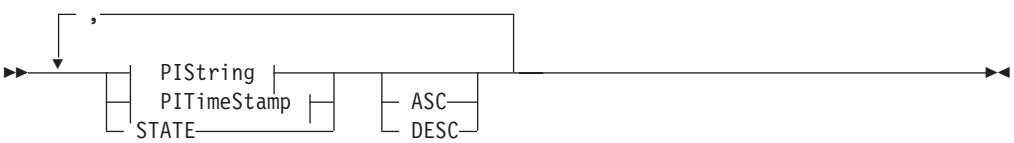


A process instance list sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PIState diagram.

PILOrderBy



Usage note

- See "Action API calls" on page 133 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
FmcjESCreateProcInstList.
   CALL    "FmcjExecutionServiceCreateProcessInstanceList"
                  USING
                  BY VALUE
                     serviceValue
                     name
                     typeValue
                     ownerValue
                     description
                     filter
                     sortCriteria
                     threshold
                  BY REFERENCE
                     newList
                  RETURNING
                     intReturnValue.
```

Parameters

description Input. A user-defined description of the process instance list.

descriptionIsNull
Input. Indicates whether a description is provided for the list.

filter Input. The filter criteria which characterize the process instances to be contained in the process instance list.

filterIsNull Input. Indicates whether a filter is provided for the list.

name Input. A user-defined name for the process instance list.

newList Input/Output. The newly created process instance list.

owner Input. The owner of the list when the type is private. Ignored for public lists.

ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the process instances in the process instance list.

sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.

threshold Input. The threshold which defines the maximum number of process instances in the process instance list to be passed to the client.

thresholdIsNull
Input. Indicates whether a threshold is provided for the list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

ProcessInstanceList
The newly created process instance list.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_INVALID_DESCRIPTION(810)
The specified description is invalid.
FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.
FMC_ERROR_INVALID_LIST_TYPE(813)
The specified list type is invalid.
FMC_ERROR_INVALID_NAME(134)
The specified process instance list name does not comply with the syntax rules.
FMC_ERROR_INVALID_USER(132)
The user ID specified for the owner of the list does not conform to the syntax rules.
FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.
FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid; exceeds the maximum possible value.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized.
FMC_ERROR_OWNER_NOT_FOUND(812)
The person to become the owner of the process instance list is not found.
FMC_ERROR_NOT_UNIQUE(121)
The name of the process instance list is not unique within the specified type.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Create a process instance list (C-language)" on page 575.
- For a C++ example see "Create a process instance list (C++)" on page 577.
- For a Java example see "Create a process instance list (Java)" on page 578.

CreateProcessTemplateList()

This API call creates a process template list on the MQ Workflow execution server so that process templates can be grouped to one's own taste or for a group of users (action call).

A process template list is identified by:
- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the process template list is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A process template list groups a set of process templates which have the same characteristics. These characteristics are defined via filters. The number of process templates in the list can be restricted via a threshold which specifies the maximum number of process templates to be returned to the client. That threshold is applied after the process template list has been sorted according to sort criteria specified. Process templates are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a process template list name:
- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:

  `* ? " ; : .`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

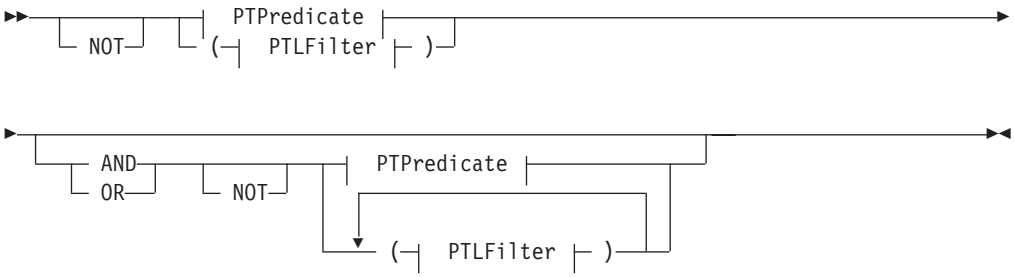
A process template list filter is specified as a character string containing a filter on process templates (refer to "How to read the syntax diagrams" on page xii).

Notes:

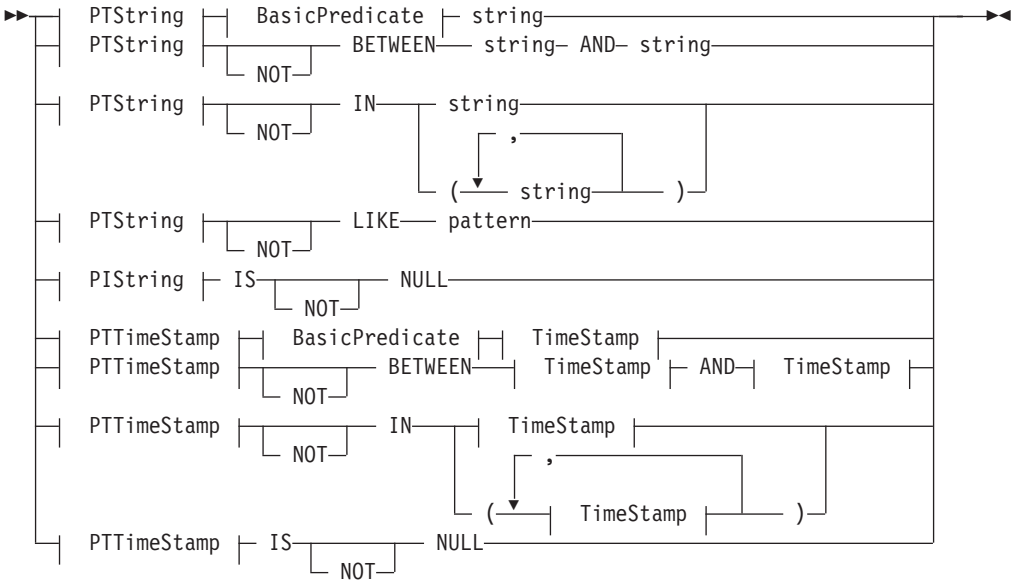
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.

5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PTLFilter



PTPredicate

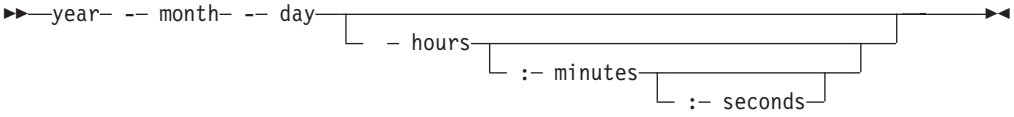


BasicPredicate

PTString

PTTimeStamp

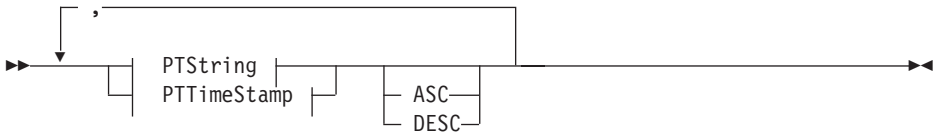
TimeStamp



A process template list sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTLOrderBy



Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESCreateProcTemplList.
           CALL    "FmcjExecutionServiceCreateProcessTemplateList"
                         USING
                         BY VALUE
                            serviceValue
                            name
                            typeValue
                            ownerValue
                            description
                            filter
                            sortCriteria
                            threshold
                         BY REFERENCE
                            newList
                         RETURNING
                            intReturnValue.
```

Parameters

description Input. A user-defined description of the process template list.

descriptionIsNull
Input. Indicates whether a description is provided for the list.
filter Input. The filter criteria which characterize the process templates in the process template list.
filterIsNull Input. Indicates whether a filter is provided for the list.
name Input. A user-defined name for the process template list.
newList Input/Output. The newly created process template list.
owner Input. The owner of the list when the type is private. Ignored for public lists.
ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.
service Input. A handle to the service object representing the session with the execution server.
sortCriteria Input. The sort criteria to be applied to the process templates in the process template list.
sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.
threshold Input. The threshold which defines the maximum number of process templates in the process template list.
thresholdIsNull
Input. Indicates whether a threshold is provided for the list.
type Input. An indication whether a private or a public list is to be created.

Return type
long/ APIRET The return code of calling this API call - see return codes below.
ProcessTemplateList
The newly created process template list.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_INVALID_DESCRIPTION(810)
The specified description is invalid.
FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.
FMC_ERROR_INVALID_LIST_TYPE(813)
The specified list type is invalid.
FMC_ERROR_INVALID_NAME(134)
The specified process template list name does not comply with the syntax rules.
FMC_ERROR_INVALID_USER(132)
The user ID specified for the owner of the list does not conform to the syntax rules.
FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.
FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid; exceeds the maximum possible value.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized.
FMC_ERROR_OWNER_NOT_FOUND(812)
The person to become the owner of the process template list is not found.
FMC_ERROR_NOT_UNIQUE(121)
The name of the process template list is not unique within the specified type.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Create a process instance list (C-language)" on page 575.
- For a C++ example see "Create a process instance list (C++)" on page 577.
- For a Java example see "Create a process instance list (Java)" on page 578.

CreateWorklist()

This API call creates a worklist on the MQ Workflow execution server so that work items or notifications can be grouped to one's own taste or for a group of users (action call).

A worklist is identified by:
- Its name, which is unique per type
- Its type, that is, an indicator whether the list is for public or private usage
- Its owner, that is, the owner of the list when the type is private

If the list is for public usage, any owner specification is ignored. If the list is for private usage and no owner is provided, then the list is created for the logged-on user.

When the worklist is to be created for public usage or for the private usage of another user, that is, not the logged-on user itself, then the logged-on user needs to have staff definition authorization.

A worklist groups a set of work items or notifications which have the same characteristics. These characteristics are defined via filters. The number of items in

the worklist can be restricted via a threshold which specifies the maximum number of items to be returned to the client. That threshold is applied after the worklist has been sorted according to sort criteria specified. Items are sorted on the server, that is, the code page of the server determines the sort sequence.

The following rules apply for specifying a worklist name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:

  `* ? " ; : .`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

The following rules apply for specifying a description:

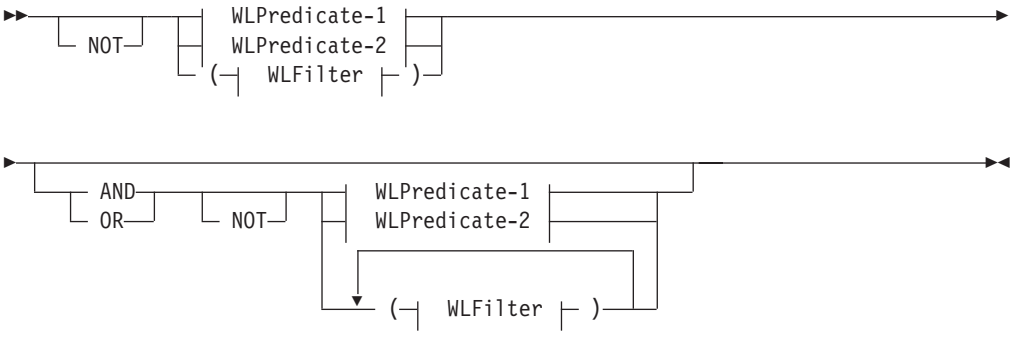
- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

A worklist filter is specified as a character string containing a filter on the items in the worklist (refer to "How to read the syntax diagrams" on page xii).

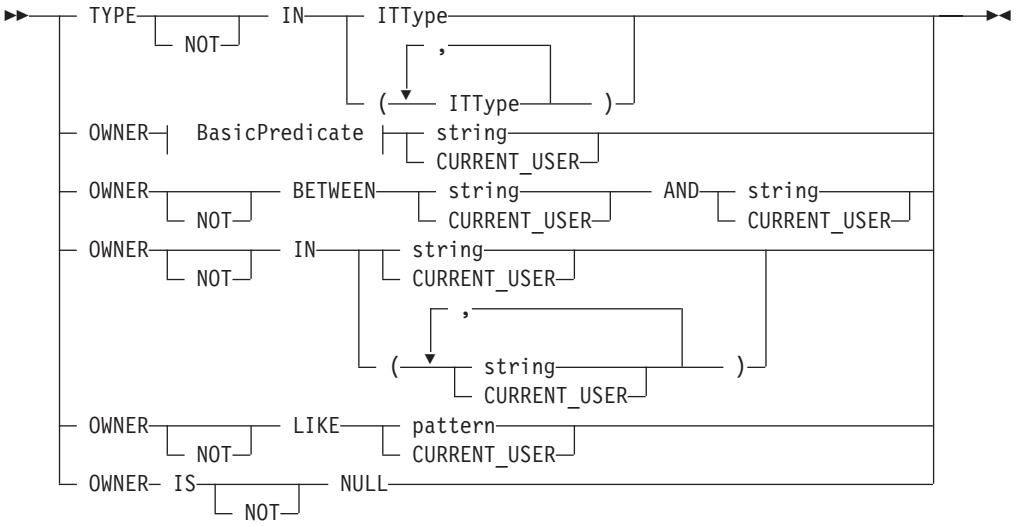
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

WLFilter

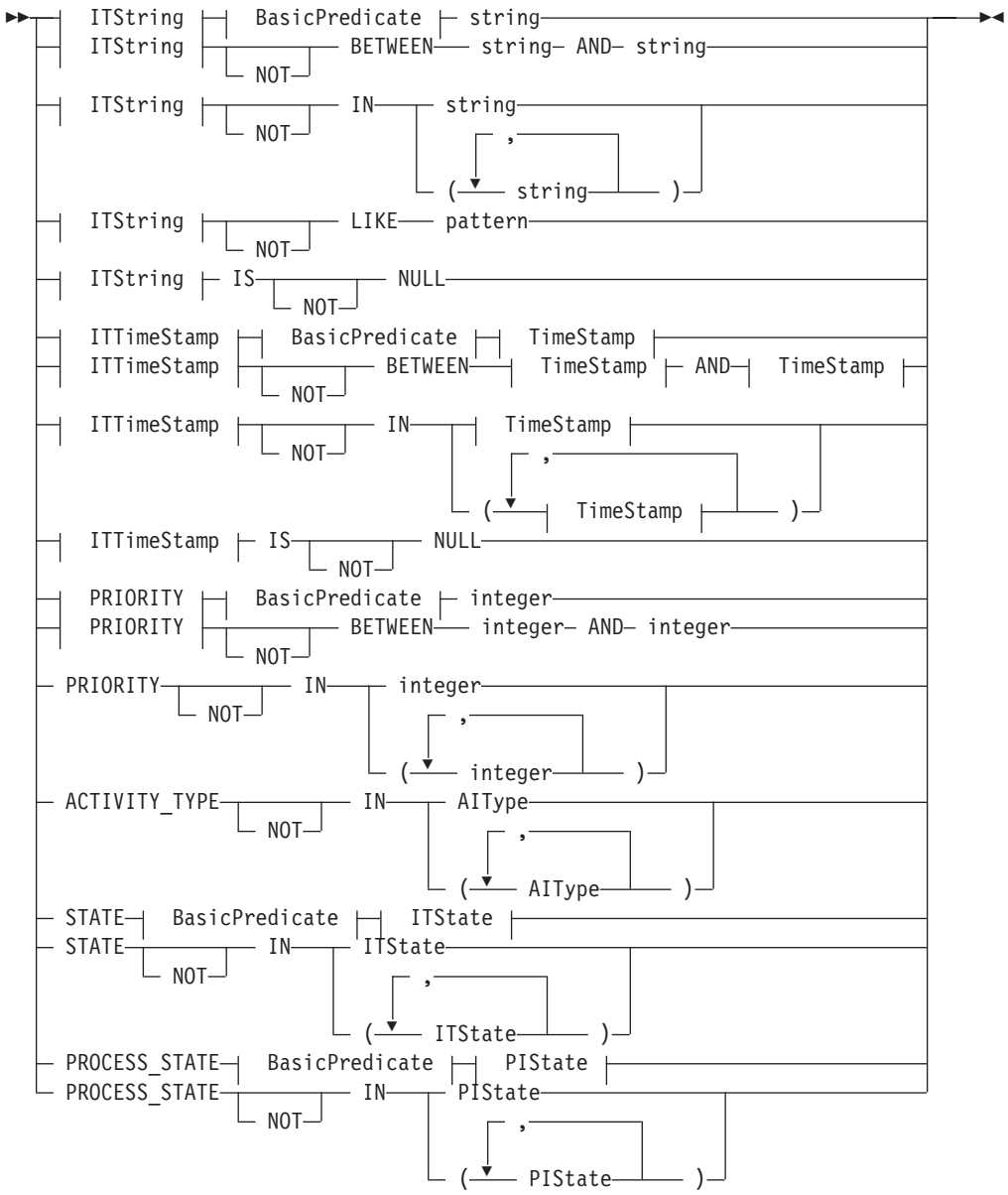


WLPredicate-1

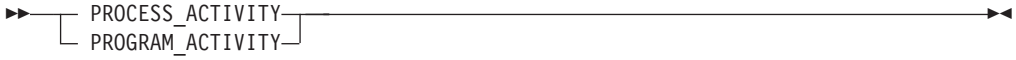


Action and activity implementation calls

WLPredicate-2

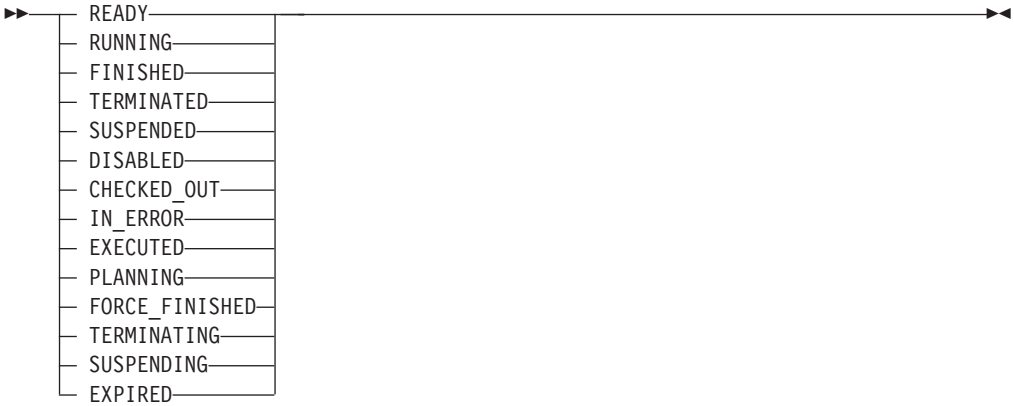


AIType



BasicPredicate

ITState



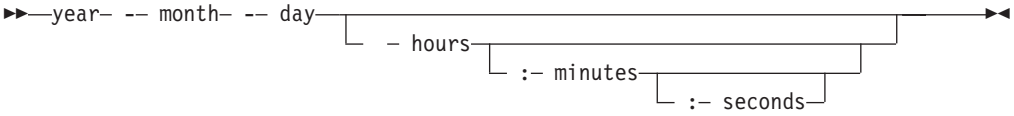
ITString

ITTimeStamp

ITType

PIState

TimeStamp



A worklist sort criterion is specified as a character string.

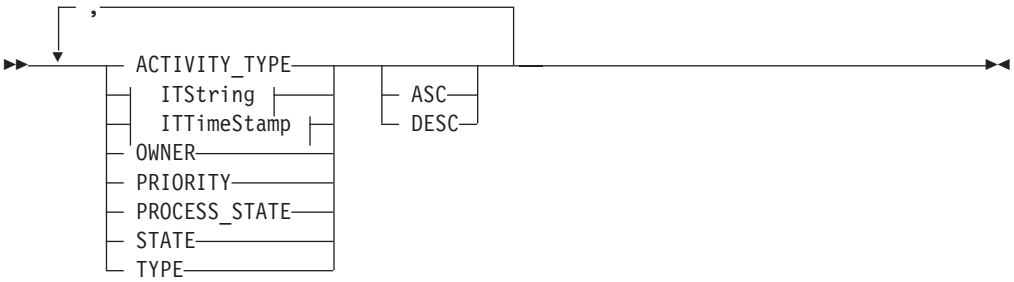
Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

Item types are sorted according to the sequence shown in the ITType diagram.

States are sorted according to the sequence shown in the ITState respectively the PIState diagram.

WLOrderBy



Usage note

- See "Action API calls" on page 133 for general information.

Authorization

None or staff definition or be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
FmcjESCreateWorklist.
   CALL    "FmcjExecutionServiceCreateWorklist"
                  USING
                  BY VALUE
                     serviceValue
                     name
                     typeValue
                     ownerValue
                     description
                     filter
                     sortCriteria
                     threshold
                  BY REFERENCE
                     newList
                  RETURNING
                     intReturnValue.
```

Parameters

description Input. A user-defined description of the worklist.

descriptionIsNull
Input. Indicates whether a description is provided for the list.

filter Input. The filter criteria which characterize the items in the worklist.

filterIsNull Input. Indicates whether a filter is provided for the list.

name Input. A user-defined name for the worklist.

newList Input/Output. The newly created worklist.

owner Input. The owner of the list when the type is private. Ignored for public lists.

ownerIsNull Input. Indicates whether a list owner is provided. No owner is needed for public lists.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the items in the worklist.

sortCriteriaIsNull
Input. Indicates whether sort criteria are provided for the list.

threshold Input. The threshold which defines the maximum number of items in the worklist.

thresholdIsNull
Input. Indicates whether a threshold is provided for the list.

type Input. An indication whether a private or a public list is to be created.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

WorkList The newly created worklist.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_DESCRIPTION(810)
The specified description is invalid.
FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.
FMC_ERROR_INVALID_LIST_TYPE(813)
The specified list type is invalid.
FMC_ERROR_INVALID_NAME(134)
The specified worklist name does not comply with the syntax rules.
FMC_ERROR_INVALID_USER(132)
The user ID specified for the owner of the list does not conform to the syntax rules.
FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.
FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid; exceeds the maximum possible value.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized.
FMC_ERROR_OWNER_NOT_FOUND(812)
The person to become the owner of the worklist is not found.
FMC_ERROR_NOT_UNIQUE(121)
The name of the worklist is not unique within the specified type.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Create a process instance list (C-language)" on page 575.
- For a C++ example see "Create a process instance list (C++)" on page 577.
- For a Java example see "Create a process instance list (Java)" on page 578.

Logoff()

This API call allows the application to finish the specified user session with an MQ Workflow execution server (action call).

When logoff has been successfully executed, no further client/server calls are accepted using this execution service object.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogoff(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET Logoff()
```

Java signature

```
public abstract
void   logoff() throws FmcException
```

COBOL

```
        FmcjESLogoff.
           CALL    "FmcjExecutionServiceLogoff"
                          USING
                          BY VALUE
                             serviceValue
                          RETURNING
                             intReturnValue.
```

Parameters

service Input. A handle to the service object representing the session with the execution server.

Return type
long/ APIRET

The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Chapter 6. Examples" on page 575.

Logon()

This API call allows an application to establish a user session with an MQ Workflow execution server (action call).

A successful Logon() is the prerequisite for using all other action and program execution management API calls of the MQ Workflow API.

You either log on by specifying a user ID and a password or you log on by specifying user credentials which are then verified by your authentication exit.

The user ID to log on with, respectively the user ID returned by your authentication exit, must be a registered MQ Workflow user.

When the execution server supports *unified logon* and when you log on with a user ID and password, an empty password and user ID can be provided. The user ID to log on with is then retrieved from the operating system, that is, logon must have been performed at the client. The client is trusted and the execution server performs no password checking.

After a successful logon, the execution service object represents that single user session. A further request to log on with a different user ID will be rejected. You can, however, establish as many sessions as needed, even for the same user, using different execution service objects, one for each session.

At logon time, you can specify your mode of operation. When you are operating in a *present* session mode, the execution server can assume that you are able to react to requests from activity implementations which might ask, for example, for container data. Thus, activity instances that are started automatically may be scheduled on your behalf - provided that you also started a program execution agent.

Furthermore, the *present* mode indicates to MQ Workflow that the session can handle unsolicitedmessages pushed by the execution server - see "The push data access model" on page 13 for additional prerequisites.

There can only be a single present session for one user. The *present here* option can be used, to force that other present session logoff and to newly establish a present session here. Note that using a present here session mode also requests to shut down the program execution agent.

When you are operating in a *default* session mode, the execution server does not assume that you are able to react. Activity instances are not automatically started on your behalf and messages are not pushed to you. There can be multiple sessions for one user with the *default* session mode.

The following enumeration types can be used to specify the session mode:

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.

- **C-language** Fmc_SM_Default
- **C++** FmcjService::Default respectively FmcjExecutionService::Default
- **JAVA** SessionMode.DEFAULT
- **COBOL** Fmc-SM-Default

Present Indicates that you want to operate in a present session mode.

- **C-language** Fmc_SM_Present
- **C++** FmcjService::Present respectively FmcjExecutionService::Present
- **JAVA** SessionMode.PRESENT
- **COBOL** Fmc-SM-Present

PresentHere Indicates that you want to operate in a present session mode. If a session with the present session mode already exists, then it should be logged off.

- **C-language** Fmc_SM_PresentHere

C++ FmcjService::PresentHere respectively FmcjExecutionService::PresentHere

JAVA SessionMode.PRESENT_HERE

COBOL Fmc-SM-PresentHere

At logon time, you can also specify whether you are back in case you are set to be absent. When you are not absent you participate in work assignment; otherwise no work items are assigned to you.

The following enumeration types can be used to deal with your absence:

C-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.

C-language Fmc_SA_NotSet

C++ FmcjService::NotSet respectively FmcjExecutionService::NotSet

JAVA AbsenceIndicator.NOT_SET

COBOL Fmc-SA-NotSet

Reset Indicates that your absence setting is to be reset; you are back.

C-language Fmc_SA_Reset

C++ FmcjService::Reset respectively FmcjExecutionService::Reset

JAVA AbsenceIndicator.RESET

COBOL Fmc-SA-Reset

Leave Indicates that your absence setting should stay as is; you are either absent or not.

C-language Fmc_SA_Leave

C++ FmcjService::Leave respectively FmcjExecutionService::Leave

JAVA AbsenceIndicator.LEAVE

COBOL Fmc-SA-Leave

Usage note

- See “Action API calls” on page 133 for general information.

Authorization

Be a registered MQ Workflow user

Required connection

Action and activity implementation calls

None

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogon (
       FmcjExecutionServiceHandle        service,
       char const *                      userID,
       char const *                      password,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator )

APIRET FMC_APIENTRY FmcjExecutionServiceLogonWithCredentials(
       FmcjExecutionServiceHandle        service,
       FmcjBinary const *                userCredentials,
       unsigned long                     userCredentialsLength,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator,
       char const *                      userName )
```

C++ language signature

```
APIRET Logon(
       string const &                    userID,
       string const &                    password,
       SessionMode                       sessionMode = Present,
       AbsenceIndicator                  absenceIndicator = NotSet )

APIRET Logon(
       FmcjBinary const *                userCredentials,
       unsigned long                     userCredentialsLength,
       SessionMode                       sessionMode = Present,
       AbsenceIndicator                  absenceIndicator = NotSet,
       string const *                    userName = 0 )
```

Java signature

```
public abstract
void   logon ( String userID,   String password )

public abstract
void   logon2( String           userID,
               String           password,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator ) throws FmcException

public abstract
void   logon3( Byte[]           userCredentials  ) throws FmcException

public abstract
void   logon4( Byte[]           userCredentials,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator,
               String           userName         ) throws FmcException
```

COBOL

```
        FmcjESLogon.
           CALL    "FmcjExecutionServiceLogon"
                          USING
                          BY VALUE
                             serviceValue
                             userID
                             passwordValue
                             sessionMode
                             absenceIndicator
                          RETURNING
                             intReturnValue.

     FmcjESLogonWithCredentials.
           CALL    "FmcjExecutionServiceLogonWithCredentials"
                          USING
                          BY VALUE
                             serviceValue
                             userCredentials
                             userCredentialsLength
                             sessionMode
                             absenceIndicator
                             userName
                          RETURNING
                             intReturnValue.
```

Parameters

absenceIndicator
Input. An indicator to state how to handle any absence set.

password
Input. The password of the user. Can be empty for unified logon.

service
Input. A handle to the service object representing the session to be established with the execution server.

sessionMode
Input. The mode of the session to be established.

userCredentials
Input. The user credentials to be passed to a user-provided authentication exit.

userCredentialsLength
Input. The length of the binary user credentials string.

userID Input. The user ID of the user on whose behalf a logon is to be made. Can be empty for unified logon.

userName Input. An optional user name to be passed to a user-provided authentication exit.

Return type
long/ APIRET

The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_ALREADY_LOGGED_ON(11)
The user is already logged on with present mode or the execution service object already represents a different user session.

FMC_ERROR_AUTHENTICATION(513)
Your authentication exit rejected the logon request. See the first parameter in the result object for the error reason.

FMC_ERROR_BACK_LEVEL_VERSION(504)
The version of the client is out-of-date, that is, not supported by this server.

FMC_ERROR_INVALID_ABSENCE_SPEC(905)
An unknown absence setting has been specified.

FMC_ERROR_INVALID_SESSION_MODE(901)
An unknown session mode has been specified.

FMC_ERROR_LOGON_DENIED(512)
The logon request has been denied by your authentication exit.

FMC_ERROR_NEWER_VERSION(505)
The version of the client is newer than the server version, that is, not supported.

FMC_ERROR_NOT_AUTHORIZED(119)
A user-provided authentication exit or entry points in the DLL to pass the credentials to are not found. Or, the authentication exit returns a recoverable error.

FMC_ERROR_PASSWORD(12)
Incorrect password.

FMC_ERROR_PROFILE(124)
The configuration profile or profile entries (system group, system) cannot be found.

FMC_ERROR_USERID_UNKNOWN(10)
No user ID registered with MQ Workflow has been provided.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

For examples see "Chapter 6. Examples" on page 575.

Passthrough()

This API call can be used by an activity implementation to establish a user session with an MQ Workflow execution server from within this program (activity-implementation call).

When successfully executed, a session to the same execution server is set up from where the work item implemented by this program was started; the user on whose behalf the session is set up is the same one on whose behalf the work item was started.

Note: This call will fail after the COMMAREA or IMS I/O Area has been changed with SetOutContainer().

Usage note
- See "Activity implementation API calls" on page 133 for general information.

Authorization

Activity implementation started by MQ Workflow

Required connection

None but active MQ Workflow program execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServicePassthrough(
       FmcjExecutionServiceHandle service )
```

C++ language signature

```
APIRET Passthrough()
```

Java signature

```
public abstract
void passthrough()  throws FmcException
```

COBOL

```
        FmcjESPassthrough.
           CALL    "FmcjExecutionServicePassthrough"
                          USING
                          BY VALUE
                             serviceValue
                          RETURNING
                             intReturnValue.
```

Parameters

service Input. A handle to the service object which is to represent the session to be established with the execution server.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_PROGRAM_EXECUTION(126)
Passthrough was not called from within an activity implementation or the program execution server is not active.

FMC_ERROR_USERID_UNKNOWN(10)
The user who started the work item does no longer exist.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Programming an executable (C-language)" on page 623.
- For a C++ example see "Programming an executable (C++)" on page 624.

QueryActivityInstanceNotifications()

This API call retrieves the activity instance notifications the user has access to from the MQ Workflow execution server (action call).

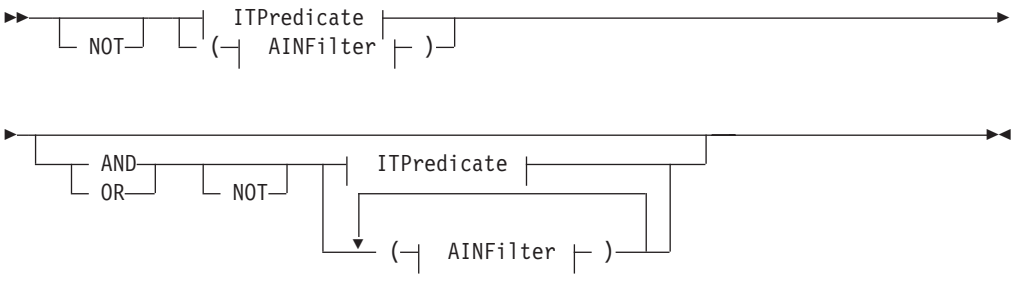
In C, C++, and COBOL, any activity instance notifications retrieved are appended to the supplied vector. If you want to read the current activity instance notifications only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL, respectively erase all elements of the vector in the C++ API.

The activity instance notifications to be retrieved can be characterized by a filter. An activity instance notification filter is specified as a character string:

Notes:

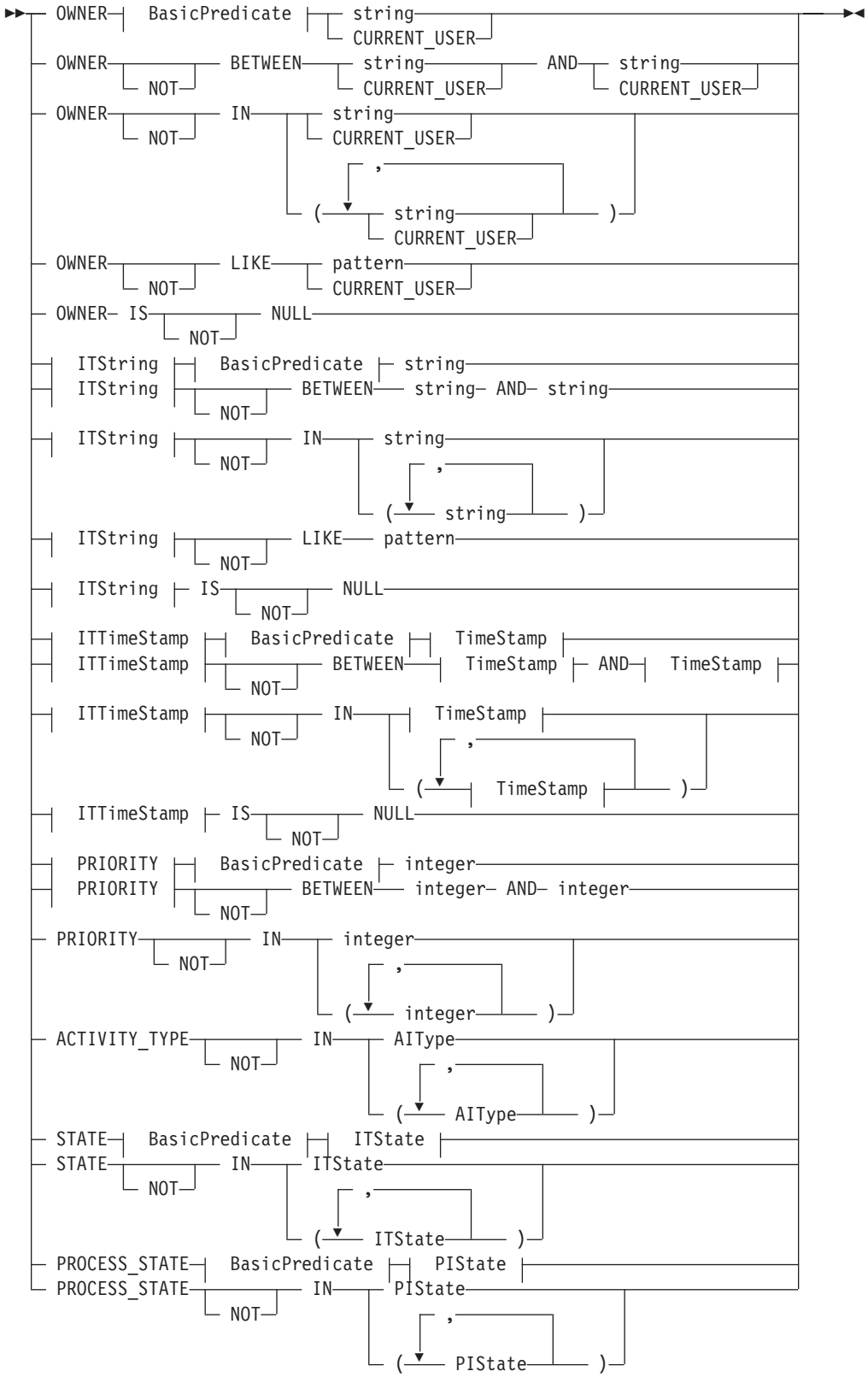
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

AINFilter



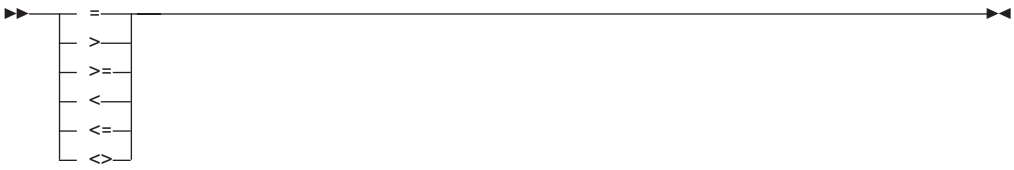
Action and activity implementation calls

ITPredicate



AIType

BasicPredicate



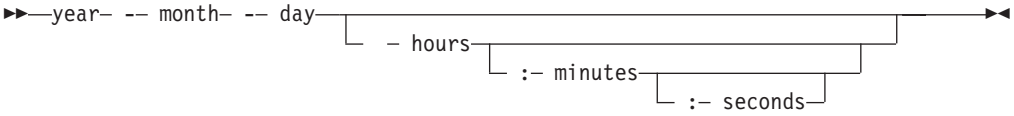
ITState

ITString

ITTimeStamp

PIState

TimeStamp



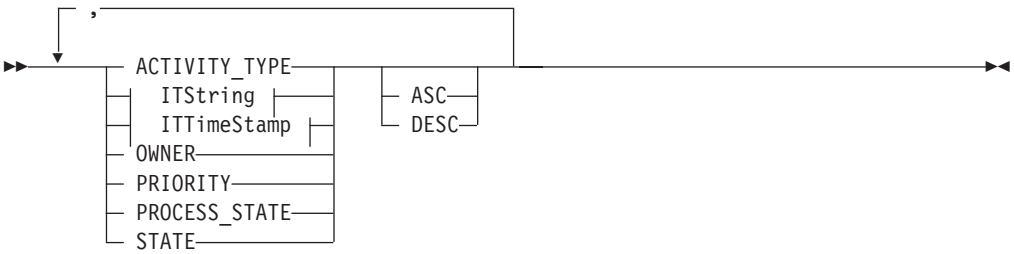
Activity instance notifications can be sorted. An activity instance notification sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState respectively the PIState diagram.

AINOrderBy



The number of activity instance notifications to be retrieved can be restricted via a threshold which specifies the maximum number of activity instance notifications to be returned to the client. That threshold is applied after the activity instance notifications have been sorted according to the sort criteria specified. Note that the activity instance notifications are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each activity instance notification is:
- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation

- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryActInstNotifs.
           CALL
              "FmcjExecutionServiceQueryActivityInstanceNotifications"
                          USING
                          BY VALUE
                             serviceValue
                             filter
                             sortCriteria
                             threshold
                          BY REFERENCE
                             notifications
                          RETURNING
                             intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the activity instance notifications to be retrieved.

notifications Input/Output. The qualifying vector of activity instance notifications.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the activity instance notifications found.

threshold Input. The threshold which defines the maximum number of activity instance notifications to be returned to the client.

Return type

APIRET The return code of calling this API call - see return codes below.

ActivityInstanceNotification[]
The qualifying activity instance notifications.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of activity instance notifications to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query process instances (C-language)" on page 601.
- For a C++ example see "Query process instances (C++)" on page 602.
- For a Java example see "Query process instances (Java)" on page 603.

QueryItems()

This API call retrieves the work items or notifications the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any items retrieved are appended to the supplied vector. If you want to read the current items only, you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

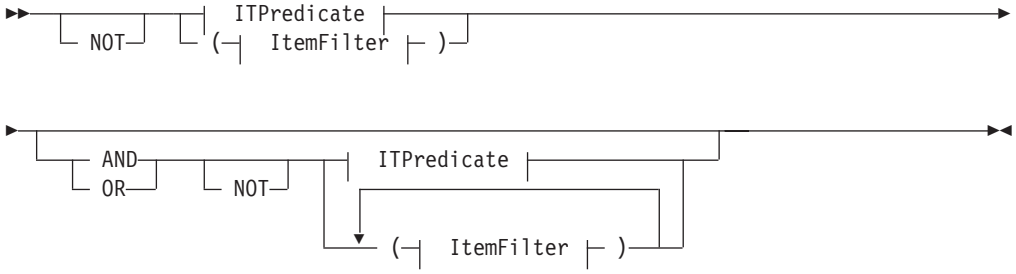
The items to be retrieved can be characterized by a filter. An item filter is specified as a character string.

Notes:

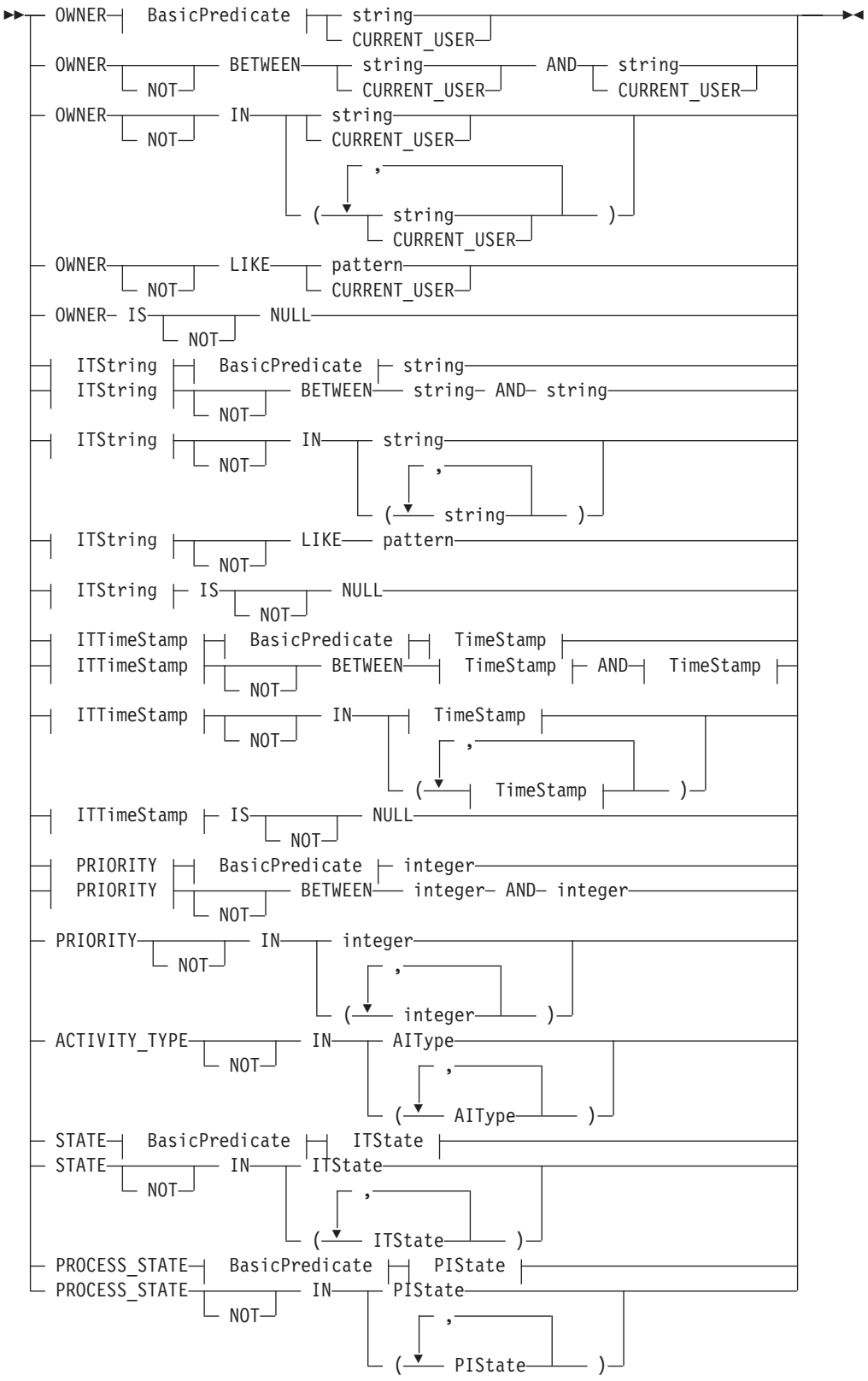
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.

6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

ItemFilter



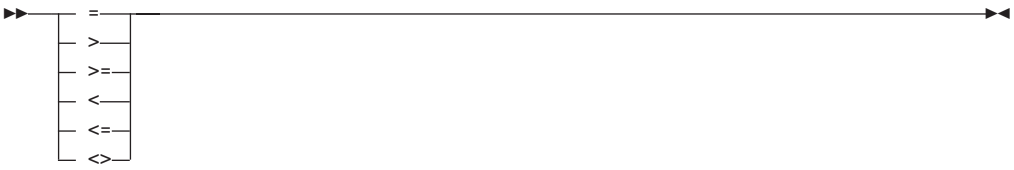
ITPredicate



Action and activity implementation calls

AIType

BasicPredicate



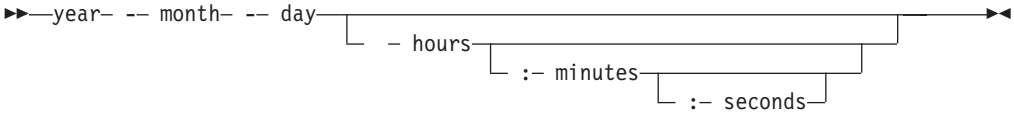
ITState

ITString

ITTimeStamp

PIState

TimeStamp



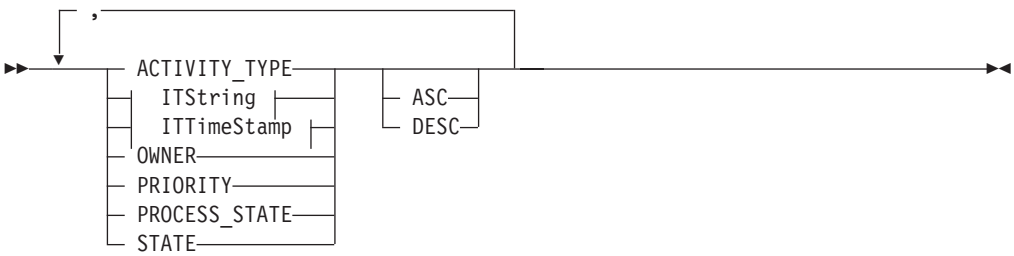
Items can be sorted. An item sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState respectively the PIState diagram.

ItemOrderBy



The number of items to be retrieved can be restricted via a threshold which specifies the maximum number of items to be returned to the client. That threshold is applied after the items have been sorted according to the sort criteria specified. Note that the items are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each item is:
- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime

- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
FmcjESQueryItems.
   CALL    "FmcjExecutionServiceQueryItems"
                  USING
                  BY VALUE
                     serviceValue
                     filter
                     sortCriteria
                     threshold
                  BY REFERENCE
                     items
                  RETURNING
                     intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the items to be retrieved.

items Input/Output. The qualifying vector of items.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the items found.

threshold Input. The threshold which defines the maximum number of items to be returned to the client.

Return type

APIRET The return code of calling this API call - see return codes below.

Item[] The qualifying items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of items to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 601.
- For a C++ example see "Query process instances (C++)" on page 602.
- For a Java example see "Query process instances (Java)" on page 603.

QueryProcessInstanceLists()

This API call retrieves the process instance lists the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any process instance lists retrieved are appended to the supplied vector. If you want to read the current process instance lists only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL, respectively erase all elements of the vector in the C++ API.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryProcInstLists.
           CALL    "FmcjExecutionServiceQueryProcessInstanceLists"
                         USING
                         BY VALUE
                            serviceValue
                         BY REFERENCE
                            lists
                         RETURNING
                            intReturnValue.
```

Parameters

lists Input/Output. The vector of process instance lists.

service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

ProcessInstanceList[]
The qualifying process instance lists.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instance lists to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query worklists (C-language)" on page 586.
- For a C++ example see "Query worklists (C++)" on page 588.
- For a Java example see "Query worklists (Java)" on page 590.

QueryProcessInstanceNotifications()

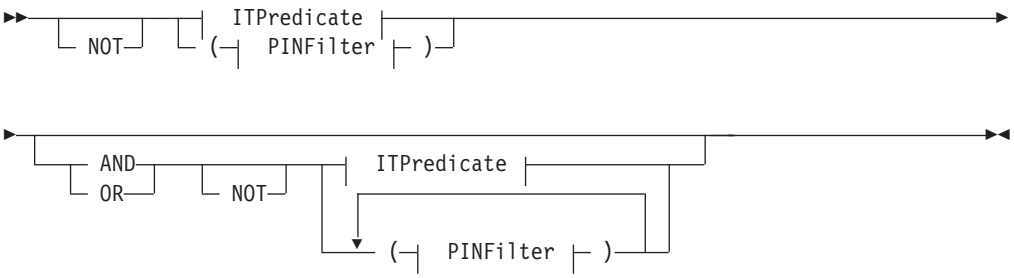
This API call retrieves the process instance notifications the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any process instance notifications retrieved are appended to the supplied vector. If you want to read the current process instance notifications only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

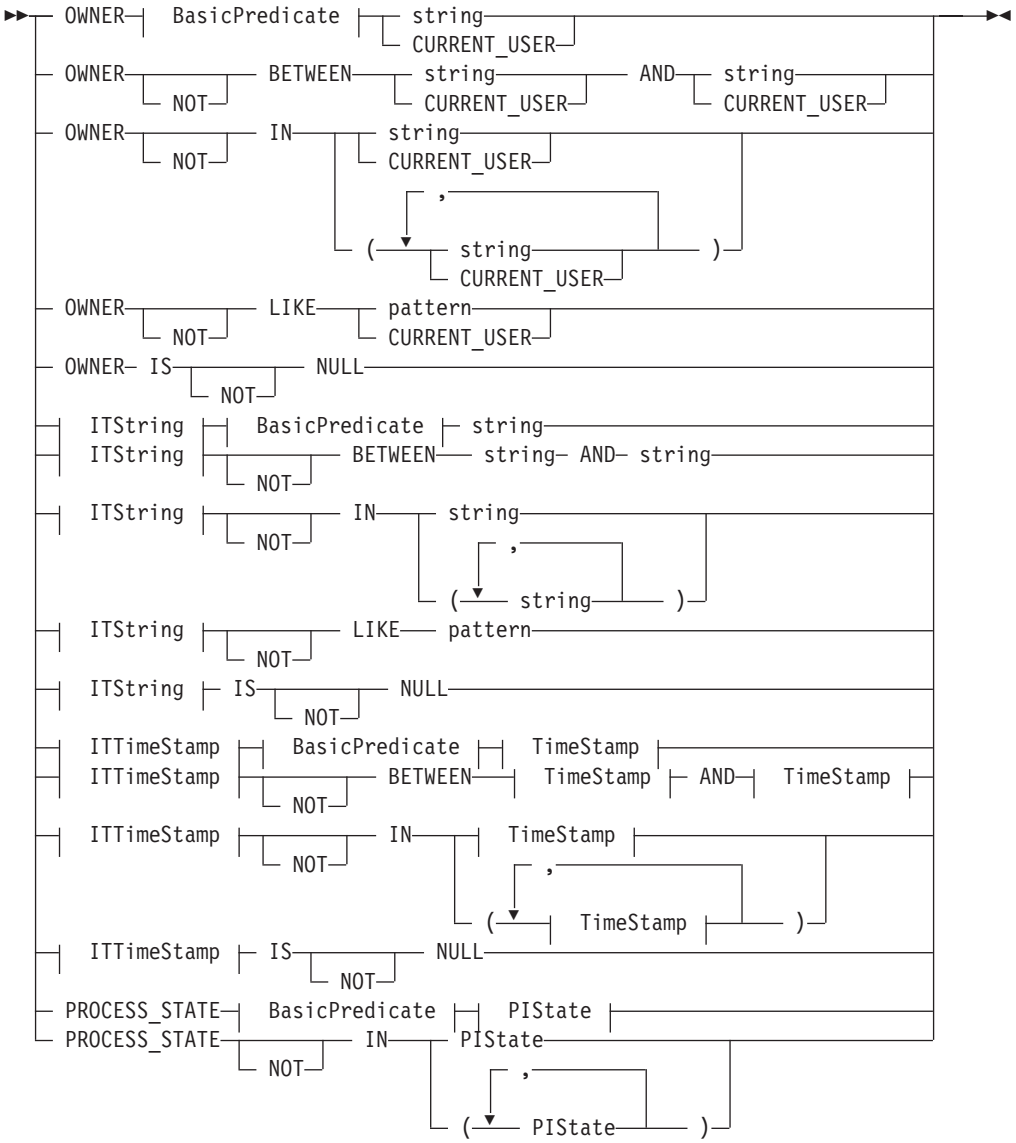
The process instance notifications to be retrieved can be characterized by a filter. A process instance notification filter is specified as a character string.

Notes:
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PINFilter



ITPredicate

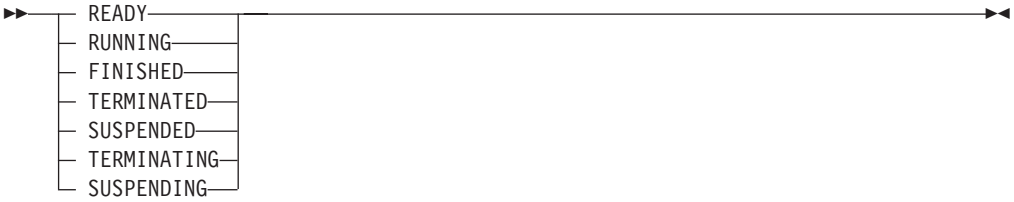


BasicPredicate

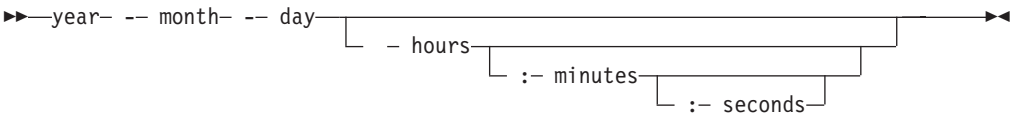
ITString

ITTimeStamp

PIState



TimeStamp



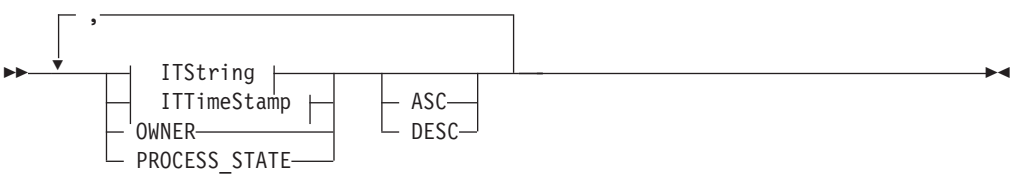
Process instance notifications can be sorted. A process instance notification sort criterion is specified as a character string.

Note: The default sort order is ascending.

Activity types are sorted according to the sequence shown in the AIType diagram.

States are sorted according to the sequence shown in the ITState respectively the PIState diagram.

PINOrderBy



The number of process instance notifications to be retrieved can be restricted via a threshold which specifies the maximum number of process instance notifications to be returned to the client. That threshold is applied after the activity instance notifications have been sorted according to the sort criteria specified. Note that the process instance notifications are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process instance notification is:
- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryProcInstNotifs.
           CALL
               "FmcjExecutionServiceQueryProcessInstanceNotifications"
                         USING
                         BY VALUE
                            serviceValue
                            filter
                            sortCriteria
                            threshold
                         BY REFERENCE
                            notifications
                         RETURNING
                            intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the process instance notifications to be retrieved.

items Input/Output. The qualifying vector of process instance notifications.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the process instance notifications found.

threshold Input. The threshold which defines the maximum number of process instance notifications to be returned to the client.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstanceNotification[]
The qualifying process instance notifications.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is not applicable to process instance notifications.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are not applicable to process instance notifications.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instance notifications to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 601.
- For a C++ example see “Query process instances (C++)” on page 602.
- For a Java example see “Query process instances (Java)” on page 603.

QueryProcessInstances()

This API call retrieves the current process instances the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL any process instances retrieved are appended to the supplied vector. If you want to read the current process instances only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

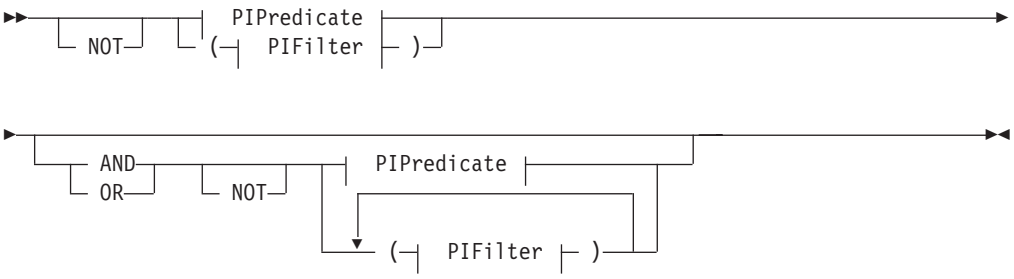
A filter on process instances is specified as a character string containing a filter predicate:

Notes:

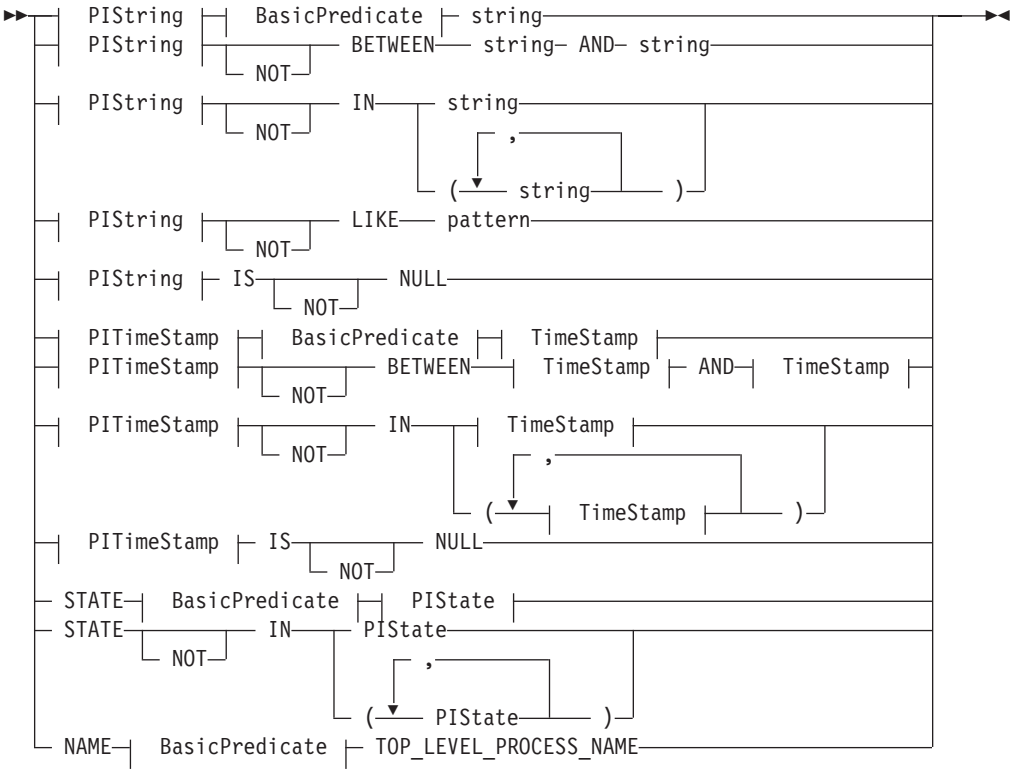
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').

3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PIFilter

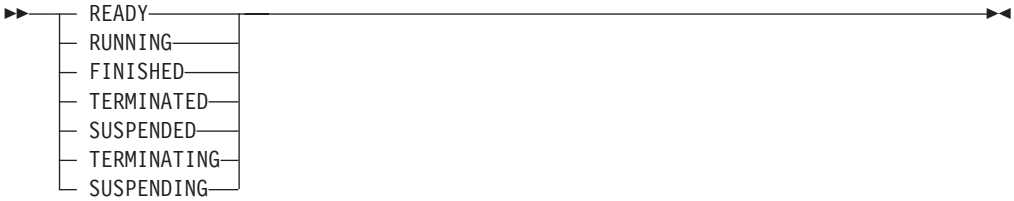


PIPredicate



BasicPredicate

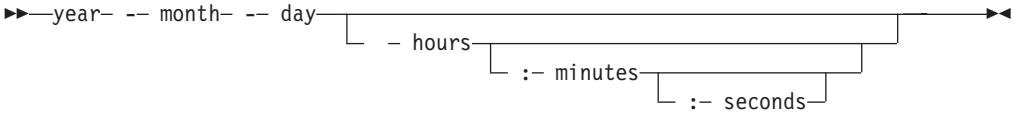
PIState



PIString

PITimeStamp

TimeStamp

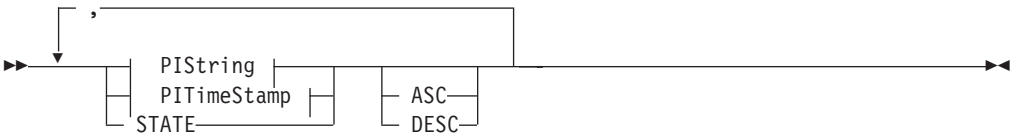


Process instances can be sorted. A process instance sort criterion is specified as a character string.

Note: The default sort order is ascending.

States are sorted according to the sequence shown in the PIState diagram.

PIOrderBy



The number of process instances to be retrieved can be restricted via a threshold which specifies the maximum number of process instances to be returned to the client. That threshold is applied after the process instances have been sorted according to the sort criteria specified. Note that the process instances are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process instance is:
- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- State
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

Usage note
- See “Action API calls” on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryProcInsts.
           CALL    "FmcjExecutionServiceQueryProcessInstances"
                         USING
                         BY VALUE
                            serviceValue
                            filter
                            sortCriteria
                            threshold
                         BY REFERENCE
                            instances
                         RETURNING
                            intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the process instances to be retrieved.

instances Input/Output. The qualifying vector of process instances.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the process instances found.

threshold Input. The threshold which defines the maximum number of process instances to be returned to the client.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance[]
The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is not applicable to process instances.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are not applicable to process instances.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instances to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see “Query process instances (C-language)” on page 601.
- For a C++ example see “Query process instances (C++)” on page 602.
- For a Java example see “Query process instances (Java)” on page 603.

QueryProcessTemplateLists()

This API call retrieves the current process template lists the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any process template lists retrieved are appended to the supplied vector. If you want to read the current process template lists only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note

- See “Action API calls” on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryProcTemplLists.
           CALL    "FmcjExecutionServiceQueryProcessTemplateLists"
                          USING
                          BY VALUE
                             serviceValue
                          BY REFERENCE
                             lists
                          RETURNING
                             intReturnValue.
```

Parameters

lists	Input/Output. The vector of process template lists.
service	Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET	The return code of calling this API call - see return codes below.
ProcessTemplateList[]	The qualifying process template lists.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process template lists to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query worklists (C-language)" on page 586.
- For a C++ example see "Query worklists (C++)" on page 588.
- For a Java example see "Query worklists (Java)" on page 590.

QueryProcessTemplates()

This API call retrieves the current process templates from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any process templates retrieved are appended to the supplied vector. If you want to read the current process templates only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

A filter on process templates is specified as a character string containing a filter predicate:

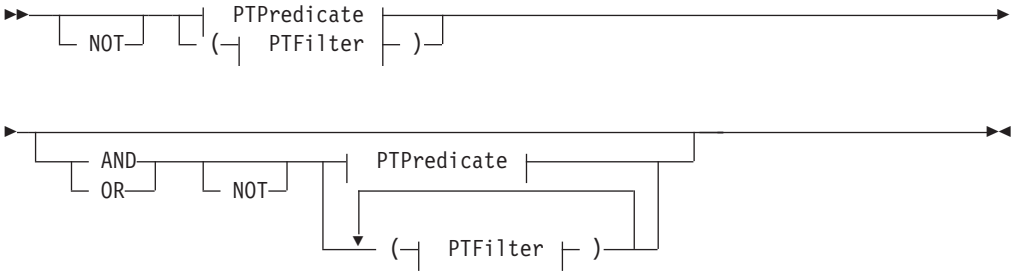
Notes:

1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.

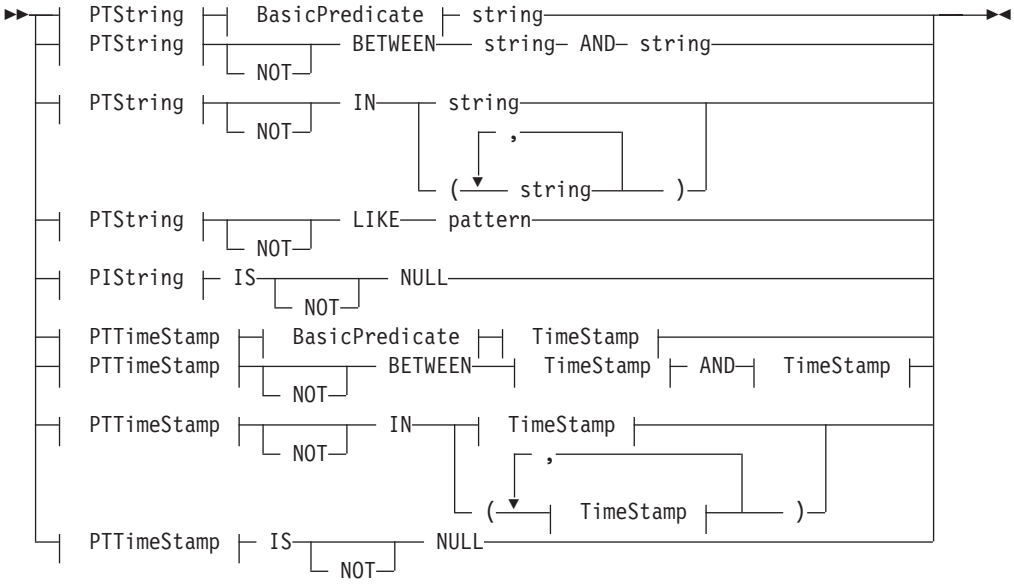
- The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).

4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

PTFilter



PTPredicate



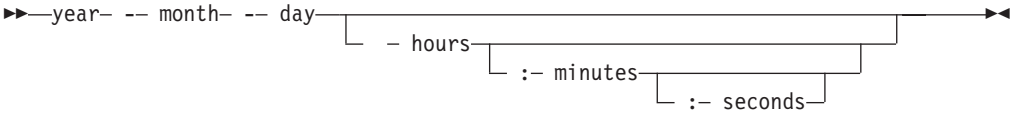
BasicPredicate

Action and activity implementation calls

PTString

PTTimeStamp

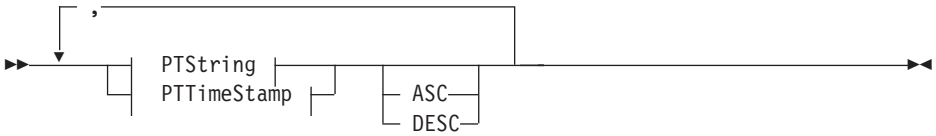
TimeStamp



Process templates can be sorted. A process template sort criterion is specified as a character string.

Note: The default sort order is ascending.

PTOrderBy



The number of process templates to be retrieved can be restricted via a threshold which specifies the maximum number of process templates to be returned to the client. That threshold is applied after the process templates have been sorted according to the sort criteria specified. Note that the process templates are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each process template is:
- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryProcTempls.
           CALL    "FmcjExecutionServiceQueryProcessTemplates"
                         USING
                         BY VALUE
                            serviceValue
                            filter
                            sortCriteria
                            threshold
                         BY REFERENCE
                            templates
                         RETURNING
                            intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the process templates to be retrieved.
service Input. A handle to the service object representing the session with the execution server.
sortCriteria Input. The sort criteria to be applied to the process templates found.
templates Input/Output. The qualifying vector of process templates.
threshold Input. The threshold which defines the maximum number of process templates to be returned to the client.

Return type
APIRET
The return code of calling this API call - see return codes below.
ProcessTemplate[]
The qualifying process templates.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_INVALID_FILTER(125)
The specified filter is not applicable to process templates.
FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are not applicable to process templates.
FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process templates to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see “Query process instances (C-language)” on page 601.
- For a C++ example see “Query process instances (C++)” on page 602.

- For a Java example see "Query process instances (Java)" on page 603.

QueryWorkitems()

This API call retrieves the work items the user has access to from the MQ Workflow execution server (action call).

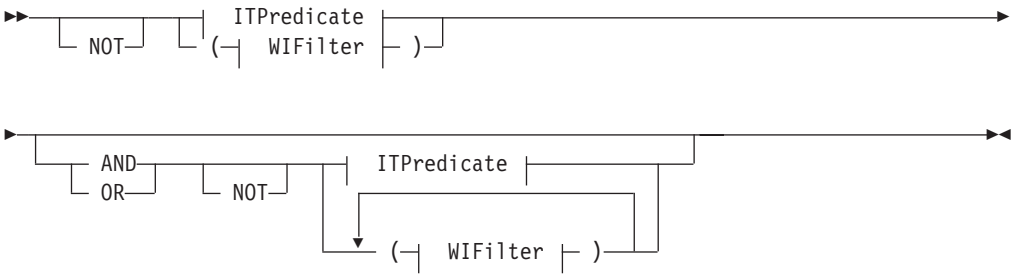
In C, C++, and COBOL, any work items retrieved are appended to the supplied vector. If you want to read the current work items only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

The work items to be retrieved can be characterized by a filter. A work item filter is specified as a character string:

Notes:

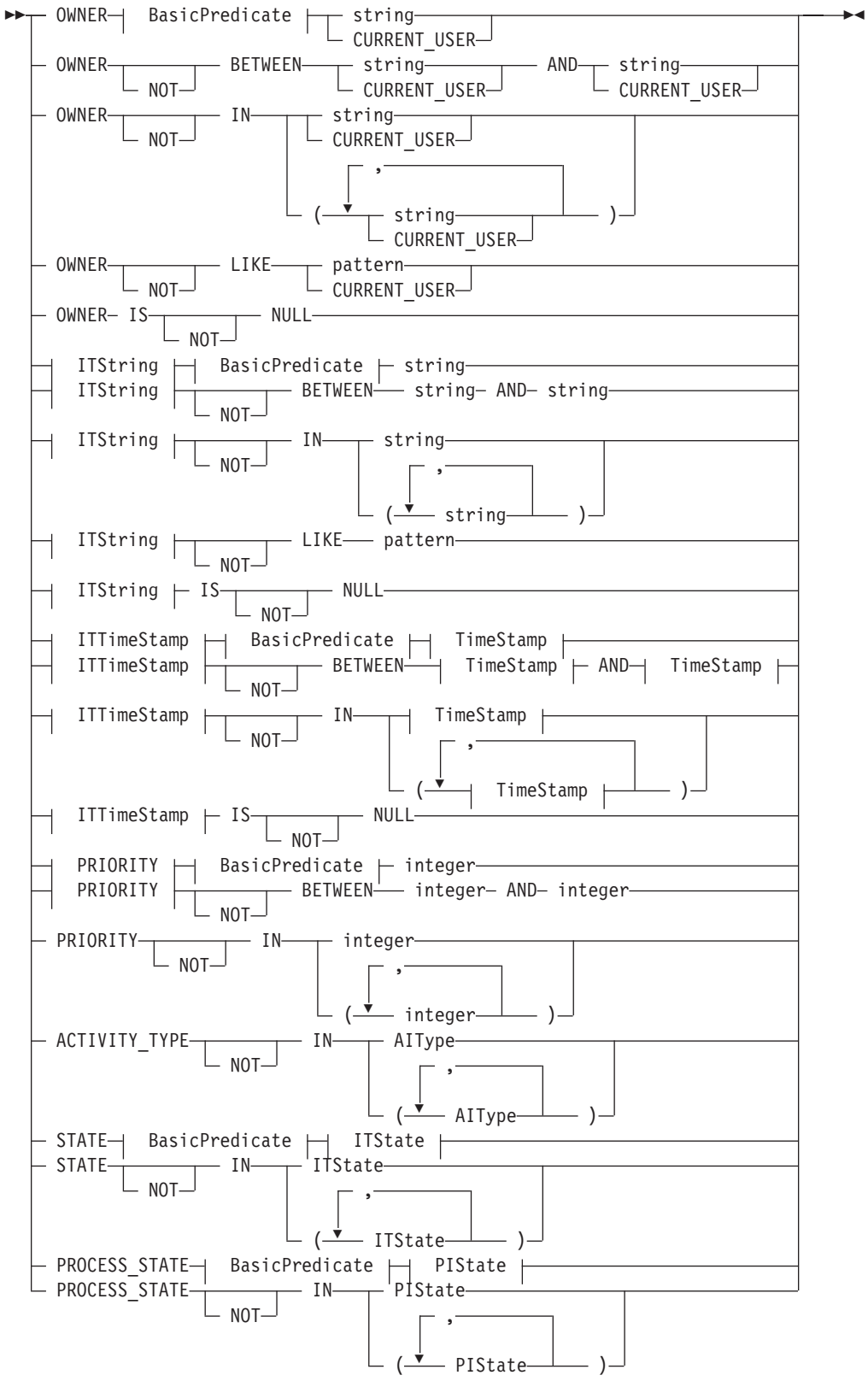
1. A *string* constant is to be enclosed in single quotes (').
2. A single quote within a string constant is to be doubled ('').
3. A *pattern* is a string constant in which the asterisk and the question mark have special meanings.
   - The question mark (?) represents any single character.
   - The asterisk (*) represents a string of zero or more characters.
   - The escape character is backslash (\) and must be used when the pattern itself contains actual question marks or asterisks. An actual backslash is to be doubled (\\).
4. A *TimeStamp* is a string constant, 24 hours based in local time.
5. Optional specifications in the *TimeStamp* are set to 0 (zero) if not specified.
6. It is not allowed to specify a percent sign (%) or an underscore (_) within a pattern for the LIKE operand, if this pattern contains mixed data. The usage of one of these letters results in an SQL error.

WIFilter



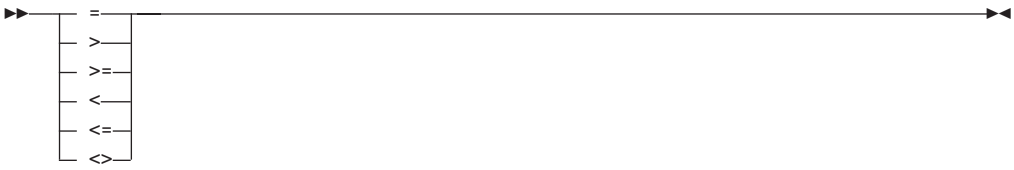
Action and activity implementation calls

ITPredicate



AIType

BasicPredicate



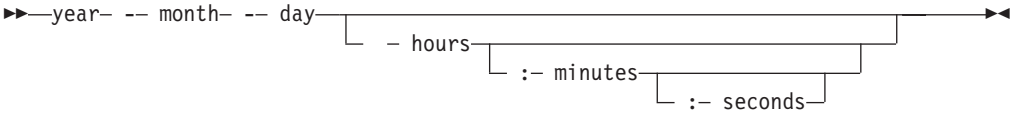
ITState

ITString

ITTimeStamp

PIState

TimeStamp



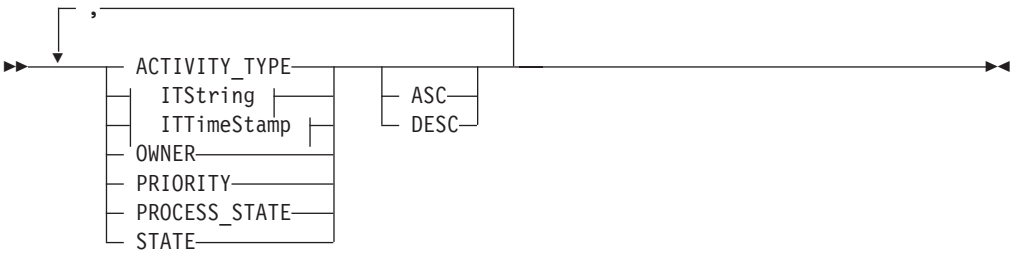
Work items can be sorted. A work item sort criterion is specified as a character string.

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.

WlOrderBy



The number of work items to be retrieved can be restricted via a threshold which specifies the maximum number of work items to be returned to the client. That threshold is applied after the items have been sorted according to the sort criteria specified. Note that the items are sorted on the server, that is, the code page of the server determines the sort sequence.

The primary information that is retrieved for each work item is:
- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind

- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorkitems(
       FmcjExecutionServiceHandle  service,
       char const *                filter,
       char const *                sortCriteria,
       unsigned long const *       threshold,
       FmcjWorkitemVectorHandle *  workitems )
```

C++ language signature

```
APIRET QueryWorkitems(
       string const *                filter,
       string const *                sortCriteria,
       unsigned long const *         threshold,
       vector<FmcjWorkitem> &        workitems ) const
```

Java signature

```
public abstract
WorkItem[] queryWorkItems(
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

COBOL

```
FmcjESQueryWorkitems.
   CALL    "FmcjExecutionServiceQueryWorkitems"
                  USING
                  BY VALUE
                     serviceValue
                     filter
                     sortCriteria
                     threshold
                  BY REFERENCE
                     workitems
                  RETURNING
                     intReturnValue.
```

Parameters

filter Input. The filter criteria which characterize the work items to be retrieved.

service Input. A handle to the service object representing the session with the execution server.

sortCriteria Input. The sort criteria to be applied to the work items found.

threshold Input. The threshold which defines the maximum number of work items to be returned to the client.

workitems Input/Output. The qualifying vector of work items.

Return type

APIRET The return code of calling this API call - see return codes below.

WorkItem[] The qualifying work items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_FILTER(125)
The specified filter is not applicable to work items.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are not applicable to work items.

FMC_ERROR_INVALID_THRESHOLD(807)
The specified threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of work items to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query process instances (C-language)" on page 601.
- For a C++ example see "Query process instances (C++)" on page 602.
- For a Java example see "Query process instances (Java)" on page 603.

You query work items similar to querying process instances.

QueryWorklists()

This API call retrieves the worklists the user has access to from the MQ Workflow execution server (action call).

In C, C++, and COBOL, any worklists retrieved are appended to the supplied vector. If you want to read the current worklists only, you have to clear the vector before you call this API call. This means that you should set the vector handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESQueryWorklists.
           CALL    "FmcjExecutionServiceQueryWorklists"
                         USING
                         BY VALUE
                            serviceValue
                         BY REFERENCE
                            lists
                         RETURNING
                            intReturnValue.
```

Parameters

lists Input/Output. The vector of worklists.

service Input. A handle to the service object representing the session with the execution server.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

WorkList[] The qualifying worklists.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of worklists to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query worklists (C-language)" on page 586.
- For a C++ example see "Query worklists (C++)" on page 588.
- For a Java example see "Query worklists (Java)" on page 590.

Receive()

This API call allows for receiving data pushed by an MQ Workflow execution server or for receiving a response on an asynchronous request.

A correlation ID can be used to receive a specific response. To receive any data sent, it must be a 0 (NULL) pointer or specify FMCJ_NO_CORRELID, that is, point to a buffer that contains all zeros (0x00). Note that the correlation ID is set on return provided that no 0 pointer is passed. This means that it has to be reset for each request.

The timeout value specifies how long the application should wait at a maximum for some data to arrive. If no data arrives, a timeout error is indicated. A timeout value of -1 indicates an indefinite wait time.

If data is successfully received, the execution data contains the data sent and can be used for updating objects or for creating new objects. See "ExecutionData" on page 273 for API calls supported by the execution data object.

The following enumeration types can be used to determine the contents of the execution data received:

C-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
COBOL	Fmc-DART-NotSet

Terminate(2) Indicates that receiving data can end.

C-language	Fmc_DART_Terminate
C++	FmcjExecutionData::Terminate
COBOL	Fmc-DART-Terminate

ItemDeleted(1000)
Indicates that a work item, an activity instance notification, or a process instance notification has been deleted.

C-language Fmc_DART_ItemDeleted

C++ FmcjExecutionData::ItemDeleted

COBOL Fmc-DART-ItemDeleted

Workitem(1002)
Indicates that a work item has been created or updated.

C-language Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

COBOL Fmc-DART-Workitem

ActivityInstanceNotification(1003)
Indicates that an activity instance notification has been created or updated.

C-language Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

COBOL Fmc-DART-ActInstNotif

ProcessInstanceNotification(1004)
Indicates that a process instance notification has been created or updated.

C-language Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

COBOL Fmc-DART-ProcInstNotif

ExecuteInstanceResponse(1100)
Indicates that the execution data contains the response on an ExecuteProcessInstance() request.

C-language Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

COBOL Fmc-DART-ExecuteInstResponse

ExecuteProgramResponse(1101)
Indicates that the execution data contains the response on an ExecuteProgram() request.

C-language Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

COBOL Fmc-DART-ExecuteProgResponse

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server (present session mode)

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA not supported

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjESReceive.

           CALL    "FmcjExecutionServiceReceive"
                          USING
                          BY VALUE
                             serviceValue
                          BY REFERENCE
                             correlID
                             data
                          BY VALUE
                             timeoutValue
                          RETURNING
                             intReturnValue.
```

Parameters

correlID Input/Output. The correlation ID by which this data can be correlated to a previous request. Must be a NULL (0) pointer or point to Fmcj_No_CorrelID if you want to receive any data.

data Output. The data sent by an MQ Workflow execution server.

service Input. A handle to the service object representing the present session with the execution server.

timeout Input. The maximum time period in milliseconds to wait for some data to arrive.

Return type

APIRET The return code of calling this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_MESSAGE_DATA(104)
The client received an unknown message.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetPersonAbsent()

This API call sets the absence indication of the specified user to the specified value (action call).

When a person is absent, this person does not participate in staff resolution, that is, this person does not get assigned any work items.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceSetPersonAbsent(
       FmcjExecutionServiceHandle  service,
       char const *                userID,
       bool                        newValue )
```

C++ language signature

```
APIRET SetPersonAbsent( string const * userID   = 0,
                        bool           newValue = true )
```

Java signature

```
public abstract
void   setPersonAbsent() throws FmcException

public abstract
void   setPersonAbsent2( String userID, boolean newValue )
throws FmcException
```

COBOL

```
        FmcjESSetPersonAbsent.
           CALL    "FmcjExecutionServiceSetPersonAbsent"
                         USING
                         BY VALUE
                            serviceValue
                            userID
                            newValue
                         RETURNING
                            intReturnValue.
```

Parameters

service Input. The handle of the service object where a person's absence is to be set.

newValue Input. True, if the person is denoted as absent, else false.

userID Input. The user ID of the person whose absence is to be set. When no user ID is provided in C++, the absence of the logged-on user is set.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_USERID_UNKNOWN(10)
The specified user ID is not known.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

TerminateReceive()

This API call causes information to be placed into the client input queue to tell that receiving data from an MQ Workflow execution server can end.

In this way, the receiving part of the application gets to know that receiving data can end. Any resulting actions are up to the application.

When the correlID parameter points to some buffer initialized to FMCJ_NO_CORRELID, that is, points to a buffer that contains all zeros (0x00), then a correlation ID is returned which can be used to explicitly receive this data.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

None

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA not supported

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjExecutionServiceTerminateReceive(
       FmcjExecutionServiceHandle  service,
       FmcjCorrelID *              correlID )
```

C++ language signature

```
APIRET TerminateReceive( FmcjCorrelID * correlID = 0 )
```

COBOL

```
        FmcjESTerminateReceive.
           CALL    "FmcjExecutionServiceTerminateReceive"
                         USING
                         BY VALUE
                            serviceValue
                         BY REFERENCE
                            correlID
                         RETURNING
                            intReturnValue.
```

Parameters

correlID Input/Output. The correlation ID by which this request can be correlated.

service Input. A handle to the service object.

Return type

APIRET The return code of calling this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_CORRELATION_ID(506)
The correlation ID passed is not FMCJ_NO_CORRELID.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Instance monitor actions

An InstanceMonitor object represents a monitor for a process instance, an activity instance of type *Block*, or an activity instance of type *Process*.

The following sections describe the actions which can be applied on an instance monitor. See "InstanceMonitor" on page 284 for a complete list of API calls.

ObtainInstanceMonitor()/ ObtainBlockMonitor()/ ObtainProcessMonitor()

This API call retrieves the instance monitor for the specified activity instance from the MQ Workflow execution server (action call). If the requested instance monitor has already been retrieved from the server, the monitor found in the API cache is returned to the caller.

When the monitor for a process instance is retrieved, the specified activity instance must be of type *Process* and be part of this instance monitor.

When the monitor for a block is retrieved, the specified activity instance must be of type *Block* and be part of this instance monitor.

When the deep option is specified, nested activity instances of type *Block* are resolved, that is, their instance monitors are also read from the server.

Note: Deep is currently not supported.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.InstanceMonitor
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
FmcjInstanceMonitorHandle FMC_APIENTRY
    FmcjInstanceMonitorObtainBlockMonitor(
                        FmcjInstanceMonitorHandle  hdlMonitor,
                        FmcjActivityInstanceHandle activity  )

FmcjInstanceMonitorHandle FMC_APIENTRY
    FmcjInstanceMonitorObtainProcessMonitor(
                        FmcjInstanceMonitorHandle  hdlMonitor,
                        FmcjActivityInstanceHandle activity,
                        bool                       deep      )
```

C++ language signature

```
FmcjInstanceMonitor *
       ObtainBlockMonitor ( FmcjActivityInstance const & activity ) const

APIRET ObtainBlockMonitor ( FmcjActivityInstance const & activity,
                            FmcjInstanceMonitor &        monitor ) const

FmcjInstanceMonitor *
       ObtainProcessMonitor(FmcjActivityInstance const & activity,
                            bool                         deep = false) const

APIRET ObtainProcessMonitor(FmcjActivityInstance const & activity,
                            FmcjInstanceMonitor &        monitor,
                            bool                         deep = false) const
```

Java signature

```
public abstract
InstanceMonitor obtainBlockMonitor  ( ActivityInstance activity )
                throws FmcException

public abstract
InstanceMonitor obtainProcesskMonitor( ActivityInstance activity,
                                       boolean          deep    )
                throws FmcException
```

COBOL

```
FmcjIMObtainBlockMon.
    CALL    "FmcjInstanceMonitorObtainBlockMonitor"
                   USING
                   BY VALUE
                      hdlMonitor
                      activity
                   RETURNING
                      intReturnValue.

FmcjIMObtainProcMon.
    CALL    "FmcjInstanceMonitorObtainProcessMonitor"
                   USING
                   BY VALUE
                      hdlMonitor
                      activity
                      deep
                   RETURNING
                      intReturnValue.
```

Parameters

activity Input. The activity instance of type *Block* or *Process* whose instance monitor is to be retrieved.

deep Input. An indicator whether monitors of activity instances of type *Block* are to be resolved, that is, their monitors are also to be retrieved. Note that deep is currently not supported.

hdlMonitor Input. The instance monitor containing the activity instance of type *Block* or *Process*.

monitor Input/Output. The instance monitor retrieved.

returnCode Input/Output. The result of calling this API call - see return codes below.

Return type

APIRET The result of calling this API call - see return codes below.

FmcjInstanceMonitor*/Handle/InstanceMonitor
The instance monitor respectively a pointer of handle to the instance monitor.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The specified activity instance is not described by the instance monitor or does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_KIND(501)
The specified activity instance is not of type *Block* or *Process*.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the instance monitor from the MQ Workflow execution server (action call).

All information about the instance monitor is retrieved.

When the deep option is specified, then activity instances of type *Block* are resolved, that is, their instance monitors are also refreshed from the server.

Note: Deep is currently not supported.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.InstanceMonitor

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjInstanceMonitorRefresh(
                        FmcjInstanceMonitorHandle  hdlMonitor,
                        bool                       deep      )
```

C++ language signature

```
APIRET Refresh( bool deep = false )
```

Java signature

```
public abstract
void Refresh( boolean deep ) throws FmcException
```

COBOL

```
       FmcjIMRefresh.
           CALL    "FmcjInstanceMonitorRefresh"
                           USING
                           BY VALUE
                              hdlMonitor
                              deep
                           RETURNING
                              intReturnValue.
```

Parameters

deep Input. An indicator whether activity instances of type *Block* are to be resolved, that is, their monitors are also to be provided. Note, deep is currently ignored.

hdlMonitor Input. The instance monitor to be refreshed.

Return type

long The result of calling this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 and COBOL, common implementations of functions are taken from FmcjItem. That is, common functions start with the prefix FmcjItem; they are also defined starting with the prefixes FmcjWorkitem, FmcjActivityInstanceNotification, and FmcjProcessInstanceNotification.

An item is uniquely identified by its object identifier.

The following sections describe the actions which can be applied on an item. See "Item" on page 285 for a complete list of API calls.

Delete()

This API call deletes the specified item from the MQ Workflow execution server (action call).

A notification can always be deleted. A work item must be in states *Ready*, *Finished*, *ForceFinished*, or *Disabled*. If the work item is in the *Ready* state and represents the only work associated with the activity instance and when the associated process instance is not *Terminating* or *Terminated*, then deletion is rejected.

There are no impacts on the transient representation of your item; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Be the item owner
- Work item authorization for the item owner
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemDelete.
           CALL    "FmcjItemDelete"
                          USING
                          BY VALUE
                             hdlItem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlItem Input. The handle of the item to be deleted.

Return type
long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.
FMC_ERROR_NOT_ALLOWED(507)
The item represents the only work associated with the activity instance.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The item is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()

This API call retrieves the instance monitor for the process instance the item is part of from the MQ Workflow execution server (action call).

When the deep option is specified, then activity instances of type Block are resolved, that is, their monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the instance monitor handle already points to some object.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjItemObtainProcessMonitor(
                    FmcjItemHandle               hdlItem,
                    bool                         deep,
                    FmcjInstanceMonitorHandle *  monitor)

#define FmcjActivityInstanceNotificationObtainProcessMonitor
        FmcjItemObtainProcessMonitor
#define FmcjProcessInstanceNotificationObtainProcessMonitor
        FmcjItemObtainProcessMonitor
#define FmcjWorkitemObtainProcessMonitor
        FmcjItemObtainProcessMonitor
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false ) const
```

Java signature

```
public abstract
InstanceMonitor obtainProcesseMonitor( boolean deep )
throws FmcException
```

COBOL

```
        FmcjItemObtainProcMon.
           CALL    "FmcjItemObtainProcessMonitor"
                          USING
                          BY VALUE
                             hdlItem
                             deep
                          BY REFERENCE
                             monitor
                          RETURNING
                             intReturnValue.
```

Parameters

deep Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is currently ignored.

hdlItem Input. The item whoseinstance 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 API call - see return codes below.

InstanceMonitorHandle*/ InstanceMonitor

A pointer to the instance monitor or the instance monitor the item is a part of.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ProcessInstance()

This API call retrieves the process instance the item is a part of from the MQ Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

In C++, when the process instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the process instance handle already points to some object.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemProcInst.
           CALL    "FmcjItemProcessInstance"
                          USING
                          BY VALUE
                             hdlItem
                          BY REFERENCE
                             instance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlItem Input. The handle of the item object to be queried.

instance Input/Output. The process instance object to be retrieved (initialized).

returnCode Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance*/ ProcessInstance

A pointer to the process instance or the process instance the item is a part of.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the item from the MQ Workflow execution server (action call).

All information about the item, primary and secondary, is retrieved.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Item

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemRefresh.
           CALL    "FmcjItemRefresh"
                           USING
                           BY VALUE
                              hdlItem
                           RETURNING
                              intReturnValue.
```

Parameters

hdlItem Input. The handle of the item object to be refreshed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This API call sets the description of the item to the specified value (action call).

If no description is provided, the description of the item is reset to the description of the associated activity instance or process instance.

The following rules apply for specifying an item description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Be the item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemSetDescription.
           CALL    "FmcjItemSetDescription"
                           USING
                           BY VALUE
                              hdlItem
                              description
                           RETURNING
                              intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlItem Input. The handle of the item object whose description is to be set.

isNull Input. If set to *True*, indicates that any description of the item is to be reset.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.
FMC_ERROR_INVALID_DESCRIPTION(810)
The description does not conform to the syntax rules.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetName()

This API call sets the name of the item (action call).

If no name is provided, the name of the item is reset to its default, the activity instance respectively the process instance name.

The following rules apply for specifying a work item or activity instance notification name:

- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale, except the following:
  `! " ' ( ) * + , - . / : ; < = > [ \ ] ^`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.
- You cannot use leading digits.
- You cannot use keywords AND, OR, NOT, IS, NULL, MOD, LOWER, UPPER, VALUE, SUBSTR, _BLOCK

The following rules apply for specifying a process instance notification name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:

```
* ? " ; : . $
```

- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemSetName.
           CALL    "FmcjItemSetName"
                          USING
                          BY VALUE
                             hdlItem
                             name
                          RETURNING
                             intReturnValue.
```

Parameters

hdlItem Input. The handle of the item to be dealt with.

name Input. The new name of the item; can be a NULL (0) pointer or null object (Java).

Return type
long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.
FMC_ERROR_INVALID_NAME(134)
The name does not conform to the syntax rules.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Transfer()

This API call transfers an item to the specified user (action call).

Notifications can always be transferred. A work item must be in states *Ready*, *InError*, *Executed*, *Suspending*, *Suspended*, or *Terminated* and the associated process instance in states *Running*, *Suspending*, or *Suspended*. Work items in states InError or Terminated can only be transferred to the process administrator.

The user who transfers the item must be the owner of the item or have work item authorization for the owner of the item and have work item authorization for the new owner.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjItemTransfer.
           CALL    "FmcjItemTransfer"
                         USING
                         BY VALUE
                            hdlItem
                            userID
                         RETURNING
                            intReturnValue.
```

Parameters

hdlItem Input. The handle of the item object to be transferred.

userID Input. The ID of the user to whom the item is to be transferred.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The item does no longer exist.

FMC_ERROR_NEW_OWNER_ABSENT(110)
The user to whom the item is to be transferred is absent, that is, the item is not transferred.

FMC_ERROR_NEW_OWNER_NOT_FOUND(107)
The user to whom the item is to be transferred is unknown.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_OWNER_ALREADY_ASSIGNED(133)
The user to whom the item is to be transferred does already have that item.

FMC_ERROR_WRONG_STATE(120)
The item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefixes FmcjProcessInstanceList, FmcjProcessTemplateList, and FmcjWorklist.

A persistent list is uniquely identified by its name, type, and owner. It can be defined for general access purposes; it is then of a *public* type. Or, it can be defined for some specific user; it is then of a *private* type.

The following sections describe the actions which can be applied on a persistent list. See “PersistentList” on page 288 for a complete list of API calls.

Delete()

This API call deletes the specified persistent list from the MQ Workflow execution server (action call).

The transient representation of the persistent list is not impacted; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note

- See “Action API calls” on page 133 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

C-language fmcjcrun.h

Action and activity implementation calls

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPLDelete.
           CALL    "FmcjPersistentListDelete"
                          USING
                          BY VALUE
                             hdlList
                          RETURNING
                             intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list to be deleted.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh

This API call refreshes the persistent list from the MQ Workflow execution server (action call).

All information about the persistent list is retrieved, for example, its description, its filter, or its sort criteria.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjPersistentListRefresh( FmcjPersistentListHandle hdlList )

#define FmcjProcessInstanceListRefresh FmcjPersistentListRefresh
#define FmcjProcessTemplateListRefresh FmcjPersistentListRefresh
#define FmcjWorklistRefresh            FmcjPersistentListRefresh
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

COBOL

```
        FmcjPLRefresh.
           CALL    "FmcjPersistentListRefresh"
                         USING
                         BY VALUE
                            hdlList
                         RETURNING
                            intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list to be refreshed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This API call sets the description of the persistent list to the specified value (action call).

If no description is provided, the description of the persistent list is erased.

The following rules apply for specifying a persistent list description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPLSetDescription.
           CALL    "FmcjPersistentListSetDescription"
                         USING
                         BY VALUE
                            hdlList
                            description
                         RETURNING
                            intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlList Input. The handle of the persistent list object whose description is to be set.

isNull Input. If set to *True*, indicates that any description is to be removed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.
FMC_ERROR_INVALID_DESCRIPTION(810)
The description does not conform to the syntax rules.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetFilter()

This API call sets the filter of the persistent list to the specified value (action call).

If no filter is provided, the current filter of the persistent list is erased. This means that all objects authorized for will be selected via this list.

Refer to the appropriate list creation for a description of a valid filter syntax.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

Action and activity implementation calls

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPLSetFilter.
           CALL    "FmcjPersistentListSetFilter"
                          USING
                          BY VALUE
                             hdlList
                             filter
                          RETURNING
                             intReturnValue.
```

Parameters

filter Input. The filter or a pointer to the filter to be set; can be a NULL (0) pointer or null object (Java).

hdlList Input. The handle of the persistent list object whose filter is to be set.

isNull Input. If set to *True*, indicates that any filter is to be removed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.
FMC_ERROR_INVALID_FILTER(125)
The specified filter is invalid.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetSortCriteria()

This API call sets the sort criteria of the persistent list to the specified value (action call).

If no sort criteria are provided, the current sort criteria of the persistent list are erased. This means that objects selected via this list will not be sorted.

Refer to the appropriate list creation for a description of a valid sort criteria syntax.

Usage note
- See "Action API calls" on page 133 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

C-language fmcjcrun.h

Action and activity implementation calls

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPLSetSortCriteria.
           CALL    "FmcjPersistentListSetSortCriteria"
                         USING
                         BY VALUE
                            hdlList
                            sortCriteria
                         RETURNING
                            intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list object whose sort criteria are to be set.

sortCriteria Input. The sort criteria or a pointer to the sort criteria to be set; can be a NULL (0) pointer or null object (Java).

isNull Input. If set to *True*, indicates that any sort criteria are to be removed.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.

FMC_ERROR_INVALID_SORT(808)
The specified sort criteria are invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetThreshold()

This API call sets the threshold of the persistent list to the specified value (action call).

If no threshold is provided, the threshold of the persistent list is erased. This means that all objects contained in the list will be provided when queried.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Be the owner of the list
- Staff definition
- Be the system administrator

Required connection

MQ Workflow execution server

Action and activity implementation calls

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.PersistentList

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPLSetThreshold.
           CALL    "FmcjPersistentListSetThreshold"
                         USING
                         BY VALUE
                            hdlList
                            threshold
                         RETURNING
                            intReturnValue.
```

Parameters

hdlList Input. The handle of the persistent list object whose threshold is to be set.

threshold Input. The threshold or a pointer to the threshold to be set; can be a NULL (0) pointer or null object (Java).

isNull Input. If set to *True*, indicates that any threshold is to be erased.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The persistent list does no longer exist.

FMC_ERROR_INVALID_THRESHOLD(807)
The threshold is invalid.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 289 for a complete list of API calls.

Refresh()

This API call refreshes the person from the MQ Workflow execution server (action call).

All information about the person, primary and secondary, is retrieved.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Action and activity implementation calls

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
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPRefresh.
           CALL    "FmcjPersonRefresh"
                         USING
                         BY VALUE
                            hdlPerson
                         RETURNING
                            intReturnValue.
```

Parameters

hdlPerson Input. The handle of the person to be refreshed.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetAbsence()

This API call sets the absence indication of the logged-on user to the specified value (action call).

When a person is absent, this person does not participate in staff resolution, that is, this person does not get assigned any work items.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX IBM MQSeries Workflow Control 3.1

C-language fmcjcrun.h

Action and activity implementation calls

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Person

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPSetAbsence.
           CALL    "FmcjPersonSetAbsence"
                         USING
                         BY VALUE
                            hdlPerson
                            newValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlPerson Input. The handle of the person object whose absence is to be set.

newValue Input. True, if the person is denoted as absent, else false.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetSubstitute()

This API call sets the substitute of the logged-on user (action call).

The substitute must be a registered MQ Workflow user ID other than the logged-on user. If no substitute is provided, the substitute of the logged-on user is erased.

Note: Changing the substitute can result in changes of persons who are authorized to access the (work) items of the logged-on user. You must refresh the person object to read the updated definitions.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Be the same user, that is, request to change the own absence
- Staff authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

Action and activity implementation calls

JAVA com.ibm.workflow.api.Person

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjPersonSetSubstitute(
                    FmcjPersonHandle  hdlPerson,
                    char const *      substitute )
```

C++ language signature

```
APIRET SetSubstitute( string const * substitute )
```

Java signature

```
public abstract
void   setSubstitute( String substitute ) throws FmcException
```

COBOL

```
        FmcjPSetSubstitute.
           CALL    "FmcjPersonSetSubstitute"
                         USING
                         BY VALUE
                            hdlPerson
                            substitute
                         RETURNING
                            intReturnValue.
```

Parameters

hdlPerson Input. The handle of the person object whose substitute is to be set.

isNull Input. If set to *True*, any substitute specification is removed.

substitute Input. The substitute or a pointer to the substitute to be set; can be a NULL (0) pointer or null object (Java).

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_INVALID_USER(132)
The specified user ID does not correspond to the syntax rules or the user cannot be logged on and be the substitute at the same time.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_USERID_UNKNOWN(10)
The specified user ID is not a registered MQ Workflow user ID.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Process instance actions

An FmcjProcessInstance or a ProcessInstance object represents an instance of a process template. A process instance is uniquely identified by its object identifier or by its name. Depending on the keep option when the process instance was created, the unique process instance name has been supplied by the user or has been generated by MQ Workflow.

The following diagram provides an overview on the possible process instance states and the actions which are allowed in those states, provided that the appropriate authority has been granted and that more specific requirements stated in the API calls descriptions have been fulfilled.

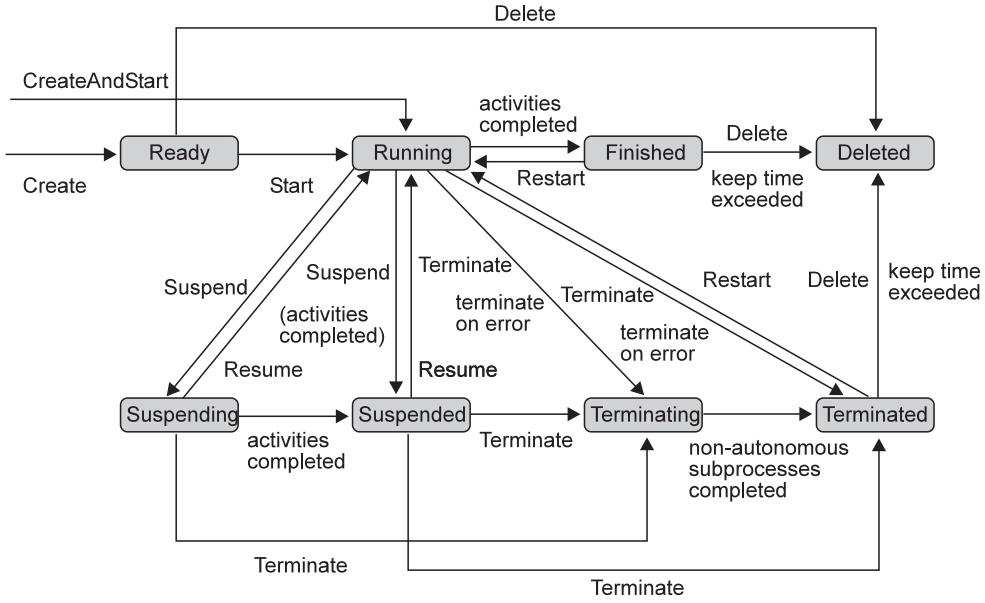


Figure 41. Process instance states

The following sections describe the actions which can be applied on a process instance. See "ProcessInstance" on page 294 for a complete list of API calls.

Delete()

This API call deletes the specified process instance from the MQ Workflow execution server (action call).

The process instance must be a top-level process and in states *Ready*, *Finished*, or *Terminated*. The creator can delete the process instance as long as it has not been started.

There are no impacts on your transient representation of the process instance; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note

- See "Action API calls" on page 133 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

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPIDelete.
           CALL    "FmcjProcessInstanceDelete"
                          USING
                          BY VALUE
                             hdlInstance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance to be deleted.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the process instance from the MQ Workflow execution server (action call).

In C++, when the container object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the container handle already points to some object.

Usage note
- See "Action API calls" on page 133 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
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPIInCtnr.
           CALL    "FmcjProcessInstanceInContainer"
                         USING
                         BY VALUE
                            hdlInstance
                         BY REFERENCE
                            inputValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object whose input container is to be retrieved.

input Input/Output. The address of the input container or of its handle respectively the input container of the process instance to be set.

Return type

long/ APIRET The result of calling this API call - see return codes below.

ReadWriteContainer
The input container of the process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ObtainProcessMonitor()

This API call obtains a monitor for the process instance from the MQ Workflow execution server (action call).

When the deep option is specified, then activity instances of type Block are resolved, that is, their block instance monitors are also fetched from the server.

Note: Deep is currently not supported.

In C++, when the instance monitor object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the instance monitor handle already points to some object.

Usage note

- See "Action API calls" on page 133 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
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceObtainProcessMonitor(
                    FmcjProcessInstanceHandle   hdlInstance,
                    bool                        deep,
                    FmcjInstanceMonitorHandle * monitor )
```

C++ language signature

```
APIRET ObtainProcessMonitor( FmcjInstanceMonitor & monitor,
                             bool                  deep= false )
```

Java signature

```
public abstract
InstanceMonitor obtainProcessMonitor( boolean deep ) throws FmcException
```

COBOL

```
        FmcjPIObtainProcMon.
           CALL    "FmcjProcessInstanceObtainProcessMonitor"
                          USING
                          BY VALUE
                             hdlInstance
                             deep
                          BY REFERENCE
                             monitor
                          RETURNING
                             intReturnValue.
```

Parameters

deep Input. An indicator whether activity instances of type Block are to be resolved, that is, their monitor is also to be provided. Note, deep is currently ignored.

hdlInstance Input. The handle of the process instance object whose monitor is to be retrieved.

monitor Input/Output. The address of the monitor handle respectively the monitor of the process instance to be set.

returnCode Input/Output. A pointer to the result of the method call - see return codes below.

Return type

APIRET The return code of calling this API call - see return codes below.

InstanceMonitor*/ProcessInstanceMonitor
A pointer to the instance monitor respectively the instance monitor.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the process instance from the MQ Workflow execution server (action call).

In C++, when the container object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the container handle already points to some object.

Usage note
- See "Action API calls" on page 133 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
COBOL	fmcvars.cpy, fmcperf.cpy

ActiveX signature

```
long OutContainer( Container * output )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessInstanceOutContainer(
       FmcjProcessInstanceHandle     hdlInstance,
       FmcjReadOnlyContainerHandle * output )
```

C++ language signature

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java signature

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

COBOL

```
FmcjPIOutCtnr.
   CALL    "FmcjProcessInstanceOutContainer"
                  USING
                  BY VALUE
                     hdlInstance
                  BY REFERENCE
                     outputValue
                  RETURNING
                     intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object whose output container is to be retrieved.

output Input/Output. The address of the output container or of its handle respectively the output container of the process instance to be set.

Return type

long/ APIRET The result of calling this API call - see return codes below.

ReadOnlyContainer

The output container of the process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)

The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)

The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)

Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)

Not logged on.

FMC_ERROR_COMMUNICATION(13)

The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)

An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)

A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)

Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)

An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the process instance from the MQ Workflow execution server (action call).

All information about the process instance, primary and secondary, is retrieved.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRefresh( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

COBOL

```
FmcjPIRefresh.
   CALL    "FmcjProcessInstanceRefresh"
                  USING
                  BY VALUE
                     hdlInstance
                  RETURNING
                     intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be refreshed.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This API call restarts the process instance on the MQ Workflow execution server (action call).

Only *finished* or *terminated* top-level process instances can be restarted. The process administrator does not change. The process starter is set to the requester of this API call.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRestart( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Restart()
```

Java signature

```
public abstract
void   restart() throws FmcException
```

COBOL

```
        FmcjPIRestart.
           CALL    "FmcjProcessInstanceRestart"
                         USING
                         BY VALUE
                            hdlInstance
                         RETURNING
                            intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be restarted.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The process instance is no top-level process instance.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Resume()

This API call resumes processing of a suspended or suspending process instance (action call).

All non-autonomous subprocesses with respect to control autonomy are also resumed, if the deep option is true.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPIResume.
           CALL    "FmcjProcessInstanceResume"
                          USING
                          BY VALUE
                             hdlInstance
                             deep
                          RETURNING
                             intReturnValue.
```

Parameters

deep	Input. If deep is true, processing of all non-autonomous subprocesses is also resumed.
hdlInstance	Input. The handle of the process instance to be resumed.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetDescription()

This API call sets the description of the process instance to the specified value (action call).

If no description is provided, the description of the process instance is erased.

The following rules apply for specifying a process instance description:

- You can specify a maximum of 254 characters.
- You can use any printable characters depending on your current locale, including the end-of-line and new-line characters.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPISetDescription.
           CALL    "FmcjProcessInstanceSetDescription"
                          USING
                          BY VALUE
                             hdlInstance
                             description
                          RETURNING
                             intReturnValue.
```

Parameters

description Input. The description or a pointer to the description to be set; can be a NULL (0) pointer or null object (Java).

hdlInstance Input. The handle of the process instance object whose description is to be set.
isNull Input. If set to *True*, any description is removed.

Return type
long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.
FMC_ERROR_INVALID_DESCRIPTION(810)
The description does not conform to the syntax rules.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetName()

This API call sets the name of the process instance to the specified value (action call).

The process instance must still be in the *Ready* state.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:

  `* ? " ; : . $`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
FmcjPISetName.
   CALL    "FmcjProcessInstanceSetName"
                  USING
                  BY VALUE
                     hdlInstance
                     name
                  RETURNING
                     intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object whose name is to be set.

name Input. The name to be set.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_INVALID_NAME(134)
The name does not conform to the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_NOT_UNIQUE(121)
The process instance name is not unique.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Start()

This API call starts a ready process instance (action call).

When successfully executed, the starter is set to the requestor of this action and the process administrator is determined.

When initial values are to be passed to the process instance to be started, an input container can be provided (see also FmcjProcessInstance:: InContainer()). When the process instance requires input and is started without specifying an input container, the input-container values are not set. So, when, for example, input-container values are queried from within an activity implementation, FMC_ERROR_MEMBER_NOT_SET is returned.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPIStart.
           CALL    "FmcjProcessInstanceStart"
                         USING
                         BY VALUE
                            hdlInstance
                            inputValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be started.

input Input. The input container of the process instance.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_INVALID_CONTAINER(509)
The passed container is invalid for the process instance; wrong schema or version.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Suspend()

This API call suspends (temporarily stops) the process instance (action call).

The process instance must be in state *Running*. All non-autonomous subprocesses with respect to control autonomy are also suspended if the deep option is true. Autonomous subprocesses are not considered.

The process instance remains in state *Suspending* as long as there are running program activity implementations, suspending non-autonomous subprocesses, or checked-out work items. When the activity implementations completed their executions and the non-autonomous subprocesses reached the *Suspended* state, and when the checked-out work items are checked in, the process instance is put into the *Suspended* state.

Optionally, a date may be specified up to when the process instance is suspended. The date is to be specified in local time. The process instance is then automatically resumed, together with the non-autonomous subprocesses, if the deep option had been specified.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process administration authorization
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

Action and activity implementation calls

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPISuspend.
           CALL    "FmcjProcessInstanceSuspend"
                          USING
                          BY VALUE
                             hdlInstance
                             deep
                          RETURNING
                             intReturnValue.
```

Parameters

deep	Input. An indicator whether also non-autonomous subprocesses are to be suspended.
hdlInstance	Input. The handle of the process instance object to be suspended.
time	Input. The date/time respectively a pointer to the date/time up to when the process instance is to be suspended.

Return type

long/ APIRET	The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This API call terminates a process instance and all of its non-autonomous subprocesses (action call).

The process instance must be in states *Running*, *Suspended*, or *Suspending*.

The process instance is put into state terminating as long as there are terminating non-autonomous subprocesses. When the non-autonomous subprocesses terminated, the process instance is put into the *Terminated* state. When the process instance has reached the *Terminated* state, it is deleted depending on the setting of the "delete finished items" option.

Usage note

- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessInstanceTerminate( FmcjProcessInstanceHandle hdlInstance )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

COBOL

```
        FmcjPITerminate.
           CALL    "FmcjProcessInstanceTerminate"
                          USING
                          BY VALUE
                             hdlInstance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlInstance Input. The handle of the process instance object to be terminated.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_STATE(120)
The process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjProcessInstanceList.

The following sections describe the actions which can be applied on a process instance list. See "ProcessInstanceList" on page 298 for a complete list of API calls.

QueryProcessInstances()

This API call retrieves the primary information for all process instances characterized by the specified process instance list from the MQ Workflow execution server (action call).

From the set of qualifying process instances, only those are retrieved the user is authorized for. The user is authorized for a process instance if the process instance:
- Does not belong to any category
- Does belong to a category and the user has global process authorization or global process administration authorization or selected process authorization or selected process administration authorization for that category

The primary information that is retrieved for each process instance is:
- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- StartTime
- State
- SuspensionExpirationTime
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

In C, C++, and COBOL, any process instances retrieved are appended to the supplied vector of process instances. If you want to read those process instances only which are currently included in the process instance list, you have to clear the vector before you call this API call.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessInstanceList

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPILQueryProcInsts.
           CALL    "FmcjProcessInstanceListQueryProcessInstances"
                          USING
                          BY VALUE
                             hdlList
                          BY REFERENCE
                             instances
                          RETURNING
                             intReturnValue.
```

Parameters

hdlList Input. The handle of the process instance list to be queried.

instances Input/Output. The vector of qualifying process instances.

Return type

long/ APIRET The result of calling this API call - see return codes below.

ProcessInstance[]
The qualifying process instances.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process instance list does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instances to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query worklists (C-language)" on page 586
- For a C++ example see "Query worklists (C++)" on page 588

Process template actions

An FmcjProcessTemplate or a ProcessTemplate object is the frozen state of a process model from which it is created via translation. All program definitions and data structures referenced by the process model are copied into the process template (early binding). Subprocesses are bound lately. Their definitions are only located during execution.

A process template is uniquely identified by its object identifier or by its name and a valid-from date. This *valid-from date* determines since when the process template can be used to create process instances.

When process templates are queried from the execution server, then only currently valid process templates are returned.

The following sections describe the actions which can be applied on a process template. See “ProcessTemplate” on page 300 for a complete list of API calls.

CreateAndStartInstance()

This API call creates a process instance from the specified process template and starts the resulting process instance (action call).

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:

  ```
  * ? " ; : . $
  ```
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQ Workflow, the following applies:

- If no or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged; FmcjProcessInstance::Name() returns the actual name of the process instance created. The newly created process instance contains the primary attribute values only.

When initial values are to be passed to the process instance to be created and started, an input container can be provided - see also FmcjProcessTemplate::InContainer(). When a process instance that requires input is started without specifying an input container, the input-container values are not set. When, for example, input-container values are queried from within an activity implementation, FMC_ERROR_MEMBER_NOT_SET is returned.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

See createAndStartInstance; additionally allows to pass an input container.

Usage note

- See “Action API calls” on page 133 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Action and activity implementation calls

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessTemplate

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPTCreateAndStartInst.
           CALL    "FmcjProcessTemplateCreateAndStartInstance"
                          USING
                          BY VALUE
                             hdlTemplate
                             name
                             reserved1
                             reserved2
                             inputValue
                             keepName
                          BY REFERENCE
                             newInstance
                          RETURNING
                             intReturnValue.
```

XML message

```
<!-- ProcessTemplateCreateAndStart =================== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
          ( ProcTemplName,
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgInstName              (#PCDATA) >
<!ELEMENT KeepName                  (#PCDATA) >
               <!--  Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
```

XML message continued

```
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
          ( ProcessInstance
          | Exception ) >
<!ELEMENT ProcessInstance
          ( ProcInstID,
            ProcInstName,
            ProcInstParentName?,
            ProcInstTopLevelName,
            ProcInstDescription?,
            ProcInstState,
            LastStateChangeTime,
            LastModificationTime,
            ProcTemplID,
            ProcTemplName,
            Icon,
            Category? ) >
```

XML message continued

```
<!ELEMENT ProcInstID                (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcInstState             (#PCDATA)  >
              <!-- Expected values: {Ready,Running,Finished,Terminated,
                                     Suspended, Terminating,
                                     Suspending,Deleted} -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >
<!ELEMENT ProcTemplID               (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
          <!-- Message text is optional, as it will be ignored
               in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
          (Parameter*) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
```

XML message continued

```
<!--========================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ========================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

Parameters

hdlTemplate Input. The handle of the process template object to be used.

input Input. The input container of the process instance.

keepName Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated.

name Input. The name of the process instance to be created and started.

nameIsNull Input. Indicates whether a name is specified for the process instance to be created and started.

newInstance Input/Output. The newly created and started process instance.

returnCode Input/Output. The result of calling this method - see below.

reserved1/reserved2
Input. Pass a 0 (NULL) pointer or an empty string.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance*/ ProcessInstance
A pointer to the newly created and started process instance respectively the newly created and started process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_INVALID_CONTAINER(509)
The passed container is invalid for the process instance to be started; wrong schema or version.

FMC_ERROR_INVALID_NAME(134)
The specified process instance name does not comply with the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_NOT_UNIQUE(121)
The name of the process instance is not unique.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.

FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.
FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
The type of the container is incorrect.
FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
The specified data member is not part of the container.
FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
The type of the data member value passed is incorrect.

XML example - MQ Workflow XML request sent to MQ Workflow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateCreateAndStartInstance>
   <ProcTemplName>OnlineCreditRequest</ProcTemplName>
   <ProgInstName>Credit Request #658321<ProgInstName>
   <KeepName>true</KeepName>
   <ProcInstInputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
   </ProcInstInputData>
 </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

XML example - MQ Workflow response sent from MQ Workflow to the reply queue:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateCreateAndStartInstanceResponse>
    <ProcessInstance>
      <ProcInstID>42424242EFEFEFEF</ProcInstID>
      <ProcInstName>Credit Request#658321</ProcInstName>
      <ProcInstTopLevelName>Credit Request#658321</ProcInstTopLevelName>
      <ProcInstDescription>Sample description</ProcInstDescription>
      <ProcInstState>Finished</ProcInstState>
      <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
      <LastModificationTime>1999-05-19 23:40:00</LastModTime>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
 </ProcessTemplateCreateAndStartInstanceResponse>
</WfMessage>
```

CreateInstance()

This API call creates a process instance from the specified process template (action call).

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:

  `* ? " ; : . $`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQ Workflow, the following applies:

- If no name or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged; FmcjProcessInstance::Name() returns the actual name of the process instance created. The newly created process instance contains the primary attribute values only.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

Usage note

- See “Action API calls” on page 133 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPTCreateInst.
           CALL    "FmcjProcessTemplateCreateInstance"
                         USING
                         BY VALUE
                            hdlTemplate
                            name
                            reserved1
                            reserved2
                            keepName
                         BY REFERENCE
                            newInstance
                         RETURNING
                            intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object to be used.

keepName Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated.

name Input. The name of the process instance to be created.

nameIsNull Input. Indicates whether a name is specified for the process instance to be created.

newInstance Input/Output. The newly created process instance.

reserved1/reserved2
Input. Pass a 0 (NULL) pointer or an empty string.

returnCode Input/Output. The result of calling this method - see below.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance*/ ProcessInstance
A pointer to the newly created process instance respectively the newly created process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_INVALID_NAME(134)
The specified process instance name does not comply with the syntax rules.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_NOT_UNIQUE(121)
The name of the process instance is not unique.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Delete()

This API call deletes the specified process template(s) from the execution server (action call).

Since process templates are versioned, you can specify whether you want to delete the currently valid process template, the past versions of the process template, or the future versions of the process template. When all options are specified, all versions of the process template are deleted. Deletion always applies to the currently exisiting process templates only.

There are no impacts on your transient representation of the process template; in C and C++, you have to destruct or deallocate the transient object when it is no longer needed.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process modeling authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPTDelete.
           CALL    "FmcjProcessTemplateDelete"
                          USING
                          BY VALUE
                             hdlTemplate
                          RETURNING
                             intReturnValue.
```

Parameters

currentVersion
Input. An indication whether the current version of this process template is to be deleted.

futureVersions
Input. An indication whether future versions of this process template are to be deleted.

hdlTemplate Input. The handle of the process template to be deleted.

pastVersions Input. An indication whether past versions of this process template are to be deleted.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template or its specified versions do no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ExecuteProcessInstance()

This API call creates a process instance from the specified process template and executes the resulting process instance (action call).

Note that the program execution agent(s) must have been started so that the activity implementations are executed.

This API call can be called synchronously and asynchronously. When called synchronously, the process instance should be short running enough to complete within the application wait time.When called asynchronously, a user context can be specified to correlate the response received later. Additionally, a correlation ID can be received which can be used to wait for the specific response. If a buffer to hold the correlation ID is specified, then it must initially point to FMCJ_NO_CORRELID, that is, contain all zeros (0x00).

Depending on the keepName option, a process instance name must be provided. If the process instance name is to be kept *as is*, you cannot provide an empty string.

The following rules apply for specifying a process instance name:

- You can specify a maximum of 63 characters.
- You can use any printable characters depending on your current locale, except the following:
  `* ? " ; : . $`
- You can use blanks with these restrictions: no leading blanks, no trailing blanks, and no consecutive blanks.

If a unique name may be generated by MQ Workflow, the following applies:

- If no or an empty process instance name is provided, an instance is created with a default name *ProcessTemplateName$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the process template name can be shortened.
- If a process instance name is provided, that name is kept as long as it is unique. If the provided process instance name is already used for another instance, an instance is created with the name *name$Oid*, where *Oid* is a printable version of the process instance object identifier. Since the process instance name cannot be longer than 63 characters, the name can be shortened.

The passed name parameter value remains unchanged; FmcjProcessInstance::Name() returns the actual name of the process instance created. The newly created process instance contains all attributes, primary and secondary.

When initial values are to be passed to the process instance to be created and started, an input container can be provided - see also FmcjProcessTemplate::InContainer(). When a process instance that requires input is

started without specifying an input container, the input-container values are not set. When, for example, input-container values are queried from within an activity implementation, FMC_ERROR_MEMBER_NOT_SET is returned.

Pass a NULL (0) pointer or an empty string for the reserved parameters.

On completion, the executed process instance and its output container are returned. The process instance contains values for the primary attributes only. In case of process instance termination, a container is not returned, that is, a 0-pointer respectively an empty container is returned.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signatures

```
APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstance(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjProcessInstanceHandle *   newInstance,
       FmcjReadOnlyContainerHandle * output     )

APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstanceAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjCorrelID *                correlID,
       char const *                  userContext )
```

C++ language signatures

```
APIRET ExecuteProcessInstance(
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const

APIRET ExecuteProcessInstance(
       FmcjReadWriteContainer const &  input,
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const

APIRET ExecuteProcessInstanceAsync(
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false,
       FmcjCorrelID  *                 correlID = 0,
       string const *                  userContext = 0 )

APIRET ExecuteProcessInstanceAsync(
        FmcjReadWriteContainer const & input,
        string const *                 name = 0,
        string const *                 reserved1 = 0,
        string const *                 reserved2 = 0,
        bool                           keepName = false,
        FmcjCorrelID  *                correlID = 0,
        string const *                 userContext = 0 )
```

Java signature

```
public abstract
ProcessInstance executeProcessInstance(  ReadOnlyContainerHolder output,
                                         String                  name,
                                         String                  reserved1,
                                         String                  reserved2,
                                         Boolean                 keepName
) throws FmcException

public abstract
ProcessInstance executeProcessInstance2( ReadWriteContainer      input,
                                         ReadOnlyContainerHolder output,
                                         String                  name,
                                         String                  reserved1,
                                         String                  reserved2,
                                         Boolean                 keepName
) throws FmcException
```

COBOL

```
        FmcjPTExecuteProcInst.
           CALL    "FmcjProcessTemplateExecuteProcessInstance"
                          USING
                          BY VALUE
                             hdlTemplate
                             name
                             reserved1
                             reserved2
                             inputValue
                             keepName
                          BY REFERENCE
                             newInstance
                             outputValue
                          RETURNING
                             intReturnValue.

        FmcjPTExecuteProcInstAsync.

           CALL    "FmcjProcessTemplateExecuteProcessInstanceAsync"
                          USING
                          BY VALUE
                             hdlTemplate
                             name
                             reserved1
                             reserved2
                             inputValue
                             keepName
                          BY REFERENCE
                             correlID
                          BY VALUE
                             userContext
                          RETURNING
                             intReturnValue.
```

XML message

```
<!-- ProcessTemplateExecute ========================== -->
<!ELEMENT ProcessTemplateExecute
          ( ProcTemplName,
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgramName               (#PCDATA) >
<!ELEMENT KeepName                  (#PCDATA) >
                            <!--  Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
<!ELEMENT ProcessTemplateExecuteResponse
          ( ( ProcessInstance,
              ProcInstOutputData? )
          | Exception ) >
<!ELEMENT ProcessInstance
          ( ProcInstID,
            ProcInstName,
            ProcInstParentName?,
            ProcInstTopLevelName,
            ProcInstDescription?,
            ProcInstState,
            LastStateChangeTime,
            LastModificationTime,
            ProcTemplID,
            ProcTemplName,
            Icon,
            Category? ) >
```

XML message continued

```
<!ELEMENT ProcInstID                (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcInstState             (#PCDATA)  >
     <!-- Expected values: { Ready,      Running,
                                Finished,   Terminated,
                                Suspended,  Terminating,
                                Suspending, Deleted    } -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >
<!ELEMENT ProcTemplID               (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT ProcInstOutputData        (%CONTAINER;) >
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
          <!-- Message text is optional, as it will be ignored
               in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
          (Parameter*) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
```

XML message continued

```
<!--==========================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ==========================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

Parameters

correlID Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response.

hdlTemplate Input. The handle of the process template object to be used.

input Input. The input container of the process instance.

keepName Input. True, if only the specified name can be used for the process instance. False, if a unique name can be generated.

name Input. The name of the process instance to be executed.

nameIsNull Input. Indicates whether a name is specified for the process instance to be executed.

newInstance Input/Output. The executed process instance.

output Output. A pointer or an object to contain the output container of the process instance.

returnCode Input/Output. The result of calling this method - see below.

reserved1/reserved2
Input. Pass a 0 (NULL) pointer or an empty string.

userContext Input. A user-defined context which can be used for correlation.

Return type

APIRET The return code of calling this API call - see return codes below.

ProcessInstance*
A pointer to the newly created and executed process instance.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_INVALID_CONTAINER(509)
The specified container is invalid for process instance execution; wrong schema or version.

FMC_ERROR_INVALID_CORRELATION_ID(506)
The specified correlation ID does not point to FMCJ_NO_CORRELID.

FMC_ERROR_INVALID_NAME(134)
The specified process instance name does not comply with the syntax rules.

FMC_ERROR_INVALID_USER_CONTEXT(819)
The specified user context is longer than 254 characters.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_NOT_UNIQUE(121)
The name of the process instance is not unique.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.
FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.
FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.
FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.
FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.
FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.
FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.
FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.
FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
The type of the container is incorrect.
FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
The specified data member is not part of the container.
FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
The type of the data member value passed is incorrect.

XML example - MQ Workflow XML request sent to MQ Workflow:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateExecute>
```

```
    <ProcTemplName>OnlineCreditRequest</ProcTemplName>
    <ProcInstName>Credit Request #658321</ProcInstName>
    <KeepName>true</KeepName>
    <ProcInstInputData>
      </CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage>
```

XML example - MQ Workflow XML response sent from MQ Workflow to the reply queue:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateExecuteResponse>
    <ProcessInstance>
      <ProcInstID>42424242EFEFEFEF</ProcInstID>
      <ProcInstName>Credit Request #658321</ProcInstName>
      <ProcInstTopLevelName>Credit Request #658321</ProcInstTopLevelName>
      <ProcInstDescription>Sample description</ProcInstDescription>
      <ProcInstState>Finished</ProcInstState>
      <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
      <LastModificationTime>1999-05-19 23:40:00</LastModTime>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
    <ProcInstOutputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstOutputData>
  </ProcessTemplateExecuteResponse>
</WfMessageHeader>
```

InitialInContainer()

This API call retrieves the input container associated with the process template from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the system administrator

Action and activity implementation calls

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateInitialInContainer(
       FmcjProcessTemplateHandle      hdlTemplate,
       FmcjReadWriteContainerHandle * input )
```

C++ language signature

```
APIRET InitialInContainer( FmcjReadWriteContainer & input )
```

Java signature

```
public abstract
ReadWriteContainer initialInContainer() throws FmcException
```

COBOL

```
        FmcjPTInitialInCtnr.
           CALL    "FmcjProcessTemplateInitialInContainer"
                          USING
                          BY VALUE
                             hdlTemplate
                          BY REFERENCE
                             inputValue
                          RETURNING
                             intReturnValue.
```

Parameters

hdlTemplate	Input. The handle of the process template object whose input container is to be retrieved.
input	Input/Output. The address of the input container handle respectively the input container of the process template to be set.

Return type

long/ APIRET	The result of calling this API call - see return codes below.
ReadWriteContainer	The input container of the process template.

Return codes/ FmcException

FMC_OK(0)	The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ProgramTemplate()

This API call retrieves the program template identified by the passed name from the MQ Workflow execution server (action call).

A program template comprises data about its associated input and output containers, implementation data for all specified platforms and various other properties. In case *structures from activity* was specified for the program during Buildtime, no input or output container information is available; any container can be passed to the program when executed.

When containers are provided for a program template, then they are initial containers. Such, no default values are set for data members. Also predefined data members are not set.

The result of calling this API call is dependent on the system where the request is executed because there are values returned that can be inherited from the system.

The program template is versioned within the context of the corresponding process template.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjProcessTemplateProgramTemplate(
       FmcjProcessTemplateHandle   hdlTemplate,
       char const *                programName,
       FmcjProgramTemplateHandle * program )
```

C++ language signature

```
APIRET ProgramTemplate( string const &         programName,
                        FmcjProgramTemplate &  program    ) const
```

Java signature

```
public abstract
ProgramTemplate  programTemplate( String programName )
throws FmcException
```

COBOL

```
FmcjPTProgramTempl.
   CALL    "FmcjProcessTemplateProgramTemplate"
                  USING
                  BY VALUE
                     hdlTemplate
                     programName
                  BY REFERENCE
                     programValue
                  RETURNING
                     intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template where a program template is to be retrieved.

program Input/Output. The program template retrieved.

programName Input. The name of the program template to be retrieved.

returnCode Input/Output. The result of calling this method - see below.

Return type

APIRET/long* The result of calling this API call - see return codes below.

ProgramTemplate/ProgramTemplate*
The program template respectively a pointer to the program template retrieved.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_BACK_LEVEL_OBJECT
The request can only be executed on process templates translated after MQ Workflow 3.2.1 has been installed.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid or the program template does not exist within the process template.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Refresh()

This API call refreshes the process template from the MQ Workflow execution server (action call).

All information about the process template - primary and secondary - is retrieved.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

One of:
- Process authorization
- Process administration authorization
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjProcessTemplateRefresh( FmcjProcessTemplateHandle hdlTemplate )
```

C++ language signature

```
APIRET Refresh()
```

Java signature

```
public abstract
void   refresh() throws FmcException
```

COBOL

```
FmcjPTProgramTempl.
   CALL    "FmcjProcessTemplateProgramTemplate"
                  USING
                  BY VALUE
                     hdlTemplate
                     programName
                  BY REFERENCE
                     programValue
                  RETURNING
                     intReturnValue.
```

Parameters

hdlTemplate Input. The handle of the process template object to be refreshed.

Return type

long/ APIRET The result of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The process template does no longer exist or is no longer valid.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjProcessTemplateList.

The following sections describe the actions which can be applied on a process template list. See “ProcessTemplateList” on page 302 for a complete list of API calls.

QueryProcessTemplates()

This API call retrieves the primary information for all process templates characterized by the specified process template list from the MQ Workflow execution server (action call).

From the set of qualifying process templates, only those are retrieved, the user is authorized for. The user is authorized for a process template if the process template:
- Does not belong to any category
- Does belong to a category and the user has global process authorization or global process administration authorization or selected process authorization or selected process administration authorization for that category

The primary information that is retrieved for each process template is:
- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name
- ValidFromTime

In C and C++, any process templates retrieved are appended to the supplied vector of process templates. If you want to read those process templates only which are currently included in the process template list, you have to clear the vector before you call this API call.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplateList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjPTLQueryProcTempls.
           CALL    "FmcjProcessTemplateListQueryProcessTemplates"
                          USING
                          BY VALUE
                             hdlList
                          BY REFERENCE
                             templates
                          RETURNING
                             intReturnValue.
```

Parameters

hdlList Input. The handle of the process template list to be queried.

templates Input/Output. The vector of qualifying process templates.

Return type
long/ APIRET The result of calling this API call - see return codes below.
ProcessTemplate[]
The qualifying process templates.

Return codes/ FmcException
FMC_OK(0) The API call completed successfully.
FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.
FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.
FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The process template list does no longer exist.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process templates to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query worklists (C-language)" on page 586
- For a C++ example see "Query worklists (C++)" on page 588

Program template actions

An FmcjProgramTemplate or a ProgramTemplate object represents the definition of a program within a process template.

A program template is uniquely identified by its name and the process template wherin it is contained. This means that it is versioned via the containing process template.

The following sections describe the actions which can be applied on a program template. See "ProgramTemplate" on page 305 for a complete list of API calls.

Execute()

This API call requests the execution of the specified program template on the program execution server (PES) of the system where the user is logged on.

This API call can be called synchronously and asynchronously. When called synchronously, the program should be short running enough to complete within the application wait time. When called asynchronously, a user context can be specified to correlate the response received later. Additionally, a correlation ID can be received which can be used to wait for the specific response. If a buffer to hold the correlation ID is specified, then it must initially point to FMCJ_NO_CORRELID, that is, contain all zeros (0x00).

Depending on the *input container access* definition of the program template, an input container must be specified for execution. Depending on the *output container access* definition, an output container can be specified to hold the values returned by program execution. If an output container is accessed by the program but none is provided, then the output container defined for the program is used. When *structures from activity* is defined, containers passed can be of any type since the program thus states that it is able to handle any container. When *structures from activity* is not defined, any containers passed must conform to the type defined in the program settings.

Initial containers returned by FmcjProcessTemplate::ProgramTemplate() do not contain any default values. When initial values are to be passed to the program, they can be set in the input or output container before calling this API call.

The output container, if any, is returned on completion. The _RC data member of the output container denotes the program return code.

Specification of a priority influences OS/390 Workload management. The priority must be a value between 0 and 999.

Notes:

1. Passthrough() cannot be called from a program executed on the PES.
2. The output container is an input/output parameter. This means for Java, that it is passed as an input parameter and that it is returned as the return value of the execute method; the input parameter is not changed.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Action and activity implementation calls

Be logged on

Required connection

MQ Workflow program execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProgramTemplate
COBOL	fmcjvars.cpy, fmcjperf.cpy

C-language signatures

```
APIRET FMC_APIENTRY FmcjProgramTemplateExecute(
       FmcjProcessTemplateHandle     hdlTemplate,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output   )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptions(
       FmcjProcessTemplateHandle     hdlTemplate,
       unsigned long                 priority,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output,
       FmcjCorrelID *                correlID,
       char const *                  userContext )

APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptionsAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       unsigned long                 priority,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output,
       FmcjCorrelID *                correlID,
       char const *                  userContext )
```

C++ language signatures

```
APIRET Execute(      FmcjReadWriteContainer const * input    = 0,
                     FmcjReadWriteContainer *       output   = 0,
                     unsigned long                  priority = 0 ) const

APIRET ExecuteAsync( FmcjReadWriteContainer const * input       = 0,
                     FmcjReadWriteContainer const * output      = 0,
                     FmcjCorrelID  *                correlID    = 0,
                     string const *                 userContext = 0,
                     unsigned long                  priority    = 0) const
```

Java signatures

```
public abstract
ReadWriteContainer execute() throws FmcException

public abstract
ReadWriteContainer execute2( ReadWriteContainer input,
                             ReadWriteContainer output,
                             long                priority )
throws FmcException
```

COBOL

```
FmcjPgTExecute.
   CALL    "FmcjProgramTemplateExecute"
                  USING
                  BY VALUE
                     hdlTemplate
                     inputValue
                     outputValue
                  BY REFERENCE
                     returnCode
                  RETURNING
                     intReturnValue.

FmcjPgTExecuteWithOptions.
   CALL    "FmcjProgramTemplateExecuteWithOptions"
                  USING
                  BY VALUE
                     hdlTemplate
                     priority
                     inputValue
                     outputValue
                  BY REFERENCE
                     returnCode
                  RETURNING
                     intReturnValue.

FmcjPgTExecuteAsync.
   CALL    "FmcjProgramTemplateExecuteAsync"
                  USING
                  BY VALUE
                     hdlTemplate
                     inputValue
                     outputValue
                  BY REFERENCE
                     correlID
                  BY VALUE
                     userContext
                  RETURNING
                     intReturnValue.

FmcjPgTExecuteWithOptionsAsync.
   CALL    "FmcjProgramTemplateExecuteWithOptionsAsync"
                  USING
                  BY VALUE
                     hdlTemplate
                     priority
                     inputValue
                     outputValue
                  BY REFERENCE
                     correlID
                  BY VALUE
                     userContext
                  RETURNING
                     intReturnValue.
```

Parameters

correlID Input/Output. If specified, contains the correlation ID by which this request can be correlated to a later response.

hdlTemplate Input. The handle of the program template object to be executed.

input Input. The input container of the program.

output Input/Output. The output container of the program.

priority Input. The priority of the program to be executed.

userContext Input. A user-defined context which can be used for correlation.

Return type

long/APIRET The return code of calling this API call - see return codes below.

ReadWriteContainer
The output container of the program.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_EXIT_ERROR(32204)
A program execution server exit reported an error.

FMC_ERROR_IMPLEMENTATION_SUPPORT_MISMATCH(32015)
The program definition for the operating system platform the PES is running on is not found.

FMC_ERROR_INVALID_CONTAINER(509)
The type or version of the container is incorrect or a container is expected but not passed.

FMC_ERROR_INVALID_CORRELATION_ID(506)
The specified correlation ID does not point to FMCJ_NO_CORRELID.

FMC_ERROR_INVALID_SPECIFICATION(816)
The specified priority must be in the range 0<=priority<=999.

FMC_ERROR_INVALID_USER_CONTEXT(819)
The specified user context is longer than 254 characters.

FMC_ERROR_LOCAL_USER_REQUIRED(32203)
The program must be executed by a local user.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_SUPPORT_MODE_MISMATCH(32014)
The execution mode of the program and the execution mode of the PES do not match.

FMC_ERROR_UNEXPECTED_CONTAINER(510)
A container is passed but not expected by the program.

FMC_ERROR_USER_NOT_AUTHORIZED(32202)
The user is not authorized to execute the program.

FMC_ERROR_USER_SUPPORT_MISMATCH(32013)
The execution user of the program and the execution user of the PES do not match.

FMC_ERROR_COMMUNICATION(13)
The specified program execution server cannot be reached.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

Action and activity implementation calls

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.
FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
The maximum allowed backout count is exceeded.
FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
The value of the MQMD format field is incorrect.
FMC_ERROR_XML_DOCUMENT_INVALID(1100)
The document is not a valid XML document.
FMC_ERROR_XML_INVALID_ELEMENT(1110)
There is an invalid element in the XML message.
FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
The document is not a valid MQ Workflow XML document.
FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
There is an invalid parameter in the XML message.
FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
There is an invalid parameter combination in the XML message.
FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
The type of the container is incorrect.
FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
The specified data member is not part of the container.
FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
The type of the data member value passed is incorrect.

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 or COBOL, common implementations of functions are taken from FmcjService. That is, common functions start with the prefix FmcjService; they are also defined starting with the prefix FmcjExecutionService.

The following sections describe the actions which can be applied on a service. See “Service” on page 310 for a complete list of API calls.

Refresh()

This API call refreshes the logon status from the server (action call).

Usage note
- See “Action API calls” on page 133 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Service

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjSrvRefresh.
           CALL    "FmcjServiceRefresh"
                          USING
                          BY VALUE
                             serviceValue
                          RETURNING
                             intReturnValue.
```

Parameters

service Input. A handle to the service object representing the session with an MQ Workflow server.

Return type

APIRET/long The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

SetPassword()

This API call allows a user's password to be changed (action call).

Note: The password is case-sensitive.

The following rules apply for specifying a password:
- You can specify a maximum of 32 characters.
- You can use any printable characters depending on your current locale.
- Do not use DBCS characters.

Note: If you intend to work in a multi-platform environment or switch between codepages, it is recommended that you use alphabetic characters, digits, and blanks only. This is because it cannot be guaranteed that special characters are available in all codepages.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjSrvSetPassword.
           CALL    "FmcjServiceSetPassword"
                          USING
                          BY VALUE
                             serviceValue
                             newPassword
                          RETURNING
                             intReturnValue.
```

Parameters

newPassword Input. The new password to be used.

service Input. A handle to the service object representing the session with an MQ Workflow server.

Return type

long/ APIRET The return code of calling this API call - see return codes below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_USERID_UNKNOWN(10)
The user does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_PASSWORD(12)
The password does not comply with the MQ Workflow syntax rules.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

UserSettings()

This API call returns all settings of the logged on user (action call).

An empty object respectivly a null pointer is returned if no user has logged on yet via this service object.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Logon required

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjSrvUserSettings.
           CALL    "FmcjServiceUserSettings"
                          USING
                          BY VALUE
                             serviceValue
                          BY REFERENCE
                             user
                          RETURNING
                             intReturnValue.
```

Parameters

returnCode Input/Output. The return code of calling this method - see return codes below.

service Input. A handle to the service object representing the session with an MQ Workflow server.

user Input/Output. The person object to contain respectively the address of the person handle to point to the settings of the logged on user.

Return type

APIRET The return code of calling this API call - see return codes below.

IDispatch*/ Person
A pointer to the person settings or the person settings of the logged on user.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common implementations of functions are taken from FmcjItem. That is, common functions start with the prefix FmcjItem; they are also defined starting with the prefix FmcjWorkitem. .

A work item is uniquely identified by its object identifier.

The following diagrams provide an overview on the possible work item states and the actions which are allowed in those states, provided that the appropriate authority has been granted and that more specific requirements stated in the API calls descriptions have been fulfilled. Note that the actions and possible states are dependent on the process instance state, the work item is a part of.

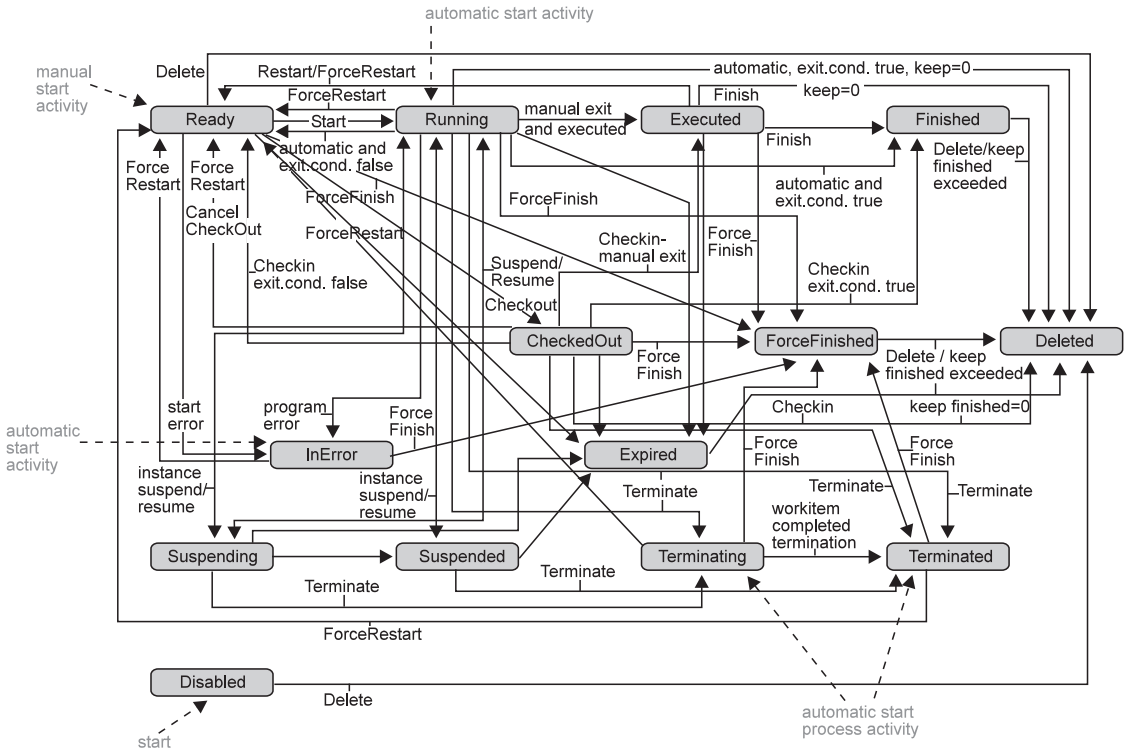


Figure 42. Work item states - process instance state running

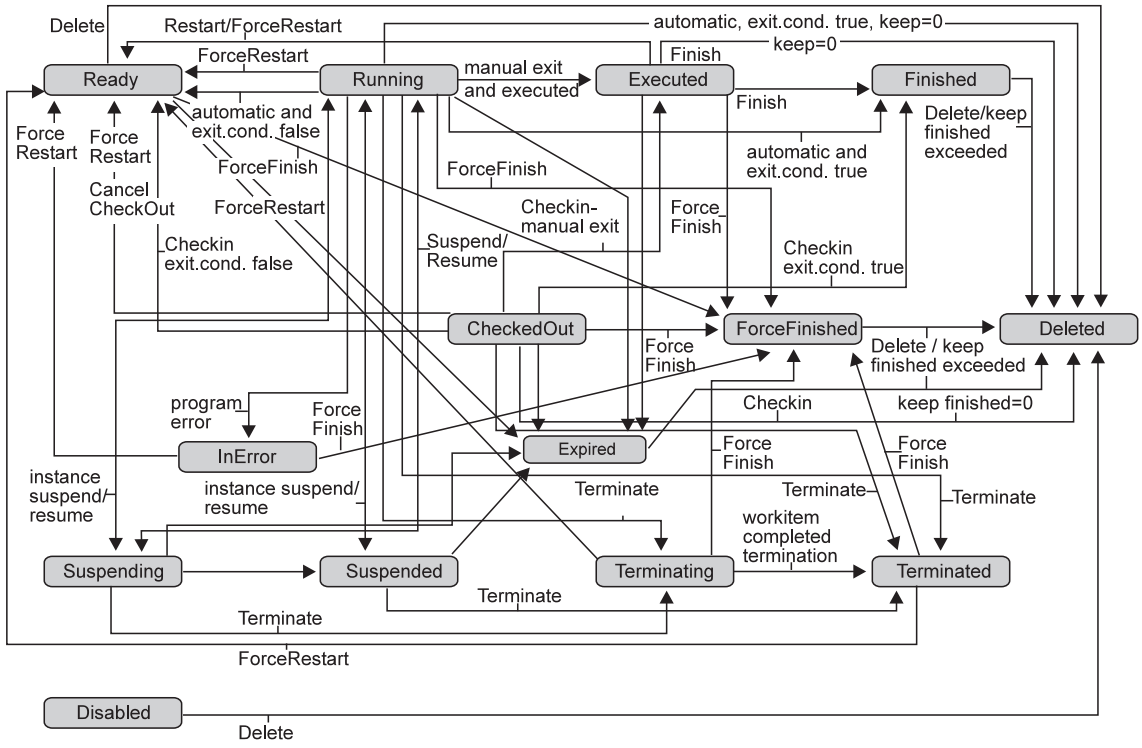


Figure 43. Work item states - process instance state suspending or suspended

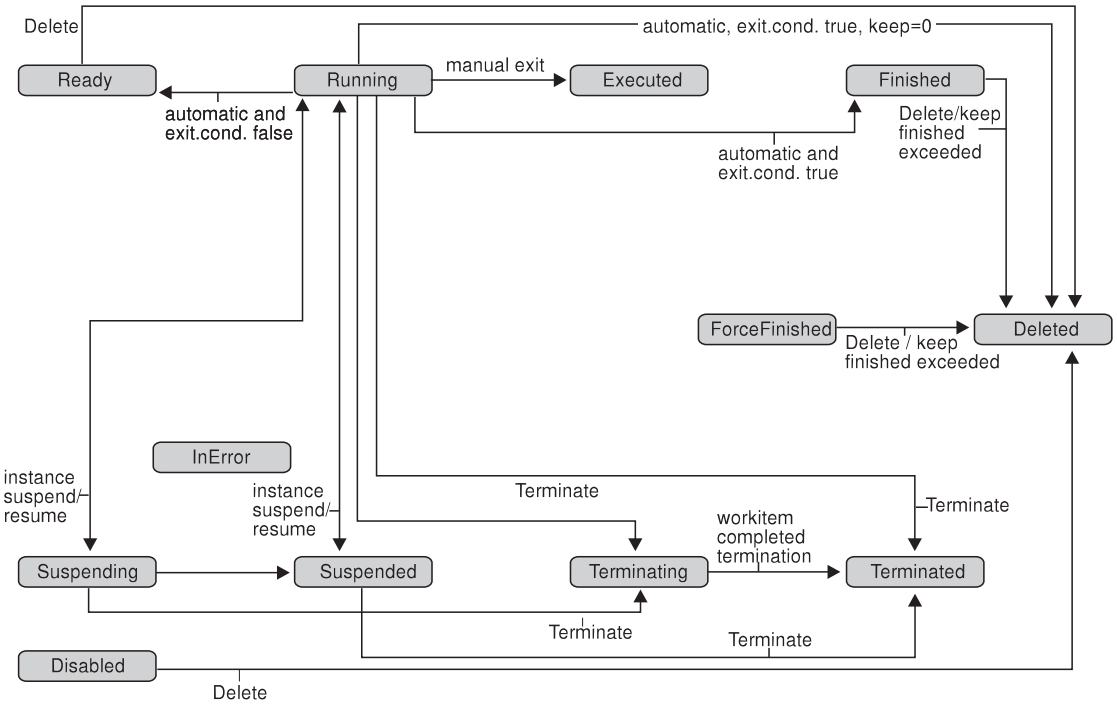


Figure 44. Work item states - process instance state terminating or terminated

The following sections describe the actions which can be applied on a work item. See "Workitem" on page 312 for a complete list of API calls.

ActivityInstance()

This API call retrieves the activity instance the work item is associated to from the MQ Workflow execution server (action call).

All information about the activity instance, primary and secondary, is retrieved.

In C++, when the activity instance object to be initialized is not empty, that object is destructed before the new one is assigned. In C, the application is completely responsible for the ownership of objects, that is, it is not checked whether the activity instance handle already points to some object.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

One of:

- Process authorization
- Process administration authorization
- Be the process creator
- Be the process administrator
- Be the system administrator

Required connection

MQ Workflow execution server

API interface declarations

ActiveX	IBM MQSeries Workflow Control 3.1
C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

ActiveX signature

```
ActivityInstance* ActivityInstance( long * returnCode )
```

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemActivityInstance(
       FmcjWorkitemHandle               hdlWorkitem,
       FmcjActivityInstanceHandle *     instance )
```

C++ language signature

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java signature

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

COBOL

```
        FmcjWIActInst.
           CALL    "FmcjWorkitemActivityInstance"
                          USING
                          BY VALUE
                             hdlWorkitem
                          BY REFERENCE
                             instance
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem	Input. The handle of the work item object to be queried.
instance	Input/Output. The activity instance object to be retrieved (initialized).
returnCode	Input/Output. The return code of calling this method - see return codes below.

Return type

APIRET	The return code of calling this API call - see return codes below.
ActivityInstance*/ ActivityInstance	A pointer to the activity instance or the activity instance the work item is associated to.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item or activity instance does not exist. The activity instance may not exist when the transient work item object has been recreated from its OID and when it is not a work item but a process instance notification.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

CancelCheckOut()

This API call cancels the checkout of the work item (action call).

The work item must have been checked out and is put into the *Ready* state. The associated process instance must be in the *Running*, *Suspending*, *Suspended*, or *Terminating* state.

Note that all sibling work items set into the *Disabled* state by the previous Checkout() request are also reset into the *Ready* state.

Usage note

- See "Action API calls" on page 133 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
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWICancelCheckOut.
           CALL    "FmcjWorkitemCancelCheckOut"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is not in a required state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

CheckIn()

This API call allows for the check in of a work item that was previously checked out for user processing (action call).

The work item must be in the *CheckedOut* state and the associated process instance must be in the *Running* or *Suspending* state.

Checking in a work item tells MQ Workflow that user processing has finished and workflow processing under the control of MQ Workflow can continue. The return code of the user processing and, optionally, the output container values are passed back to MQ Workflow. As usual, these container values and the return code can be used in exit conditions to let navigation continue depending on the success of the

processing and in transition conditions to indicate how to proceed. The return code is automatically set into the _RC data member of the output container if this field has not been set explicitly.

When an output container is specified, then that container must be a valid container for the work item, that is, it must contain the correct schema and version definitions. In other words, it must be the (updated) output container retrieved with the *CheckOut()* request or the output container retrieved for the work item, and so on.

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
FmcjWICheckIn.
   CALL    "FmcjWorkitemCheckIn"
                  USING
                  BY VALUE
                     hdlWorkitem
                     outputValue
                     returnCode
                  RETURNING
                     intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

output Input. A handle or pointer to the output container; can be a NULL pointer.

returnCode Input. The return code of user processing.

Return type

long/ APIRET

The return code of calling this API call- see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_INVALID_CONTAINER(509)
The specified output container is invalid; wrong schema or version.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item is not checked out.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

CheckOut()

This API call checks out a ready work item for user processing (action call).

The work item must be implemented as a program and be in the *Ready* state. The associated process instance must be in the *Running* state.

Checkout then means that processing is not done by MQ Workflow's inherent program-invocation mechanism. MQ Workflow assumes that processing is done by user-specific means and changes the state of the work item to *CheckedOut*.

The caller can request program definitions for specific operating system platforms. The following enumeration types can be used to specify the requested program data.

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.

C-language	Fmc_WS_NotSet
C++	FmcjWorkitem::NotSet
JAVA	ProgramRetrieval.NOT_SET
COBOL	Fmc-DP-NotSet

CommonDataOnly
returns only data common to all platforms, the description, the icon, the unattended indicator, and the input and output containers. Any platform specification is ignored.

C-language	Fmc_WS_CommonDataOnly
C++	FmcjWorkitem::CommonDataOnly
JAVA	ProgramRetrieval.COMMON_DATA_ONLY
COBOL	Fmc-WS-CommonDataOnly

SpecifiedDefinitions
returns the program definition for the specified platform. A platform must be specified.

C-language Fmc_WS_SpecifiedDefinitions

C++ FmcjWorkitem::SpecifiedDefinitions

JAVA ProgramRetrieval.SPECIFIED_DEFINITIONS

COBOL Fmc-WS-SpecifiedDefs

AllDefinitions
returns all available program definitions. Any platform specification is ignored.

C-language Fmc_WS_AllDefinitions

C++ FmcjWorkitem::AllDefinitions

JAVA ProgramRetrieval.ALL_DEFINITIONS

COBOL Fmc-WS-AllDefs

The following enumeration types can be used to specify the platform for which program definitions are to be retrieved.

C-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.

C-language Fmc_DP_NotSet

C++ FmcjImplementationData::NotSpecified

JAVA Basis.NOT_SPECIFIED

COBOL Fmc-DP-NotSet

OS2 indicates that the program definition for the OS/2 platform is requested.

C-language Fmc_DP_OS2

C++ FmcjImplementationData::OS2

JAVA Basis.OS2

COBOL Fmc-DP-OS2

AIX indicates that the program definition for the AIX platform is requested.

C-language Fmc_DP_AIX

C++ FmcjImplementationData::AIX

JAVA Basis.AIX

COBOL Fmc-DP-AIX

HPUX indicates that the program definition for the HP-UX platform is requested.

C-language Fmc_DP_HPUX

C++ FmcjImplementationData::HPUX

JAVA Basis.HPUX

COBOL Fmc-DP-HPUX

Windows95 indicates that the program definition for the Windows 95, 98, or Me platform is requested.

C-language	Fmc_DP_Windows95
C++	FmcjImplementationData::Windows95
JAVA	Basis.WINDOWS_95
COBOL	Fmc-DP-Windows95

WindowsNT indicates that the program definition for the Windows NT or Windows 2000 platform is requested.

C-language	Fmc_DP_WindowsNT
C++	FmcjImplementationData::WindowsNT
JAVA	Basis.WINDOWS_NT
COBOL	Fmc-DP-WindowsNT

OS390 indicates that the program definition for the OS/390(R) platform is requested.

C-language	Fmc_DP_OS390
C++	FmcjImplementationData::OS390
JAVA	Basis.OS390
COBOL	Fmc-DP-OS390

Solaris indicates that the program definition for the Solaris platform is requested.

C-language	Fmc_DP_Solaris
C++	FmcjImplementationData::Solaris
JAVA	Basis.Solaris
COBOL	Fmc-DP-Solaris

Usage note

- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWICheckOut.
           CALL    "FmcjWorkitemCheckOut"
                         USING
                         BY VALUE
                            hdlWorkitem
                            requestedData
                            platform
                         BY REFERENCE
                            programData
                         RETURNING
                            intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

platform Input. The platform for which the program definition is to be returned.

programData Input/Output. The address of a handle to the program definition respectively the program definition object to be set.

requestedData
Input. An indicator which program definitions are to be returned.

returnCode Input/Output. The return code of calling this method - see below.

Return type

APIRET The return code of calling this method - see below.

ProgramData The program definition.

ReadOnlyContainer
The input container of the work item; the container is part of the program definition. Returned for Version 2 compatibility reasons.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_CHECKOUT_NOT_POSSIBLE(503)
The work item cannot be checked out.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_INVALID_SPECIFICATION(816)
Invalid combination of checkout parameters.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_PROGRAM_NOT_DEFINED(1012)
No program defined for the requested platform.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Finish()

This API call ends the execution of a manual-exit work item (action call).

The work item must be in state *Executed*, that is, must have run at least once. The work item is then put into the *Finished* state. Depending on the "delete finished items" option, it is deleted.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemFinish( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Finish()
```

Java signature

```
public abstract
void   finish() throws FmcException
```

COBOL

```
        FmcjWIFinish.
           CALL    "FmcjWorkitemFinish"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceFinish()

This API call ends the execution of a work item which is known to have completed (action call).

A work item implemented by a program must be in the states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in the states *Ready, Executed, InError*, or *Terminated*. The associated process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

Optionally, an output container can be specified to denote the result of processing. If none is specified, the output container available at the execution server is taken. For example, the output container defined with initial values.

The work item is then put into the *ForceFinished* state. The exit condition is considered to be true and navigation proceeds.

Depending on the "delete finished items" option, the work item is deleted.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemForceFinish( FmcjWorkitemHandle hdlWorkitem )

APIRET FMC_APIENTRY
FmcjWorkitemForceFinishWithContainer( FmcjWorkitemHandle   hdlWorkitem,
                                      FmcjContainerHandle  outputContainer )
```

C++ language signature

```
APIRET ForceFinish()

APIRET ForceFinish( FmcjContainer const & outputContainer )
```

Java signature

```
public abstract
void   forceFinish() throws FmcException

public abstract
void   forceFinish2( Container outputContainer ) throws FmcException
```

COBOL

```
FmcjWIForceFinish.
   CALL    "FmcjWorkitemForceFinish"
                  USING
                  BY VALUE
                     hdlWorkitem
                  RETURNING
                     intReturnValue.

FmcjWIForceFinishWithCtnr.
   CALL    "FmcjWorkitemForceFinish"
                  USING
                  BY VALUE
                     hdlWorkitem
                     outputContainer
                  RETURNING
                     intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

outputContainer
Input. The output container to be set as the result of the call; can be a read/write or read-only container.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

ForceRestart()

This API call forces MQ Workflow to enable the restart of a work item (action call).

A work item implemented by a program must be in states *Ready*, *Running*, *Executed*, *CheckedOut*, *InError*, *Terminating*, or *Terminated*. A work item implemented by a process must be in states *Ready*, *Executed, InError*, or *Terminated*. The associated process instance must be in states *Running*, *Suspending*, or *Suspended*.

Optionally, an input container can be specified to denote the input to be used for restarting the work item. If none is specified, the input container available at the execution server is taken.

The work item is then reset into the *Ready* state. Note that automatic activity instances must now be started manually.

Usage note
- See "Action API calls" on page 133 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

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY FmcjWorkitemForceRestart(
                                FmcjWorkitemHandle hdlWorkitem )

APIRET FMC_APIENTRY FmcjWorkitemForceRestartWithContainer(
                                FmcjWorkitemHandle  hdlWorkitem,
                                FmcjContainerHandle inputContainer )
```

C++ language signature

```
APIRET ForceRestart()

APIRET ForceRestart( FmcjContainer const & inputContainer )
```

Java signature

```
public abstract
void   forceRestart() throws FmcException

public abstract
void   forceRestart2( Container inputContainer ) throws FmcException
```

COBOL

```
        FmcjWIForceRestart.
           CALL    "FmcjWorkitemForceRestart"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.

        FmcjWIForceRestartWithCtnr.
           CALL    "FmcjWorkitemForceRestart"
                          USING
                          BY VALUE
                             hdlWorkitem
                             inputContainer
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

inputContainer
Input. The input container to be used when restarting the work item.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

InContainer()

This API call retrieves the input container associated with the work item from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWIInCtnr.
           CALL    "FmcjWorkitemInContainer"
                         USING
                         BY VALUE
                            hdlWorkitem
                         BY REFERENCE
                            inputValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

input Input/Output. The input container.

Return type

long/ APIRET The return code of calling this method - see below.

ReadOnlyContainer
The input container.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

OutContainer()

This API call retrieves the output container associated with the work item from the MQ Workflow execution server (action call).

Usage note

- See "Action API calls" on page 133 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

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWIOutCtnr.
           CALL    "FmcjWorkitemOutContainer"
                         USING
                         BY VALUE
                            hdlWorkitem
                         BY REFERENCE
                            outputValue
                         RETURNING
                            intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

output Input/Output. The output container.

Return type

long/ APIRET The return code of calling this method - see below.

ReadWriteContainer

The output container.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)

A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)

The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.
FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.
FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.
FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Restart()

This API call asks MQ Workflow to enable the restart of a work item (action call).

The work item must be in state *Executed*. It is then reset into the *Ready* state.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemRestart( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Restart()
```

Java signature

```
public abstract
void   restart() throws FmcException
```

COBOL

```
        FmcjWIRestart.
           CALL    "FmcjWorkitemRestart"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Start()

This API call starts a ready work item (action call).

The associated process instance must be in the *Running* state.

If the associated activity instance is implemented by a program, the program is started on the program execution agent associated to the logged-on user.

The work item is put into the *Running* state. If the activity implementation or an associated process activity cannot be started, the work item is put into the *InError* state.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemStart( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Start()
```

Java signature

```
public abstract
void   start() throws FmcException
```

COBOL

```
        FmcjWIStart.
           CALL    "FmcjWorkitemStart"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

StartTool()

This API call starts the specified support tool (action call).

The support tool must be one of the tools associated to the activity instance the work item is derived from. It is then started on the program execution agent associated to the logged-on user.

Note: A support tool can be started only via a program execution agent in the LAN environment; starting via a program execution server (in either environment) is currently not supported. Since there are only unattended processes under MQ Workflow for OS/390, it is not meaningful to start a support tool in this environment. The PES will simply ignore such an attempt.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWIStartTool.
           CALL    "FmcjWorkitemStartTool"
                         USING
                         BY VALUE
                            hdlWorkitem
                            toolName
                         RETURNING
                            intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be dealt with.
toolName Input. The support tool to be started.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_INVALID_TOOL(129)
No tool name is provided or the specified tool is not defined for the work item.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.
FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.
FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.
FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.
FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.
FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.
FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.
FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Terminate()

This API call terminates a work item implemented by a program or process (action call).

If the work item is implemented by a program, it must be in the states *CheckedOut* or *Running* and the process instance must be in the states *Running*, *Suspending*, or *Suspended*. If the work item is implemented by a process, it must be in the states *Running*, *Suspending*, or *Suspended* and the process instance must be in the states *Running*, *Suspending*, *Suspended*, or *Terminating*.

When the work item is implemented by a program and processed under the control of a program execution agent or user-defined program execution server, a message is sent to inform about the termination request. The program execution agent tries to kill fenced activity implementations.

A work item implemented by a process is terminated together with all its non-autonomous subprocesses with respect to control autonomy.

The work item is then put into the *Terminating* or *Terminated* state.

When the *Terminated* state has been reached, the exit condition is considered to be false, the output container and especially the return code (_RC) are not set, and navigation ends. Navigation can be explicitly continued by a user with process administration rights, that is, ForceFinish() or ForceRestart() repair actions can be called.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

Be the work item owner

Required connection

MQ Workflow execution server

API interface declarations

C-language fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

COBOL fmcvars.cpy, fmcperf.cpy

C-language signature

```
APIRET FMC_APIENTRY
FmcjWorkitemTerminate( FmcjWorkitemHandle hdlWorkitem )
```

C++ language signature

```
APIRET Terminate()
```

Java signature

```
public abstract
void   terminate() throws FmcException
```

COBOL

```
        FmcjWITerminate.
           CALL    "FmcjWorkitemTerminate"
                          USING
                          BY VALUE
                             hdlWorkitem
                          RETURNING
                             intReturnValue.
```

Parameters

hdlWorkitem Input. The handle of the work item to be terminated.

Return type

long/ APIRET The return code of calling this method - see below.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The work item does no longer exist.

FMC_ERROR_NOT_AUTHORIZED(119)
Not authorized to use the API call.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_WRONG_KIND(501)
The transient work item object recreated from its OID is not a work item; it is a notification.

FMC_ERROR_WRONG_STATE(120)
The work item or process instance is in the wrong state.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

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 or COBOL, common implementations of functions are taken from FmcjPersistentList. That is, common functions start with the prefix FmcjPersistentList; they are also defined starting with the prefix FmcjWorklist.

The following sections describe the actions which can be applied on a worklist. See "Worklist" on page 315 for a complete list of API calls.

QueryActivityInstanceNotifications()

This API call retrieves the primary information for all activity instance notifications characterized by the specified worklist from the MQ Workflow execution server (action call).

From the set of qualifying activity instance notifications, only those are retrieved, the user is authorized for. The user is authorized for an activity instance notification if
- He is the owner of the activity instance notification
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each activity instance notification is:
- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

In C, C++, and COBOL, any activity instance notifications retrieved are appended to the supplied vector of activity instance notifications. If you want to read those activity instance notifications only which are currently included in the worklist, you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Action and activity implementation calls

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWLQueryActInstNotifs.
           CALL    "FmcjWorklistQueryActivityInstanceNotifications"
                         USING
                         BY VALUE
                            hdlList
                         BY REFERENCE
                            notifications
                         RETURNING
                            intReturnValue.
```

Parameters

hdlList	Input. The handle of the worklist to be queried.
notifications	Input/Output. The vector of qualifying activity instance notifications.

Return type

long/ APIRET	The return code of calling this method - see below.
ActivityInstanceNotification[]	The qualifying activity instance notifications.

Return codes/ FmcException

FMC_OK(0)	The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of activity instance notifications to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query work items from a worklist (C-language)" on page 611
- For a C++ example see "Query work items from a worklist (C++)" on page 612
- For a Java example see "Query work items from a worklist (Java)" on page 614

QueryItems()

This API call retrieves the primary information for all items characterized by the specified worklist from the MQ Workflow execution server (action call).

From the set of qualifying items, only those are retrieved, the user is authorized for. The user is authorized for an item if

- He is the owner of the item
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each item is:
- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

If the item is an actual work item or an activity instance notification, then additional primary information is retrieved:
- ActivityType
- Implementation
- Priority
- SupportTools

In C, C++, or COBOL, any items retrieved are appended to the supplied vector of items. If you want to read those items only which are currently included in the worklist, you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWLQueryItems.
           CALL    "FmcjWorklistQueryItems"
                          USING
                          BY VALUE
                             hdlList
                          BY REFERENCE
                             items
                          RETURNING
                             intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.
items Input/Output. The vector of qualifying items.

Return type

APIRET The return code of calling this method - see below.
Item[] The qualifying items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of items to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples
- For a C-language example see "Query work items from a worklist (C-language)" on page 611
- For a C++ example see "Query work items from a worklist (C++)" on page 612
- For a Java example see "Query work items from a worklist (Java)" on page 614

QueryProcessInstanceNotifications()

This API call retrieves the primary information for all process instance notifications characterized by the specified worklist from the MQ Workflow execution server (action call).

From the set of qualifying process instance notifications, only those are retrieved, the user is authorized for. The user is authorized for a process instance notification if
- He is the owner of the process instance notification
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each process instance notification is:
- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

In C, C++, or COBOL, any process instance notifications retrieved are appended to the supplied vector of process instance notifications. If you want to read those process instance notifications only which are currently included in the worklist,

you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note
- See "Action API calls" on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
       FmcjWLQueryProcInstNotifs.
           CALL    "FmcjWorklistQueryProcessInstanceNotifications"
                           USING
                           BY VALUE
                              hdlList
                           BY REFERENCE
                              notifications
                           RETURNING
                              intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.

notifications Input/Output. The vector of qualifying process instance notifications.

Return type

long/ APIRET The return code of calling this method - see below.

ProcessInstanceNotification[]
The qualifying process instance notifications.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of process instance notifications to be returned exceeds the maximum size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query work items from a worklist (C-language)" on page 611
- For a C++ example see "Query work items from a worklist (C++)" on page 612
- For a Java example see "Query work items from a worklist (Java)" on page 614

QueryWorkitems()

This API call retrieves the primary information for all work items characterized by the specified worklist from the MQ Workflow execution server (action call).

From the set of qualifying work items, only those are retrieved, the user is authorized for. The user is authorized for a work item if
- He is the owner of the work item
- He has workitem authority
- He is the system administrator

The primary information that is retrieved for each work item is:
- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

In C, C++, or COBOL, any work items retrieved are appended to the supplied vector of work items. If you want to read those work items only which are currently included in the worklist, you have to clear the vector before you call this API call. This means that you should set the handle to 0 in the C-language or COBOL respectively erase all elements of the vector in the C++ API.

Usage note
- See “Action API calls” on page 133 for general information.

Authorization

None

Required connection

MQ Workflow execution server

API interface declarations

C-language	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList
COBOL	fmcvars.cpy, fmcperf.cpy

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
```

COBOL

```
        FmcjWLQueryWorkitems.
           CALL    "FmcjWorklistQueryWorkitems"
                         USING
                         BY VALUE
                            hdlList
                         BY REFERENCE
                            workitems
                         RETURNING
                            intReturnValue.
```

Parameters

hdlList Input. The handle of the worklist to be queried.
workitems Input/Output. The vector of qualifying work items.

Return type

long/ APIRET The return code of calling this method - see below.
WorkItem[] The qualifying work items.

Return codes/ FmcException

FMC_OK(0) The API call completed successfully.

FMC_ERROR(1)
A parameter references an undefined location. For example, the address of a handle is 0.

FMC_ERROR_EMPTY(122)
The object does not yet represent a persistent object, that is, it has not yet been read from the database or it has not been created from its OID.

FMC_ERROR_INVALID_HANDLE(130)
The handle provided is incorrect; it is 0 or it is not pointing to an object of the requested type.

FMC_ERROR_DOES_NOT_EXIST(118)
The worklist does no longer exist.

FMC_ERROR_NOT_LOGGED_ON(106)
Not logged on.

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
The number of work items to be returned exceeds the maximum

size allowed for query results - see the MAXIMUM_QUERY_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_COMMUNICATION(13)
The specified server cannot be reached; the server to which the connection should be established is not defined in your profile.

FMC_ERROR_INTERNAL(100)
An MQ Workflow internal error has occurred. Contact your IBM representative.

FMC_ERROR_INVALID_CHAR(16)
A string contains an incorrect character; probably a code page problem.

FMC_ERROR_INVALID_CODE_PAGE(15)
Code page conversion from the client code page into the server's code page is not supported.

FMC_ERROR_MESSAGE_FORMAT(103)
An internal message format error. Contact your IBM representative.

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
The message to be returned exceeds the maximum size allowed - see the MAXIMUM_MESSAGE_SIZE definition in your system, system group, or domain.

FMC_ERROR_TIMEOUT(14)
Timeout has occurred.

Examples

- For a C-language example see "Query work items from a worklist (C-language)" on page 611
- For a C++ example see "Query work items from a worklist (C++)" on page 612
- For a Java example see "Query work items from a worklist (Java)" on page 614

Action and activity implementation calls

Chapter 6. Examples

The following samples are provided in InstHLQ.SFMCSRC:

FMCHSCFA C Full API sample (native OS/390)

FMCHSCCA C Container API sample (CICS and IMS)

FMCHSCBF COBOL Full API sample (native OS/390)

FMCHSCBC COBOL Container API sample (IMS)

FMCHSCBN COBOL Container API sample (CICS)

In addition, the following sections illustrate examples for:

- Creating persistent lists, such as process instances
- Querying persistent lists, such as process instances
- Querying a set of objects, such as process instances and work items
- Programming an activity implementation (executable)

How to create persistent lists

The following examples show how to create a persistent list, that is, a persistent view 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 (C-language)

```
#include <stdio.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                            rc           = FMC_OK;
  FmcjExecutionServiceHandle        service      = 0;
  FmcjProcessInstanceListHandle     instanceList = 0;
  unsigned long                     threshold    = 10;
  int                               enumValue    = 0;
  char name[50]  = "MyTenInstances";
  char desc[50]   = "This list contains no more than 10 instances";

   FmcjGlobalConnect();
   /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%\n",rc);
    return -1;
  }
```

Figure 45. Sample C program to create a process instance list (Part 1 of 4)

Examples

```
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%\n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }
```

Figure 45. Sample C program to create a process instance list (Part 2 of 4)

```
 /* create a process instance list */
 rc = FmcjExecutionServiceCreateProcessInstanceList(
                          service,
                          name,
                          Fmc_LT_Private,
                          "USERID",
                          desc,
                          FmcjNoFilter,
                          FmcjNoSortCriteria,
                          &threshold,
                          &instanceList );
```

Figure 45. Sample C program to create a process instance list (Part 3 of 4)

```
  if ( rc != FMC_OK)
    printf( "CreateProcessInstanceList returns: %u%\n",rc );
  else
    printf( "CreateProcessInstanceList okay\n" );

  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 45. Sample C program to create a process instance list (Part 4 of 4)

Create a process instance list (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();

  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }

  // create a process instance list

  FmcjProcessInstanceList instanceList;
  string           name  ("MyTenInstances");
  string           desc  ("List contains no more than 10 instances");
  string           onwer ("USERID");
  unsigned long    threshold= 10;
```

Figure 46. Sample C++ program to create a process instance list (Part 1 of 2)

```
  rc = service.CreateProcessInstanceList(
                    name,
                    FmcjPersistentList::Private,
                    &owner,
                    &desc,
                    FmcjNoFilter,
                    FmcjNoSortCriteria,
                    &threshold,
                    instanceList );
  if ( rc != FMC_OK)
     cout << "CreateProcessInstanceList returns: " << rc << endl;
  else
     cout << "CreateProcessInstanceList okay" << endl;

  service.Logoff();

  FmcjGlobal::Disconnect();
  return 0;
}
```

Figure 46. Sample C++ program to create a process instance list (Part 2 of 2)

Examples

Create a process instance list (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class CreateProcInstList
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java CreateProcessInstanceList
                                  <agent> <LOC|RMI|OSA|IOR|COS>
                                  [userid] [password]");
      System.exit(0);
    }
```

Figure 47. Sample Java program to create a process instance list (Part 1 of 8)

```
try
    {
      // An agent bean representing a MQSeries Workflow domain
      String userid = "USERID";
      String passwd = "password";
      Agent agent = new Agent();
      // Parse the command line and set the locator to be used to
      // communicate with the agent.
      if      (args[1].equalsIgnoreCase("LOC"))
      {
        agent.setLocator(Agent.LOC_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("RMI"))
      {
        agent.setLocator(Agent.RMI_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("OSA"))
      {
        agent.setLocator(Agent.OSA_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("IOR"))
      {
        agent.setLocator(Agent.IOR_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("COS"))
      {
        agent.setLocator(Agent.COS_LOCATOR);
      }
      else
      {
        System.out.println("Invalid locator policy: " + args[1]);
        System.exit(0);
      }
```

Figure 47. Sample Java program to create a process instance list (Part 2 of 8)

```
      if (args.length >=3 ) userid = args[2].toUpperCase();
      if (args.length >=4 ) passwd = args[3];

      // Set the name of the Agent to be contacted. Setting the name
      // automatically instructs the agent bean to contact the Agent using
      // the current locator policy. For this reason the 'setLocator' must be
      // called before 'setName' is invoked. If the agent bean cannot contact
      // the Agent, it will raise a java.beans.PropertyVetoException instead
      // of returning from the 'setName' call.
      agent.setName(args[0]);

      // Locate the default execution service in the system group named
      // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
      // always returns successful (to prevent intrusion attempts which guess
      // at service names until they find a valid one). Of course, only using
      // a valid systemgroup and/or system name will return an ExecutionService
      // which can be used to log on.
      ExecutionService service = agent.locate("", "");
```

Figure 47. Sample Java program to create a process instance list (Part 3 of 8)

Examples

```
// Log on to the execution service. If the UserID and/or the password is
// invalid, a FmcException will be thrown.
service.logon(userid, passwd);
System.out.println("Logon successful");

String ListName      ="MyTenInstances";
String ListDesc      = "List contains no more than 10 instances";
String ListFilter    = "";
String ListSort      = "";
int    ListThreshold = 10;
```

Figure 47. Sample Java program to create a process instance list (Part 4 of 8)

```
try
{
  service.createProcessInstanceList( ListName, TypeOfList.PRIVATE,
                                     userid  , ListDesc, ListFilter,
                                     ListSort, ListThreshold);
  System.out.println("Private ProcessInstanceList created successfully");
}
catch(FmcException e)
{
   if ( e.rc == FmcException.FMC_ERROR_NOT_UNIQUE )
   {
     System.out.println("ProcessInstanceList: '" + ListName +
                        "' already exists");
   }
}
```

Figure 47. Sample Java program to create a process instance list (Part 5 of 8)

```
finally
{
   // Logoff from the execution service. This (like any other remote call)
   // may raise an FmcException indicating a communication failure.
   service.logoff();

   System.out.println("Logoff successful");
 }
}
```

Figure 47. Sample Java program to create a process instance list (Part 6 of 8)

```
catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }
```

Figure 47. Sample Java program to create a process instance list (Part 7 of 8)

```
catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 47. Sample Java program to create a process instance list (Part 8 of 8)

Examples

Create a process instance list (COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "VECTOR".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 listName      PIC X(50) VALUE z"MyTenInstances".
   01 desc          PIC X(50)
         VALUE z"This list contains no more than 10 instances".

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      PERFORM FmcjGlobalConnect.

* logon
      PERFORM FmcjESAllocate.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      MOVE Fmc-SM-Default TO sessionMode.
      MOVE Fmc-SA-Reset TO absenceIndicator.
      PERFORM FmcjESLogon.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         PERFORM FmcjESDeallocate
         MOVE -1 TO retCode
         GOBACK
      END-IF

* create a process instance list
      CALL "SETADDR" USING listName name.
      CALL "SETADDR" USING localUserID ownerValue.
      CALL "SETADDR" USING desc description.
      CALL "SETADDR" USING FmcjNoFilter filter.
      CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
      MOVE FmcjNoThreshold TO threshold.
      MOVE Fmc-LT-Private TO typeValue.
      PERFORM FmcjESCreateProcInstList.
```

Figure 48. Sample COBOL program to create a process instance list (via PERFORM) (Part 1 of 2)

```
MOVE intReturnValue TO retCode
IF retCode NOT = FMC-OK
   DISPLAY "CreateProcessInstanceList returns - rc: "
   DISPLAY retCode
ELSE
   DISPLAY "CreateProcessInstanceList okay"
END-IF

PERFORM FmcjESLogoff.
PERFORM FmcjESDeallocate.
PERFORM FmcjGlobalDisconnect.
MOVE FMC-OK TO retCode.
GOBACK.

COPY fmcperf.
```

Figure 48. Sample COBOL program to create a process instance list (via PERFORM) (Part 2 of 2)

Examples

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "VECTOR".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 listName      PIC X(50) VALUE z"MyTenInstances".
   01 desc          PIC X(50)
         VALUE z"This list contains no more than 10 instances".

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      CALL "FmcjGlobalConnect".
* logon
      CALL "FmcjExecutionServiceAllocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      CALL "FmcjExecutionServiceLogon"
         USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
         RETURNING intReturnValue.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
         MOVE -1 TO retCode
         GOBACK
      END-IF
```

Figure 49. Sample COBOL program to create a process instance list (via CALL) (Part 1 of 2)

```
* create a process instance list
      CALL "SETADDR" USING listName name.
      CALL "SETADDR" USING localUserID ownerValue.
      CALL "SETADDR" USING desc description.
      CALL "SETADDR" USING FmcjNoFilter filter.
      CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
      CALL "FmcjExecutionServiceCreateProcessInstanceList"
         USING BY VALUE serviceValue
                        name
                        Fmc-LT-Private
                        ownerValue
                        description
                        filter
                        sortCriteria
                        FmcjNoThreshold
                     BY REFERENCE
                        newList
                     RETURNING
                        intReturnValue.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "CreateProcessInstanceList returns - rc: "
         DISPLAY retCode
      ELSE
         DISPLAY "CreateProcessInstanceList okay"
      END-IF

      CALL "FmcjExecutionServiceLogoff"
         USING BY VALUE serviceValue
         RETURNING intReturnValue.
      CALL "FmcjExecutionServiceDeallocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      CALL "FmcjGlobalDisconnect".
      MOVE FMC-OK TO retCode.
      GOBACK.
```

Figure 49. Sample COBOL program to create a process instance list (via CALL) (Part 2 of 2)

How to query persistent lists

The following examples show how to retrieve persistent lists from the 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.

Examples

Query worklists (C-language)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                        rc           = FMC_OK;
  FmcjExecutionServiceHandle    service      = 0;
  FmcjWorklistHandle            worklist     = 0;
  FmcjWorklistVectorHandle      lists        = 0;
  unsigned long                 numWList     = 0;
  unsigned long                 i            = 0;
  unsigned long                 enumValue    = 0;
  char                          tInfo[4096+1]= "";
```

Figure 50. Sample C program to query worklists (Part 1 of 10)

```
  FmcjGlobalConnect();

  /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%\n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%\n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }
```

Figure 50. Sample C program to query worklists (Part 2 of 10)

```
  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &lists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%\n",rc );
  else
     printf( "QueryWorklists() returns okay\n" );
```

Figure 50. Sample C program to query worklists (Part 3 of 10)

```
if (rc == FMC_OK)
{
  numWList= FmcjWorklistVectorSize(lists);
  printf ("Number of worklists returned : %u\n", numWList);
  for( i=1; i<= numWList; i++ )
  {
    worklist= FmcjWorklistVectorNextElement(lists);
    FmcjWorklistName( worklist, tInfo, 4097 );
    printf("- Name                          : %s\n",tInfo);
```

Figure 50. Sample C program to query worklists (Part 4 of 10)

```
    enumValue= FmcjWorklistType(worklist);
    if ( enumValue == Fmc_LT_Private )
      printf("- Type                          : %s\n","private");
    if ( enumValue == Fmc_LT_Public )
      printf("- Type                          : %s\n","public");

    FmcjWorklistOwnerOfList( worklist, tInfo, 4097 );
    printf("- OwnerOfList                   : %s\n",tInfo);
    printf("- OwnerOfList is null ?         : %u\n",
             FmcjWorklistOwnerOfListIsNull(worklist) );
```

Figure 50. Sample C program to query worklists (Part 5 of 10)

```
    FmcjWorklistDescription( worklist, tInfo, 4097 );
    printf("- Description                   : %s\n",tInfo);
    printf("- Description is null ?         : %u\n",
             FmcjWorklistDescriptionIsNull(worklist) );
```

Figure 50. Sample C program to query worklists (Part 6 of 10)

```
    FmcjWorklistFilter( worklist, tInfo, 4097 );
    printf("- Filter                        : %s\n",tInfo);
    printf("- Filter is null ?              : %u\n",
             FmcjWorklistFilterIsNull(worklist) );
```

Figure 50. Sample C program to query worklists (Part 7 of 10)

```
    FmcjWorklistSortCriteria( worklist, tInfo, 4097 );
    printf("- SortCriteria                  : %s\n",tInfo);
    printf("- SortCriteria is null ?        : %u\n",
             FmcjWorklistSortCriteriaIsNull(worklist) );
```

Figure 50. Sample C program to query worklists (Part 8 of 10)

Examples

```
      printf("- Threshold                     : %u\n",
               FmcjWorklistThreshold(worklist) );
      printf("- Threshold is null ?           : %u\n",
               FmcjWorklistThresholdIsNull(worklist) );
                                /* deallocate just read object  */
      FmcjWorklistDeallocate(&worklist);
    }
    FmcjWorklistVectorDeallocate(&lists);
  }
```

Figure 50. Sample C program to query worklists (Part 9 of 10)

```
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 50. Sample C program to query worklists (Part 10 of 10)

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;
  }
```

Figure 51. Sample C++ program to query worklists (Part 1 of 10)

```
  // query worklists

  vector<FmcjWorklist> lists;
  FmcjWorklist         worklist;
  rc = service.QueryWorklists( lists );
  if ( rc != FMC_OK)
     cout << "QueryWorklists() returns: " << rc << endl;
  else
     cout << "QueryWorklists returns okay" << endl;
```

Figure 51. Sample C++ program to query worklists (Part 2 of 10)

```
if (rc == FMC_OK)
{
  unsigned int numWList= lists.size();
  cout << "Number of worklists returned : " << numWList << endl;
```

Figure 51. Sample C++ program to query worklists (Part 3 of 10)

```
  for( unsigned long i=0; i< numWList; i++ )
  {
    worklist= lists[i];
    cout << "Name              : " << worklist.Name()              << endl;
```

Figure 51. Sample C++ program to query worklists (Part 4 of 10)

```
    cout << "Type              : " <<
            ((worklist.Type() == FmcjPersistentList::Private) ? "private" :
             (worklist.Type() == FmcjPersistentList::Public)  ? "public"  :
             "not set" )           << endl;
```

Figure 51. Sample C++ program to query worklists (Part 5 of 10)

```
   cout << "Owner             : " << worklist.OwnerOfList()       << endl;
    cout << "Owner null ?      : " << worklist.OwnerOfListIsNull() << endl;
```

Figure 51. Sample C++ program to query worklists (Part 6 of 10)

```
    cout << "Description       : " << worklist.Description()       << endl;
    cout << "Description null ?: " << worklist.DescriptionIsNull() << endl;
```

Figure 51. Sample C++ program to query worklists (Part 7 of 10)

```
   cout << "Filter            : " << worklist.Filter()            << endl;
    cout << "Filter null ?     : " << worklist.FilterIsNull()      << endl;
    cout << "SortCriteria      : " << worklist.SortCriteria()      << endl;
    cout << "SortCriteria null?: " << worklist.SortCriteriaIsNull()<< endl;
```

Figure 51. Sample C++ program to query worklists (Part 8 of 10)

```
    cout << "Threshold         : " << worklist.Threshold()         << endl;
    cout << "Threshold null ?  : " << worklist.ThresholdIsNull()   << endl;
    cout << endl;     }     cout << endl;   }
```

Figure 51. Sample C++ program to query worklists (Part 9 of 10)

```
  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

Figure 51. Sample C++ program to query worklists (Part 10 of 10)

Query worklists (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;

public class QueryWorkLists
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password
    //
    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java QueryWorkLists [userid] [password]");
      System.exit(0);
    }
```

Figure 52. Sample Java program to query worklists (Part 1 of 10)

```
try
{
  // An agent bean representing a MQSeries Workflow domain
  String userid = "USERID";
  String passwd = "password";
  Agent agent = new Agent();

  // Parse the command line and set the locator to be used to
  // communicate with the agent.
  if      (args[1].equalsIgnoreCase("LOC"))
  {
    agent.setLocator(Agent.LOC_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("RMI"))
  {
    agent.setLocator(Agent.RMI_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("OSA"))
  {
    agent.setLocator(Agent.OSA_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("IOR"))
  {
    agent.setLocator(Agent.IOR_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("COS"))
  {
    agent.setLocator(Agent.COS_LOCATOR);
  }
  else
  {
    System.out.println("Invalid locator policy: " + args[1]);
    System.exit(0);
  }
```

Figure 52. Sample Java program to query worklists (Part 2 of 10)

```
  if (args.length >=3 ) userid = args[2].toUpperCase();
  if (args.length >=4 ) passwd = args[3];

  // Set the name of the Agent to be contacted. Setting the name
  // automatically instructs the agent bean to contact the Agent using
  // the current locator policy. For this reason the 'setLocator' must be
  // called before 'setName' is invoked. If the agent bean cannot contact
  // the Agent, it will raise a java.beans.PropertyVetoException instead
  // of returning from the 'setName' call.
  agent.setName(args[0]);
```

Figure 52. Sample Java program to query worklists (Part 3 of 10)

Examples

```
// Locate the default execution service in the system group named
     // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
     // always returns successful (to prevent intrusion attempts which guess
     // at service names until they find a valid one). Of course, only using
     // a valid systemgroup and/or system name will return an ExecutionService
     // which can be used to log on.
     ExecutionService service = agent.locate("", "");

     // Log on to the execution service. If the UserID and/or the password is
     // invalid, a FmcException will be thrown.

     // do a forced logon
     service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
                    AbsenceIndicator.LEAVE );
     System.out.println("Logon successful");
```

Figure 52. Sample Java program to query worklists (Part 4 of 10)

```
     // Query the set of worklists the logged on user can access.
     WorkList[] worklists = service.queryWorkLists();

     if (worklists.length == 0)
     {
       System.out.println("  No worklist found");
     }
     else
     {
       System.out.println("  Number of worklists returned: " + worklists.length
);
```

Figure 52. Sample Java program to query worklists (Part 5 of 10)

```
       // Iterate over the worklists, printing out their names.
       for (int ndx = 0; ndx < worklists.length; ndx++)
       {
         System.out.println("     Name               :" + worklists[ndx].name());

         if (worklists[ndx].type() == TypeOfList.PUBLIC )
         {
           System.out.println("     Type               :Public " );
         }
         else if (worklists[ndx].type() == TypeOfList.PRIVATE )
         {
           System.out.println("     Type               :Private" );
         }
         else
         {
           System.out.println("     Type               :NotSet " );
         }
```

Figure 52. Sample Java program to query worklists (Part 6 of 10)

```
          System.out.println("    Owner             :" + worklists[ndx].ownerOfList());
          System.out.println("    Description       :"+ worklists[ndx].description());
          System.out.println("    Filter            :"+ worklists[ndx].filter());
          System.out.println("    SortCriteria      :"+ worklists[ndx].sortCriteria());
          System.out.println("    Threshold         :"+ worklists[ndx].threshold());
          System.out.println("    ");
        }

      }/* End if*/
```

Figure 52. Sample Java program to query worklists (Part 7 of 10)

```
      // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();

      System.out.println("Logoff successful");
    }
```

Figure 52. Sample Java program to query worklists (Part 8 of 10)

```
catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }
```

Figure 52. Sample Java program to query worklists (Part 9 of 10)

```
 catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 52. Sample Java program to query worklists (Part 10 of 10)

Examples

Query worklists (COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "QUERYWL".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 numWList      PIC 9(9) BINARY VALUE 0.
   01 tInfo         PIC X(4097).
   01 i             PIC 9(9) BINARY VALUE 0.

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      PERFORM FmcjGlobalConnect.

* logon
      PERFORM FmcjESAllocate.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      MOVE Fmc-SM-Default TO sessionMode.
      MOVE Fmc-SA-Reset TO absenceIndicator.
      PERFORM FmcjESLogon.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         PERFORM FmcjESDeallocate
         MOVE -1 TO retCode
         GOBACK
      END-IF

* query worklists
      PERFORM FmcjESQueryWorklists.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryWorklists returns - rc: " retCode
      ELSE
         DISPLAY "QueryWorklists returns okay"
      END-IF
```

Figure 53. Sample COBOL program to query worklists (via PERFORM) (Part 1 of 3)

```
IF retCode = FMC-OK
   SET hdlVector TO lists
   PERFORM FmcjWLVectorSize
   MOVE ulongReturnValue TO numWList
   DISPLAY "Number of worklists returned : " numWList
   PERFORM VARYING i FROM 1 BY 1 UNTIL i >= numWList
      PERFORM FmcjWLVectorNextElement
      SET hdlList TO FmcjWLHandleReturnValue
      MOVE 4097 TO bufferLength
      CALL "SETADDR" USING tInfo listNameBuffer
      PERFORM FmcjWLName
      DISPLAY "- Name               : " tInfo
      PERFORM FmcjWLType
      IF intReturnValue = Fmc-LT-Private
         DISPLAY "- Type               : private"
      END-IF
      IF intReturnValue = Fmc-LT-Public
         DISPLAY "- Type               : public"
      END-IF
      CALL "SETADDR" USING tInfo userIdBuffer
      PERFORM FmcjWLOwnerOfList
      DISPLAY "- OwnerOfList        : " tInfo
      PERFORM FmcjWLOwnerOfListIsNull
      IF boolReturnValue = 0
         DISPLAY "- OwnerOfList is null ? : false"
      ELSE
         DISPLAY "- OwnerOfList is null ? : true"
      END-IF
      CALL "SETADDR" USING tInfo descriptionBuffer
      PERFORM FmcjWLDescription
      DISPLAY "- Description        : " tInfo
      PERFORM FmcjWLDescriptionIsNull
      IF boolReturnValue = 0
         DISPLAY "- Description is null ? : false"
      ELSE
         DISPLAY "- Description is null ? : true"
      END-IF
      CALL "SETADDR" USING tInfo filterBuffer
      PERFORM FmcjWLFilter
      DISPLAY "- Filter             : " tInfo
      PERFORM FmcjWLFilterIsNull
      IF boolReturnValue = 0
         DISPLAY "- Filter is null ?      : false"
      ELSE
         DISPLAY "- Filter is null ?      : true"
      END-IF
      CALL "SETADDR" USING tInfo sortCriteriaBuffer
      PERFORM FmcjWLSortCriteria
      DISPLAY "- SortCriteria       : " tInfo
      PERFORM FmcjWLSortCriteriaIsNull
      IF boolReturnValue = 0
         DISPLAY "- SortCriteria is null ?: false"
      ELSE
         DISPLAY "- SortCriteria is null ?: true"
      END-IF
      PERFORM FmcjWLThreshold
      DISPLAY "- Threshold          : " ulongReturnValue
      PERFORM FmcjWLThresholdIsNull
```

Figure 53. Sample COBOL program to query worklists (via PERFORM) (Part 2 of 3)

Examples

```
      IF boolReturnValue = 0
         DISPLAY "- Threshold is null ?   : false"
      ELSE
         DISPLAY "- Threshold is null ?   : true"
      END-IF
      PERFORM FmcjWLDeallocate
   END-PERFORM
   PERFORM FmcjWLVectorDeallocate
END-IF
PERFORM FmcjESLogoff.
PERFORM FmcjESDeallocate.
PERFORM FmcjGlobalDisconnect.
MOVE FMC-OK TO retCode.
GOBACK.

COPY fmcperf.
```

Figure 53. Sample COBOL program to query worklists (via PERFORM) (Part 3 of 3)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "QUERYWL".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 numWList      PIC 9(9) BINARY VALUE 0.
   01 tInfo         PIC X(4097).
   01 i             PIC 9(9) BINARY VALUE 0.
   01 bufferPtr     USAGE IS POINTER VALUE NULL.

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      CALL "FmcjGlobalConnect".
* logon
      CALL "FmcjExecutionServiceAllocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      CALL "FmcjExecutionServiceLogon"
         USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
         RETURNING intReturnValue.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
         MOVE -1 TO retCode
         GOBACK
      END-IF
```

Figure 54. Sample COBOL program to query worklists (via CALL) (Part 1 of 4)

Examples

```
* query worklists
      CALL "FmcjExecutionServiceQueryWorklists"
         USING BY VALUE serviceValue
               BY REFERENCE lists
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryWorklists returns - rc: " retCode
      ELSE
         DISPLAY "QueryWorklists returns okay"
      END-IF

      IF retCode = FMC-OK
         SET hdlVector TO lists
         CALL "FmcjWorklistVectorSize"
            USING BY VALUE hdlVector
            RETURNING ulongReturnValue
         MOVE ulongReturnValue TO numWList
         DISPLAY "Number of worklists returned : " numWList
         PERFORM VARYING i FROM 1 BY 1 UNTIL i >= numWList
            CALL "FmcjWorklistVectorNextElement"
               USING BY VALUE hdlVector
               RETURNING FmcjWLHandleReturnValue
            SET hdlList TO FmcjWLHandleReturnValue
            MOVE 4097 TO bufferLength
            CALL "SETADDR" USING tInfo bufferPtr
            CALL "FmcjPersistentListName"
               USING BY VALUE hdlList
                              bufferPtr
                              bufferLength
               RETURNING pointerReturnValue
            DISPLAY "- Name               : " tInfo
            CALL "FmcjPersistentListType"
               USING BY VALUE hdlList
               RETURNING intReturnValue
            IF intReturnValue = Fmc-LT-Private
               DISPLAY "- Type               : private"
            END-IF
            IF intReturnValue = Fmc-LT-Public
               DISPLAY "- Type               : public"
            END-IF
            CALL "FmcjPersistentListOwnerOfList"
               USING BY VALUE hdlList
                              bufferPtr
                              bufferLength
               RETURNING pointerReturnValue
            DISPLAY "- OwnerOfList        : " tInfo
            CALL "FmcjPersistentListOwnerOfListIsNull"
               USING BY VALUE hdlList
               RETURNING boolReturnValue
            IF boolReturnValue = 0
               DISPLAY "- OwnerOfList is null ? : false"
            ELSE
               DISPLAY "- OwnerOfList is null ? : true"
            END-IF
            CALL "FmcjPersistentListDescription"
               USING BY VALUE hdlList
                              bufferPtr
                              bufferLength
               RETURNING pointerReturnValue
```

Figure 54. Sample COBOL program to query worklists (via CALL) (Part 2 of 4)

```
      DISPLAY "- Description        : " tInfo
      CALL "FmcjPersistentListDescriptionIsNull"
         USING BY VALUE hdlList
         RETURNING boolReturnValue
      IF boolReturnValue = 0
         DISPLAY "- Description is null ? : false"
      ELSE
         DISPLAY "- Description is null ? : true"
      END-IF
      CALL "FmcjPersistentListFilter"
         USING BY VALUE hdlList
                        bufferPtr
                        bufferLength
         RETURNING pointerReturnValue
      DISPLAY "- Filter             : " tInfo
      CALL "FmcjPersistentListFilterIsNull"
         USING BY VALUE hdlList
         RETURNING boolReturnValue
      IF boolReturnValue = 0
         DISPLAY "- Filter is null ?      : false"
      ELSE
         DISPLAY "- Filter is null ?      : true"
      END-IF
      CALL "FmcjPersistentListSortCriteria"
         USING BY VALUE hdlList
                        bufferPtr
                        bufferLength
         RETURNING pointerReturnValue
      DISPLAY "- SortCriteria       : " tInfo
      CALL "FmcjPersistentListSortCriteriaIsNull"
         USING BY VALUE hdlList
         RETURNING boolReturnValue
      IF boolReturnValue = 0
         DISPLAY "- SortCriteria is null ?: false"
      ELSE
         DISPLAY "- SortCriteria is null ?: true"
      END-IF
      CALL "FmcjPersistentListThreshold"
         USING BY VALUE hdlList
         RETURNING ulongReturnValue
      DISPLAY "- Threshold          : " ulongReturnValue
      CALL "FmcjPersistentListThresholdIsNull"
         USING BY VALUE hdlList
         RETURNING boolReturnValue
      IF boolReturnValue = 0
         DISPLAY "- Threshold is null ?   : false"
      ELSE
         DISPLAY "- Threshold is null ?   : true"
      END-IF
      CALL "FmcjWorklistDeallocate"
         USING BY REFERENCE hdlList
         RETURNING intReturnValue
   END-PERFORM
CALL "FmcjWorklistVectorDeallocate"
   USING BY REFERENCE hdlVector
   RETURNING intReturnValue
END-IF
```

Figure 54. Sample COBOL program to query worklists (via CALL) (Part 3 of 4)

Examples

```
CALL "FmcjExecutionServiceLogoff"
   USING BY VALUE serviceValue
   RETURNING intReturnValue.
CALL "FmcjExecutionServiceDeallocate"
   USING BY REFERENCE serviceValue
   RETURNING intReturnValue.
CALL "FmcjGlobalDisconnect".
MOVE FMC-OK TO retCode.
GOBACK.
```

Figure 54. Sample COBOL program to query worklists (via CALL) (Part 4 of 4)

How to query a set of objects

The following examples show how to query objects for which you are authorized. They use a query for process instances in order to demonstrate an ad-hoc query. They use work items in order to demonstrate how to query the contents of a predefined list, a worklist.

Note: ActiveX supports querying objects only from a predefined list.

Query process instances (C-language)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                           rc           = FMC_OK;
  FmcjExecutionServiceHandle       service      = 0;
  FmcjProcessInstanceHandle        instance     = 0;
  FmcjProcessInstanceVectorHandle iList         = 0;
  unsigned long                    numIList     = 0;
  unsigned long                    i            = 0;
  char                             tInfo[4096+1]= "";

  FmcjGlobalConnect();

  /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%\n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%\n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }
  /* query process instances  */
  rc= FmcjExecutionServiceQueryProcessInstances(
                         service,
                         FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
                         &iList               );
  if ( rc != FMC_OK)
     printf( "QueryProcessInstances() returns: %u%\n",rc );
  else
     printf( "QueryProcessInstances() returns okay\n" );
```

Figure 55. Sample C program to query process instances (Part 1 of 2)

Examples

```
  if (rc == FMC_OK)
  {
    numIList= FmcjProcessInstanceVectorSize(iList);
    printf ("Number of instances returned : %u\n", numIList);

    for( i=1; i<= numIList; i++ )
    {
      instance= FmcjProcessInstanceVectorNextElement(iList);
      FmcjProcessInstanceName( instance, tInfo, 4097 );
      printf("- Name                        : %s\n",tInfo);
      FmcjProcessInstanceDeallocate(&instance);
    }

    FmcjProcessInstanceVectorDeallocate(&iList);
  }

  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 55. Sample C program to query process instances (Part 2 of 2)

Query process instances (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();

  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }
```

Figure 56. Sample C++ program to query process instances (Part 1 of 3)

```
  // query process instances

  vector<FmcjProcessInstance> instances;

  rc = service.QueryProcessInstances(
                   FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
                   instances );
  if ( rc != FMC_OK)
     cout << "QueryProcessInstances returns: " << rc << endl;
  else
     cout << "QueryProcessInstances okay" << endl;
```

Figure 56. Sample C++ program to query process instances (Part 2 of 3)

```
  if ( rc == FMC_OK )
  {
    cout << "Number of instances returned: " << instances.size() << endl;

    for ( int i=0; i < instances.size(); i++ )
      cout << "- Name: " << instances[i].Name() << endl;
  }

  service.Logoff();

  FmcjGlobal::Disconnect();
  return 0;
}
```

Figure 56. Sample C++ program to query process instances (Part 3 of 3)

Query process instances (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryProcInst
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryProcessInstances [userid] [password]");
      System.exit(0);
    }
```

Figure 57. Sample Java program to query process instances (Part 1 of 9)

Examples

```
try
    {
      // An agent bean representing a MQSeries Workflow domain
      String userid = "USERID";
      String passwd = "password";
      Agent agent = new Agent();
      // Parse the command line and set the locator to be used to
      // communicate with the agent.
      if      (args[1].equalsIgnoreCase("LOC"))
      {
        agent.setLocator(Agent.LOC_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("RMI"))
      {
        agent.setLocator(Agent.RMI_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("OSA"))
      {
        agent.setLocator(Agent.OSA_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("IOR"))
      {
        agent.setLocator(Agent.IOR_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("COS"))
      {
        agent.setLocator(Agent.COS_LOCATOR);
      }
      else
      {
        System.out.println("Invalid locator policy: " + args[1]);
        System.exit(0);
      }
```

Figure 57. Sample Java program to query process instances (Part 2 of 9)

```
    if (args.length >=3 ) userid = args[2].toUpperCase();
      if (args.length >=4 ) passwd = args[3];

      // Set the name of the Agent to be contacted. Setting the name
      // automatically instructs the agent bean to contact the Agent using
      // the current locator policy. For this reason the 'setLocator' must be
      // called before 'setName' is invoked. If the agent bean cannot contact
      // the Agent, it will raise a java.beans.PropertyVetoException instead
      // of returning from the 'setName' call.
      agent.setName(args[0]);

      // Locate the default execution service in the system group named
      // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
      // always returns successful (to prevent intrusion attempts which guess
      // at service names until they find a valid one). Of course, only using
      // a valid systemgroup and/or system name will return an ExecutionService
      // which can be used to log on.
      ExecutionService service = agent.locate("", "");
```

Figure 57. Sample Java program to query process instances (Part 3 of 9)

```
// Log on to the execution service. If the UserID and/or the password is
// invalid, a FmcException will be thrown.
// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
               AbsenceIndicator.LEAVE );
System.out.println("Logon successful");
```

Figure 57. Sample Java program to query process instances (Part 4 of 9)

```
// Query a set of processinstances (30 at maximum), sort them by name
ProcessInstance[] procInstances =
  service.queryProcessInstances("","NAME DESC", 30);

if (procInstances.length == 0)
{
  System.out.println("  No process instances found");
}
else
{
  System.out.println("Number of instances returned: " + procInstances.length);
```

Figure 57. Sample Java program to query process instances (Part 5 of 9)

```
  // Iterate over the process instances, printing out their names.
  for (int ndx = 0; ndx < procInstances.length; ndx++)
  {
    System.out.println("   - Name: " + procInstances[ndx].name());
  }
}
```

Figure 57. Sample Java program to query process instances (Part 6 of 9)

```
// Logoff from the execution service. This (like any other remote call)
  // may raise an FmcException indicating a communication failure.
  service.logoff();

  System.out.println("Logoff successful");
}
```

Figure 57. Sample Java program to query process instances (Part 7 of 9)

Examples

```
catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }
```

Figure 57. Sample Java program to query process instances (Part 8 of 9)

```
    catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 57. Sample Java program to query process instances (Part 9 of 9)

Query process instances (COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "QUERYPI".

DATA DIVISION.

  WORKING-STORAGE SECTION.

  COPY fmcvars.
  COPY fmcconst.
  COPY fmcrcs.

  01 localUserID   PIC X(30) VALUE z"USERID".
  01 localPassword PIC X(30) VALUE z"PASSWORD".
  01 numIList      PIC 9(9) BINARY VALUE 0.
  01 tInfo         PIC X(4097).
  01 i             PIC 9(9) BINARY VALUE 0.

  LINKAGE SECTION.

  01 retCode            PIC S9(9) BINARY.

PROCEDURE DIVISION USING retCode.

     PERFORM FmcjGlobalConnect.

* logon
     PERFORM FmcjESAllocate.
     MOVE intReturnValue TO retCode
     IF retCode NOT = FMC-OK
        DISPLAY "Service object could not be allocated"
        DISPLAY "rc: " retCode
        MOVE -1 TO retCode
        GOBACK
     END-IF

     CALL "SETADDR" USING localUserId userId.
     CALL "SETADDR" USING localPassword passwordValue.
     MOVE Fmc-SM-Default TO sessionMode.
     MOVE Fmc-SA-Reset TO absenceIndicator.
     PERFORM FmcjESLogon.

     MOVE intReturnValue TO retCode
     IF retCode NOT = FMC-OK
        DISPLAY "Logon failed - rc: " retCode
        PERFORM FmcjESDeallocate
        MOVE -1 TO retCode
        GOBACK
     END-IF
```

Figure 58. Sample COBOL program to query process instances (via PERFORM) (Part 1 of 2)

Examples

```
* query process instances
      CALL "SETADDR" USING FmcjNoFilter filter.
      CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
      MOVE FmcjNoThreshold TO threshold.
      PERFORM FmcjESQueryProcInsts.

      SET hdlVector TO instances.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryProcessInstances returns: " retCode
      ELSE
         DISPLAY "QueryProcessInstances returns okay"
      END-IF

      IF retCode = FMC-OK
         PERFORM FmcjPIVSize
         MOVE ulongReturnValue TO numIList
         DISPLAY "Number of instances returned: " numIList
         MOVE 4097 TO bufferLength
         CALL "SETADDR" USING tInfo instanceNameBuffer
         PERFORM VARYING i FROM 1 BY 1 UNTIL i > numIList
            PERFORM FmcjPIVNextElement
            SET hdlInstance TO FmcjPIHandleReturnValue
            PERFORM FmcjPIName
            DISPLAY "- name: " tInfo
            PERFORM FmcjPIDeallocate
         END-PERFORM
         PERFORM FmcjPIVDeallocate
      END-IF

      PERFORM FmcjESLogoff.
      PERFORM FmcjESDeallocate.
      PERFORM FmcjGlobalDisconnect.
      MOVE FMC-OK TO retCode.
      GOBACK.

      COPY fmcperf.
```

Figure 58. Sample COBOL program to query process instances (via PERFORM) (Part 2 of 2)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "QUERYPI".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 numIList      PIC 9(9) BINARY VALUE 0.
   01 tInfo         PIC X(4097).
   01 i             PIC 9(9) BINARY VALUE 0.

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

     CALL "FmcjGlobalConnect".
* logon
      CALL "FmcjExecutionServiceAllocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      CALL "FmcjExecutionServiceLogon"
         USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
         RETURNING intReturnValue.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
         MOVE -1 TO retCode
         GOBACK
      END-IF
```

Figure 59. Sample COBOL program to query process instances (via CALL) (Part 1 of 2)

Examples

```
* query process instances
      CALL "SETADDR" USING FmcjNoFilter filter.
      CALL "SETADDR" USING FmcjNoSortCriteria sortCriteria.
      CALL "FmcjExecutionServiceQueryProcessInstances"
         USING BY VALUE     serviceValue
                            filter
                            sortCriteria
                            FmcjNoThreshold
               BY REFERENCE instances
               RETURNING intReturnValue.

      SET hdlVector TO instances.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryProcessInstances returns: " retCode
      ELSE
         DISPLAY "QueryProcessInstances returns okay"
      END-IF

      IF retCode = FMC-OK
         CALL "FmcjProcessInstanceVectorSize"
            USING BY VALUE hdlVector
            RETURNING ulongReturnValue
         MOVE ulongReturnValue TO numIList
         DISPLAY "Number of instances returned: " numIList
         MOVE 4097 TO bufferLength
         CALL "SETADDR" USING tInfo instanceNameBuffer
         PERFORM VARYING i FROM 1 BY 1 UNTIL i > numIList
            CALL "FmcjProcessInstanceVectorNextElement"
               USING BY VALUE hdlVector
               RETURNING FmcjPIHandleReturnValue
            SET hdlInstance TO FmcjPIHandleReturnValue
            CALL "FmcjProcessInstanceName"
               USING BY VALUE hdlInstance
                              instanceNameBuffer
                              FMC-PROC-INST-NAME-LENGTH
               RETURNING pointerReturnValue
            DISPLAY "- name: " tInfo
            CALL "FmcjProcessInstanceDeallocate"
               USING BY REFERENCE hdlInstance
               RETURNING intReturnValue
         END-PERFORM
         CALL "FmcjProcessInstanceVectorDeallocate"
            USING BY REFERENCE hdlVector
            RETURNING intReturnValue
      END-IF

      CALL "FmcjExecutionServiceLogoff"
         USING BY VALUE serviceValue
         RETURNING intReturnValue.
      CALL "FmcjExecutionServiceDeallocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      CALL "FmcjGlobalDisconnect".
      MOVE FMC-OK TO retCode.
      GOBACK.
```

Figure 59. Sample COBOL program to query process instances (via CALL) (Part 2 of 2)

Query work items from a worklist (C-language)

```
#include <stdio.h>
#include <string.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */

int main (int argc, char ** argv)
{
  APIRET                        rc            = FMC_OK;
  FmcjExecutionServiceHandle    service       = 0;
  FmcjWorklistVectorHandle      wLists        = 0;
  FmcjWorklistHandle            worklist      = 0;
  FmcjWorkitemVectorHandle      wVector       = 0;
  FmcjWorkitemHandle            workitem      = 0;
  unsigned long                 numWList      = 0;
  char                          tInfo[4096+1] = "";

  FmcjGlobalConnect();

  /*  Logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated: %u%\n",rc);
    return -1;
  }

  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet );
  if ( rc != FMC_OK )
  {
    printf("Logon failed - rc    : %u%\n",rc);
    rc= FmcjExecutionServiceDeallocate( &service );
    return -1;
  }

  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &wLists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%\n",rc );
  else
     printf( "QueryWorklists() returns okay\n" );
```

Figure 60. Sample C program to query work items from a worklist (Part 1 of 2)

Examples

```
  if (rc == FMC_OK)
  {
    numWList= FmcjWorklistVectorSize(wLists);
    printf ("Number of worklists returned : %u\n", numWList);
    if ( numWList == 0 )
    {
      printf("No worklist found \n");
      FmcjWorklistVectorDeallocate(&wLists);
      rc= FmcjExecutionServiceDeallocate( &service );
      return -1;
    }

    worklist= FmcjWorklistVectorFirstElement(wLists);
    FmcjWorklistName( worklist, tInfo, 4097 );
    printf("Name                          : %s\n",tInfo);

    /* query workitems  */
    rc= FmcjWorklistQueryWorkitems( worklist, &wVector );
    printf("\nQuery workitems of list returns rc: %u\n",rc);

    if (rc == FMC_OK)
    {
      while ( 0 != (workitem= FmcjWorkitemVectorNextElement(wVector)) )
      {
        FmcjWorkitemName( workitem, tInfo, 4097 );
        printf("- Name                        : %s\n",tInfo);

        FmcjWorkitemDeallocate(&workitem);
      }
    }

    FmcjWorklistDeallocate(&worklist);
    FmcjWorklistVectorDeallocate(&wLists);
  }

  /*  Logoff */
  rc= FmcjExecutionServiceLogoff(service);
  rc= FmcjExecutionServiceDeallocate( &service );

  FmcjGlobalDisconnect();
  return 0;
}
```

Figure 60. Sample C program to query work items from a worklist (Part 2 of 2)

Query work items from a worklist (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();
```

Figure 61. Sample C++ program to query work items from a worklist (Part 1 of 5)

```
// logon
FmcjExecutionService service;
APIRET rc = service.Logon("USERID", "password");
if ( rc != FMC_OK )
{
  cout << "Logon failed, - rc: " << rc << endl;
  return -1;
}
```

Figure 61. Sample C++ program to query work items from a worklist (Part 2 of 5)

```
// query worklists

vector<FmcjWorklist> lists;
FmcjWorklist         worklist;

rc = service.QueryWorklists( lists );
if ( rc != FMC_OK)
   cout << "QueryWorklists() returns: " << rc << endl;
else
   cout << "QueryWorklists returns okay" << endl;

if (rc == FMC_OK)
{
  unsigned int numWList= lists.size();
  cout << "Number of worklists returned : " << numWList << endl;
  if ( numWList == 0 )
  {
    cout << "No worklist found" << endl;
    return -1;
  }
```

Figure 61. Sample C++ program to query work items from a worklist (Part 3 of 5)

```
  worklist= lists[0];
  cout << "Name              : " << worklist.Name()              << endl;

  vector<FmcjWorkitem> wVector;
  FmcjWorkitem         workitem;

  rc= worklist.QueryWorkitems( wVector );
  cout << "Query workitems of list returns: " << rc << endl;
  cout << "Number of workitems              " << wVector.size() << endl;
```

Figure 61. Sample C++ program to query work items from a worklist (Part 4 of 5)

Examples

```
    if (rc == FMC_OK)
    {
      for ( int i= 0; i < wVector.size(); i++ )
      {
        workitem= wVector[i];
        cout << "Name              : " << workitem.Name() << endl;
      }
    }
  }

  rc = service.Logoff();

  FmcjGlobal::Disconnect();
  return 0;
}
```

Figure 61. Sample C++ program to query work items from a worklist (Part 5 of 5)

Query work items from a worklist (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;

public class QueryWorkItems
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryWorkitems [userid] [password]");
      System.exit(0);
    }
```

Figure 62. Sample Java program to query work items from a worklist (Part 1 of 8)

```
try
{
  // An agent bean representing a MQSeries Workflow domain
  String userid = "USERID";
  String passwd = "password";
  Agent agent = new Agent();

  // Parse the command line and set the locator to be used to
  // communicate with the agent.
  if      (args[1].equalsIgnoreCase("LOC"))
  {
    agent.setLocator(Agent.LOC_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("RMI"))
  {
    agent.setLocator(Agent.RMI_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("OSA"))
  {
    agent.setLocator(Agent.OSA_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("IOR"))
  {
    agent.setLocator(Agent.IOR_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("COS"))
  {
    agent.setLocator(Agent.COS_LOCATOR);
  }
  else
  {
    System.out.println("Invalid locator policy: " + args[1]);
    System.exit(0);
  }
```

Figure 62. Sample Java program to query work items from a worklist (Part 2 of 8)

Examples

```
if (args.length >=3 ) userid = args[2].toUpperCase();
if (args.length >=4 ) passwd = args[3];

// Set the name of the Agent to be contacted. Setting the name
// automatically instructs the agent bean to contact the Agent using
// the current locator policy. For this reason the 'setLocator' must be
// called before 'setName' is invoked. If the agent bean cannot contact
// the Agent, it will raise a java.beans.PropertyVetoException instead
// of returning from the 'setName' call.
agent.setName(args[0]);

// Locate the default execution service in the system group named
// 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
// always returns successful (to prevent intrusion attempts which guess
// at service names until they find a valid one). Of course, only using
// a valid systemgroup and/or system name will return an ExecutionService
// which can be used to log on.
ExecutionService service = agent.locate("", "");

// Log on to the execution service. If the UserID and/or the password is
// invalid, a FmcException will be thrown.
// do a forced logon
service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
               AbsenceIndicator.LEAVE );
System.out.println("Logon successful");
```

Figure 62. Sample Java program to query work items from a worklist (Part 3 of 8)

```
// Query the set of worklists the logged on user can access.
WorkList[] worklists = service.queryWorkLists();

if (worklists.length == 0)
{
  System.out.println("  No worklist found");
}
else
{
  System.out.println("  Number of worklists returned: " + worklists.length);

  WorkList worklist = worklists[0];
  System.out.println("  Name: "+worklist.name());
```

Figure 62. Sample Java program to query work items from a worklist (Part 4 of 8)

```
  // Query the set of workitems in the first worklist.
  WorkItem[] workitems = worklist.queryWorkItems();
  System.out.println("  Number of workitems: " + workitems.length);

  // Iterate over the workitems, printing out their names.
  for (int ndx = 0; ndx < workitems.length; ndx++)
  {
    System.out.println("    " + workitems[ndx].name());
  }
}/* End if*/
```

Figure 62. Sample Java program to query work items from a worklist (Part 5 of 8)

```
      // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();

      System.out.println("Logoff successful");
    }
```

Figure 62. Sample Java program to query work items from a worklist (Part 6 of 8)

```
 catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }
```

Figure 62. Sample Java program to query work items from a worklist (Part 7 of 8)

```
   catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }

    System.exit(0);
  }
}
```

Figure 62. Sample Java program to query work items from a worklist (Part 8 of 8)

Examples

Query work items from a worklist (COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "QUERYWI".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 numWList      PIC 9(9) BINARY VALUE 0.
   01 tInfo         PIC X(4097).

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      PERFORM FmcjGlobalConnect.

* logon
      PERFORM FmcjESAllocate.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      MOVE Fmc-SM-Default TO sessionMode.
      MOVE Fmc-SA-Reset TO absenceIndicator.
      PERFORM FmcjESLogon.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         PERFORM FmcjESDeallocate
         MOVE -1 TO retCode
         GOBACK
      END-IF

* query worklists
      PERFORM FmcjESQueryWorklists.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryWorklists returns - rc: " retCode
      ELSE
         DISPLAY "QueryWorklists returns okay"
      END-IF
```

Figure 63. Sample COBOL program to query work items from a worklist (via PERFORM) (Part 1 of 2)

```
      IF retCode = FMC-OK
         SET hdlVector TO lists
         PERFORM FmcjWLVectorSize
         MOVE ulongReturnValue TO numWList
         DISPLAY "Number of worklists returned : " numWList
         IF numWList = 0
            DISPLAY "No worklist found"
            PERFORM FmcjWLDeallocate
            PERFORM FmcjESDeallocate
            MOVE -1 TO retCode
            GOBACK
         END-IF

         PERFORM FmcjWLVectorFirstElement
         SET hdlList TO FmcjWLHandleReturnValue
         MOVE 4097 TO bufferLength
         CALL "SETADDR" USING tInfo listNameBuffer
         PERFORM FmcjWLName
         DISPLAY "Name               : " tInfo

* query workitems
         PERFORM FmcjWLQueryWorkitems
         MOVE intReturnValue TO retCode
         DISPLAY "Query workitems of list returns rc:" retCode
         SET hdlVector TO workitems
         CALL "SETADDR" USING tInfo itemNameBuffer
         IF retCode = FMC-OK
            PERFORM FmcjWIVNextElement
            SET hdlItem TO FmcjWIHandleReturnValue
            PERFORM UNTIL pointerReturnValue = NULL
               PERFORM FmcjWIName
               DISPLAY "- Name             : " tInfo
               PERFORM FmcjWIDeallocate
               PERFORM FmcjWIVNextElement
            END-PERFORM
         END-IF
         PERFORM FmcjWLDeallocate
         PERFORM FmcjWLVectorDeallocate
      END-IF

      PERFORM FmcjESLogoff.
      PERFORM FmcjESDeallocate.
      PERFORM FmcjGlobalDisconnect.
      MOVE FMC-OK TO retCode.
      GOBACK.

      COPY fmcperf.
```

Figure 63. Sample COBOL program to query work items from a worklist (via PERFORM) (Part 2 of 2)

Examples

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "QUERYWI".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcconst.
   COPY fmcrcs.

   01 localUserID   PIC X(30) VALUE z"USERID".
   01 localPassword PIC X(30) VALUE z"PASSWORD".
   01 numWList      PIC 9(9) BINARY VALUE 0.
   01 tInfo         PIC X(4097).

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      CALL "FmcjGlobalConnect".
* logon
      CALL "FmcjExecutionServiceAllocate"
         USING BY REFERENCE serviceValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Service object could not be allocated"
         DISPLAY "rc: " retCode
         MOVE -1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING localUserId userId.
      CALL "SETADDR" USING localPassword passwordValue.
      CALL "FmcjExecutionServiceLogon"
         USING BY VALUE serviceValue
                        userID
                        passwordValue
                        Fmc-SM-Default
                        Fmc-SA-Reset
         RETURNING intReturnValue.

      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "Logon failed - rc: " retCode
         CALL "FmcjExecutionServiceDeallocate"
            USING BY REFERENCE serviceValue
            RETURNING intReturnValue
         MOVE -1 TO retCode
         GOBACK
      END-IF
```

Figure 64. Sample COBOL program to query work items from a worklist (via CALL) (Part 1 of 3)

```
* query worklists
      CALL "FmcjExecutionServiceQueryWorklists"
         USING BY VALUE serviceValue
               BY REFERENCE lists
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode
      IF retCode NOT = FMC-OK
         DISPLAY "QueryWorklists returns - rc: " retCode
      ELSE
         DISPLAY "QueryWorklists returns okay"
      END-IF

      IF retCode = FMC-OK
         SET hdlVector TO lists
         CALL "FmcjWorklistVectorSize"
            USING BY VALUE hdlVector
            RETURNING ulongReturnValue
         MOVE ulongReturnValue TO numWList
         DISPLAY "Number of worklists returned : " numWList
         IF numWList = 0
            DISPLAY "No worklist found"
            CALL "FmcjWorklistDeallocate"
               USING BY REFERENCE hdlList
               RETURNING intReturnValue
            CALL "FmcjExecutionServiceDeallocate"
               USING BY REFERENCE serviceValue
               RETURNING intReturnValue
            MOVE -1 TO retCode
            GOBACK
         END-IF

         CALL "FmcjWorklistVectorFirstElement"
            USING BY VALUE hdlVector
            RETURNING FmcjWLHandleReturnValue
         SET hdlList TO FmcjWLHandleReturnValue
         MOVE 4097 TO bufferLength
         CALL "SETADDR" USING tInfo listNameBuffer
         CALL "FmcjPersistentListName"
            USING BY VALUE hdlList
                           listNameBuffer
                           bufferLength
            RETURNING pointerReturnValue
         DISPLAY "Name               : " tInfo

* query workitems
         CALL "FmcjWorklistQueryWorkitems"
            USING BY VALUE hdlList
            BY REFERENCE workitems
            RETURNING intReturnValue
         MOVE intReturnValue TO retCode
         DISPLAY "Query workitems of list returns rc:" retCode
         SET hdlVector TO workitems
         CALL "SETADDR" USING tInfo itemNameBuffer
         IF retCode = FMC-OK
            CALL "FmcjWorkitemVectorNextElement"
               USING BY VALUE hdlVector
               RETURNING FmcjWIHandleReturnValue
            SET hdlItem TO FmcjWIHandleReturnValue
```

Figure 64. Sample COBOL program to query work items from a worklist (via CALL) (Part 2 of 3)

Examples

```
      PERFORM UNTIL pointerReturnValue = NULL
         CALL "FmcjItemName"
            USING BY VALUE hdlItem
                           itemNameBuffer
                           bufferLength
            RETURNING pointerReturnValue
         DISPLAY "- Name             : " tInfo
         CALL "FmcjWorkitemDeallocate"
            USING BY REFERENCE hdlWorkitem
            RETURNING intReturnValue
         CALL "FmcjWorkitemVectorNextElement"
            USING BY VALUE hdlVector
            RETURNING FmcjWIHandleReturnValue
      END-PERFORM
   END-IF
   CALL "FmcjWorklistDeallocate"
      USING BY REFERENCE hdlList
      RETURNING intReturnValue
   CALL "FmcjWorklistVectorDeallocate"
      USING BY REFERENCE hdlVector
      RETURNING intReturnValue
END-IF

    CALL "FmcjExecutionServiceLogoff"
   USING BY VALUE serviceValue
   RETURNING intReturnValue.
CALL "FmcjExecutionServiceDeallocate"
   USING BY REFERENCE serviceValue
   RETURNING intReturnValue.
CALL "FmcjGlobalDisconnect".
MOVE FMC-OK TO retCode.
GOBACK.
```

Figure 64. Sample COBOL program to query work items from a worklist (via CALL) (Part 3 of 3)

An activity implementation

The following examples show the concept of how to query and set containers from within an activity implementation. Refer to the examples provided with the product for more details.

Programming an executable (C-language)

```
#include <stdio.h>
#include <fmcjccon.h>                  /* MQ Workflow Container API */
int main()
{
  FILE                           *  file1             = 0;
  APIRET                            rc                = FMC_OK;
  FmcjReadOnlyContainerHandle       input             = 0;
  FmcjReadWriteContainerHandle      output            = 0;
  char                              stringBuffer[4097]="";

/*- keep results in a file ------------------------------------------------*/
  file1 = fopen ("sample.out", "a");
  if ( file1 == 0 )
    return -1;
  fprintf(file1,"\n----- C-API Activity Implementation called -----\n");
  fflush(file1);
```

Figure 65. Sample activity implementation (C-language) (Part 1 of 4)

```
  FmcjGlobalConnect();

/*-- retrieve the input container from the PEA who started the program --*/
  rc = FmcjContainerInContainer( &input );
  fprintf(file1, "Get Input Container - rc:  %u\n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }

  fprintf(file1, "Input Container Name: %s\n",
          FmcjReadOnlyContainerType(input, stringBuffer, 4097));
```

Figure 65. Sample activity implementation (C-language) (Part 2 of 4)

```
/*-- retrieve the output container from the PEA who started the program -*/
  rc = FmcjContainerOutContainer( &output );
  fprintf(file1, "Get Output Container - rc:  %u\n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }

  fprintf(file1, "Output Container Name: %s\n",
          FmcjReadWriteContainerType(output, stringBuffer, 4097));
```

Figure 65. Sample activity implementation (C-language) (Part 3 of 4)

Examples

```
/*----- Modify output values -------------------------------------------*/
  rc= FmcjReadWriteContainerSetLongValue(output, "aFieldInTheOutput",42);
  fprintf(file1, "\nSetting long value returns rc: %u\n", rc);

  ...

/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainerSetOutContainer( output );
  fprintf(file1, "\nSet Output Container - rc: %u\n",rc);
  fflush(file1);

  FmcjGlobalDisconnect();
  fclose(file1);
  return 0;                            // _RC passed to MQ Workflow
}
```

Figure 65. Sample activity implementation (C-language) (Part 4 of 4)

Programming an executable (C++)

```
#include <fstream.h>
#include <bool.h>                      // bool
#include <fmcjstr.hxx>                 // string
#include <vector.h>                    // vector
#include <fmcjpcon.hxx>                // MQ Workflow Container API
int main()
{
/*- keep results in a file ----------------------------------------------*/
  ofstream file1("sample.out");
  if ( file1 == 0 )
    return -1;

  file1 << "\n----- C++-API Activity Implementation called -----\n" << endl;
```

Figure 66. Sample activity implementation (C++) (Part 1 of 4)

```
  FmcjGlobal::Connect();

/*-- retrieve the input container from the PEA who started the program --*/
  FmcjReadOnlyContainer input;

  APIRET rc = FmcjContainer::InContainer( input );
  file1 << "Get Input Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }

  file1 << "Input Container Name: " << input.Type() << endl;
```

Figure 66. Sample activity implementation (C++) (Part 2 of 4)

```
/*-- retrieve the output container from the PEA who started the program -*/
  FmcjReadWriteContainer  output;

  rc = FmcjContainer::OutContainer( output );
  file1 << "Get Output Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }

  file1 << "Output Container Name: " << output.Type() << endl;
/*----- Modify output values ------------------------------------------*/
  rc= output.SetValue("aFieldInTheOutput",42L);
  file1 << "Setting long value returns rc: " << rc << endl;

  ...
```

Figure 66. Sample activity implementation (C++) (Part 3 of 4)

```
/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainer::SetOutContainer( output );
  file1 << "Set Output Container - rc: " << rc << endl;

  FmcjGlobal::Disconnect();
  file1.close();
  return 0;                           // _RC passed to MQ Workflow
}
```

Figure 66. Sample activity implementation (C++) (Part 4 of 4)

Examples

Programming an executable (Java)

```
import com.ibm.workflow.api.*;

// Various needed classes

import java.io.*;
import java.util.*;

public class ActivityImplementation {
  public static PrintWriter out;

  public static void main(String args[])
  {

    try
    { out = new PrintWriter(
                new BufferedWriter(new FileWriter("ActivityImplementation.log")));
    }
    catch (IOException e) {}

    // Maximum nesting depth is 10
    ContainerElement[][] members = new ContainerElement[10][];
    // Maximum element number is 200
    ContainerElement[][] leaves = new ContainerElement[200][];

    String membername;
    String membertype;

    String strvalbuf;
    byte[] binvalbuf = { 0,1,2,3,4,5,6,7,8,9,10 };
    int lvalbuf      = 0;
    double fvalbuf   = 0.0;
    String tInfo;
    int j = 0, k = 0, l = 0;
    int index = 0;
     Calendar ltime = Calendar.getInstance();

  int param = -1;
  int pimreturn = 0;

  String setstr ="";
  int setlong = 0;
  double setdbl = 0;

  ltime.setTime(new Date());

  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  boolean again = false;
  String choice;
```

Figure 67. Sample activity implementation (Java) (Part 1 of 25)

```
/*-------------------------------------------------------------------*/
/* Check command line parameters                                     */
/*-------------------------------------------------------------------*/

  if (args.length!=1)
  {
    System.out.println("\nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>\n"
                       + "-rx: x specifies the return code of the program\n\n"
                       + "Program output is written to ActivityImplementation.log\n");
    System.exit(-1);
  }

  try { // general try clause for whole example

  Agent agent = new Agent();
  agent.setLocator(Agent.LOC_LOCATOR);
  agent.setName("LOCAL");

  ExecutionService service = agent.locate("","");
  System.out.println();

  if (args[0].equals("-r0")) param=0;
  if (args[0].equals("-r1")) param=1;
  if (args[0].equals("-r2")) param=2;
  if (args[0].equals("-r3")) param=3;
  if (args[0].equals("-r4")) param=4;
  if (args[0].equals("-r5")) param=5;

  switch (param)
  {
   case -1:
   System.out.println("\nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>\n"
                      + "-rx: x specifies the return code of the program\n\n"
                      + "Program output is written to ActivityImplementation.log\n");
   System.exit(-1);
   break;
   case 0: pimreturn = 0; break;
   case 1: pimreturn = 1; break;
   case 2: pimreturn = 2; break;
   case 3: pimreturn = 3; break;
   case 4: pimreturn = 4; break;
   case 5: pimreturn = 5; break;
  }

  out.write("\n\n" + "-----------------------------------------------------\n"
        + "API Tutorial - Activity implementation example program.\n"
        + "Called at : " + ltime.getTime().toString() + "\n"
        + "Desired RC: " + pimreturn + "\n\n");
```

Figure 67. Sample activity implementation (Java) (Part 2 of 25)

Examples

```
/*********************************************************************/
/* First we cope with the activity's INPUT container:                */
/*  - get the container                                              */
/*  - get the leaves of the container                                */
/*  - display the member values                                      */
/*********************************************************************/

/*----- Get Input Container -------------------------------------------*/

  ReadOnlyContainer inctnr;
  ExecutionAgent eAgent = agent.getExecutionAgent();

  inctnr = eAgent.inContainer();

  out.write("\nReceived input container '" + inctnr.type() + "'\n");
  System.out.println("\nReceived input container '" + inctnr.type() + "'\n");

/*----- Get Leave Count -----------------------------------------------*/

  int ulLeafCount = inctnr.allLeafCount();

  out.write("Input container AllLeafCount() =  " + ulLeafCount + "\n");

/*----- Get the Leaves ------------------------------------------------*/

  leaves[0]=inctnr.allLeaves();

  if (ulLeafCount != leaves[0].length)
  {
    out.write("LeafCount - vector size mismatch: "
          + "ulLeafCount = " + ulLeafCount
          + " size = "       + leaves[0].length + "\n");
  }
```

Figure 67. Sample activity implementation (Java) (Part 3 of 25)

```
/*----- Show the data members ------------------------------------------*/

  out.write("Input container leaves:\n");

  ContainerElement element = leaves[0][0];

  for (j=0; j < ulLeafCount; j++)
  {
    membername = element.fullName();
    membertype = element.type();

    if (membertype.equals("STRING"))
    {
      try
      {
        strvalbuf = element.getString();
        out.write("STRING '" + membername + "'= " + strvalbuf  + "\n");
      }
      catch (FmcException e)
      {
       switch (e.rc)
       {
         case FmcException.FMC_ERROR_MEMBER_NOT_SET:
           out.write("STRING '" + membername  + "'= <not set>" + "\n");
           break;
         default:
           out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
           break;
       }
      }
    } /* end if STRING */
```

Figure 67. Sample activity implementation (Java) (Part 4 of 25)

```
    if (membertype.equals("LONG"))
    {
      try
      {
        lvalbuf = element.getLong();
        out.write("LONG '" + membername + "'= " + lvalbuf  + "\n");
      }
      catch (FmcException e)
      {
       switch (e.rc)
       {
         case FmcException.FMC_ERROR_MEMBER_NOT_SET:
           out.write("LONG '" + membername  + "'= <not set>" + "\n");
           break;
         default:
           out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
           break;
       }
      }
    } /* end if LONG */
```

Figure 67. Sample activity implementation (Java) (Part 5 of 25)

Examples

```
if (membertype.equals("FLOAT"))
{
  try
  {
    fvalbuf = element.getDouble();
    out.write("FLOAT '" + membername + "'= " + fvalbuf  + "\n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("FLOAT '" + membername  + "'= <not set>" + "\n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
       break;
   }
  }
} /* end if FLOAT */
```

Figure 67. Sample activity implementation (Java) (Part 6 of 25)

```
if (membertype.equals("BINARY"))
{
  try
  {
    binvalbuf = element.getBuffer();
    out.write("BINARY '" + membername + "'= <not shown>\n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("BINARY '" + membername  + "'= <not set>" + "\n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "\n");
       break;
   }
  }
 } /* end if BINARY */

element = leaves[0][j];

} /* end for */
```

Figure 67. Sample activity implementation (Java) (Part 7 of 25)

```
/*********************************************************************/
/* Now we process the activity's OUTPUT container, but only if the      */
/* program was not started with the -s switch                           */
/*  - get the container                                                 */
/*  - navigate through the ContainerElement levels to the leaves        */
/*  - modify the member values                                          */
/*  - send the container to the server                                  */
/*********************************************************************/

/*----- Get Output Container ------------------------------------------*/

  ReadWriteContainer outctnr = eAgent.outContainer();

  out.write("Received output container '" + outctnr.type() + "'\n");
  System.out.println("Received output container '" + outctnr.type() + "'");


/*----- Navigate through the ContainerElement structures --------------*/

  int level            = 0;  // Current nesting level
  int start            = 0;  // Start "stack pointer" of vectors
  int current          = 0;  // End "stack pointer" of vectors
  boolean AllLeavesReached = false;

  // Get the number of 1st level container elements

  int ulMemberCount = outctnr.memberCount();

  if (ulMemberCount == 0)       // No data structure at all, error?
  {
    AllLeavesReached = true;
  }
  else                          // Get the vector of 1st level container elements
  {
    members[level] = outctnr.structMembers();
  }

  while (AllLeavesReached == false)
  {
    out.write("Number of members: " + ulMemberCount + "\n");

    // Check consistency

    if (members[level].length != ulMemberCount)
    {
      out.write("MemberCount - Vector Size mismatch: "
            + "ulMemberCount = " + ulMemberCount
            + " Size = "         + members[level].length + "\n");
    }

    element = members[level][0];

    out.write("IsStruct-IsLeaf-IsArray-Cardinality-Membername\n");
```

Figure 67. Sample activity implementation (Java) (Part 8 of 25)

Examples

```
for (j=0; j < ulMemberCount; j++)
    {
      out.write("       " + element.isStruct());

      if (element.isStruct())
      {
        // Put the next nesting level of this data structure "on the stack"
        leaves[current] = element.structMembers();
        current++;
      }

      out.write("      " + element.isLeaf() + "       " + element.isArray()
            + "           " + element.cardinality() + " "
            + element.fullName() + "\n");

      element = members[level][j];

    } /* end for */

    if (start >= current)        // Nothing "on the stack"
    {
      AllLeavesReached = true;
    }
    else
    {
      level++;
      // Get the next vector from the stack and increase stack pointer
      members[level] = leaves[start];
      start++;
      ulMemberCount = members[level].length;
    }

  } /* end while */
```

Figure 67. Sample activity implementation (Java) (Part 9 of 25)

```
/*----- Modify the data members ----------------------------------------*/

  out.write("\n\nSetting the data members...\n");
  System.out.println("\nSetting the data members...\n");

  leaves[0] = outctnr.leaves();

  ulLeafCount = leaves[0].length;

  try
  {

    for (j=0; j < ulLeafCount; j++)
    {
      element = leaves[0][j];

      membername = element.fullName();
      membertype = element.type();

      if (membertype.equals("STRING"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
               // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
```

Figure 67. Sample activity implementation (Java) (Part 10 of 25)

```
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }

          System.out.print("Enter a value for STRING array member '"
               + membername + "[" + index + "]'? [y/n] ");

          setstr = in.readLine();
```

Figure 67. Sample activity implementation (Java) (Part 11 of 25)

Examples

```
      if (setstr.equalsIgnoreCase("y"))
      {
        System.out.print("Value: ");
        setstr = in.readLine();

        outctnr.setString2(membername,index,setstr);
        strvalbuf = outctnr.getString2(membername,index);

        out.write("Setting ArrayStringValue '" + membername + "[" + index +
                  "]'=" + strvalbuf + "\n");
        System.out.println("Setting ArrayStringValue '" + membername + "[" +
                           index + "]' = " + strvalbuf + "\n");

      }
    }
```

Figure 67. Sample activity implementation (Java) (Part 12 of 25)

```
    else
    {

      System.out.print("Enter a value for STRING member '"
           + membername + "'? [y/n] ");
      setstr = in.readLine();

      if (setstr.equalsIgnoreCase("y"))
      {
        System.out.print("Value: ");
        setstr = in.readLine();

        outctnr.setString(membername,setstr);
        strvalbuf = outctnr.getString(membername);

        out.write("Setting StringValue '" +
                    membername + "'= " + strvalbuf + "\n");
        System.out.println("Setting StringValue '" + membername +
                           "'= " + strvalbuf + "\n");
      }
    }
  } /* end if STRING */
```

Figure 67. Sample activity implementation (Java) (Part 13 of 25)

```
if (membertype.equals("LONG"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
          // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }

          System.out.print("Enter a value for LONG array member '"
               + membername + "[" + index + "]'? [y/n] ");

          setstr = in.readLine();
```

Figure 67. Sample activity implementation (Java) (Part 14 of 25)

```
           if (setstr.equalsIgnoreCase("y"))
          {
            do
            {
              again=false;
              System.out.print("\nValue: ");
              choice = in.readLine();
              try { setlong = Integer.parseInt(choice); }
              catch (NumberFormatException e) { again=true; }
               } while (again);

            outctnr.setLong2(membername,index,setlong);
            lvalbuf = outctnr.getLong2(membername,index);

            out.write("Setting ArrayLongValue '" + membername + "[" + index +
                      "]' = " + lvalbuf + "\n");
            System.out.println("Setting ArrayLongValue '" + membername + "[" +
                               index + "]' = " + lvalbuf + "\n");
          }
        }
```

Figure 67. Sample activity implementation (Java) (Part 15 of 25)

Examples

```
else
 {

   System.out.print("Enter a value for LONG member '"
        + membername + "'? [y/n] ");

   setstr = in.readLine();

   if (setstr.equalsIgnoreCase("y"))
   {
     do
     {
       again=false;
       System.out.print("\nValue: ");
       choice = in.readLine();
       try { setlong = Integer.parseInt(choice); }
       catch (NumberFormatException e) { again=true; }
        } while (again);

     outctnr.setLong(membername,setlong);
     lvalbuf = outctnr.getLong(membername);

     out.write("Setting LongValue '" +membername+ "' = " +lvalbuf+ "\n");
     System.out.println("Setting LongValue '" +membername+"' = " +
                        lvalbuf + "\n");
   }
 }
} /* end if LONG */
```

Figure 67. Sample activity implementation (Java) (Part 16 of 25)

```
if (membertype.equals("FLOAT"))
     {
       if ( membername.endsWith("]") || membername.endsWith(")") )

       // Array element, take the array function
       {
              // Remember the index and remove the suffix from the name

         if (membername.endsWith("]"))
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                              membername.indexOf("]")-1));
           }
           catch (NumberFormatException e) {}
         }
         else
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                              membername.indexOf(")")-1));
           }
           catch (NumberFormatException e) {}
         }

         System.out.print("Enter a value for FLOAT array member '"
              + membername + "[" + index + "]'? [y/n] ");

         setstr = in.readLine();
```

Figure 67. Sample activity implementation (Java) (Part 17 of 25)

```
         if (setstr.equalsIgnoreCase("y"))
         {
           do
           {
             again=false;
             System.out.print("\nValue: ");
             choice = in.readLine();
             try { setdbl = (Double.valueOf(choice)).doubleValue(); }
             catch (NumberFormatException e) { again=true; }
              } while (again);

           outctnr.setDouble2(membername,index,setdbl);
           fvalbuf = outctnr.getDouble2(membername,index);

           out.write("Setting ArrayFloatValue '" + membername + "[" + index +
                     "]' = " + fvalbuf + "\n");
           System.out.println("Setting ArrayFloatValue '" + membername + "[" +
                              index + "]' = " + fvalbuf + "\n");
         }
       }
```

Figure 67. Sample activity implementation (Java) (Part 18 of 25)

Examples

```
else
   {
     System.out.print("Enter a value for FLOAT member '"
          + membername + "'? [y/n] ");

     setstr = in.readLine();

     if (setstr.equalsIgnoreCase("y"))
     {
       do
       {
         again=false;
         System.out.print("\nValue: ");
         choice = in.readLine();
         try { setdbl = (Double.valueOf(choice)).doubleValue(); }
         catch (NumberFormatException e) { again=true; }
          } while (again);

       outctnr.setDouble(membername,setdbl);
       fvalbuf = outctnr.getDouble(membername);

       out.write("Setting FloatValue '" +membername+ "' = " +fvalbuf+ "\n");
       System.out.println("Setting FloatValue '" + membername + "' = " +
                          fvalbuf + "\n");
     }
   }
 } /* end if FLOAT */
```

Figure 67. Sample activity implementation (Java) (Part 19 of 25)

```
if (membertype.equals("BINARY"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )

        // Array element, take the array function
        {
               // Remember the index and remove the suffix from the name

          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }

          outctnr.setBuffer2(membername,index,binvalbuf);
          binvalbuf = outctnr.getBuffer2(membername,index);

          out.write("Setting ArrayBinaryValue '" + membername + "[" + index +
                    "]' = <not shown>\n");
          System.out.println("Setting ArrayBinaryValue '" + membername + "[" +
                             index + "]' = <not shown>\n");
        }
```

Figure 67. Sample activity implementation (Java) (Part 20 of 25)

```
        else
        {
          outctnr.setBuffer(membername,binvalbuf);
          binvalbuf = outctnr.getBuffer(membername);

          out.write(
          "Setting BinaryValue '" + membername + "' = <not shown>\n");
          System.out.println(
          "Setting BinaryValue '" + membername + "' = <not shown>\n");
        }
      } /* end if BINARY */

    } /* end for */
  }/* End try*/
  catch (FmcException e)
  {
  }/* End catch*/
```

Figure 67. Sample activity implementation (Java) (Part 21 of 25)

Examples

```
/*--------------------------------------------------------------------*/
/* Send output container to PEA                                       */
/*--------------------------------------------------------------------*/
  System.out.println("\nSetting output container.");
  out.write("\nSetting output container.\n");
  eAgent.setOutContainer(outctnr);
```

Figure 67. Sample activity implementation (Java) (Part 22 of 25)

```
/*--------------------------------------------------------------------*/
/* Logoff and deinit the API environment.                             */
/*--------------------------------------------------------------------*/

  System.out.println("\nLogging off.");
  out.write("\nLogging off.\n");
  service.logoff();
  out.flush();
  out.close();
  System.exit(pimreturn);

  } // end of general try clause
```

Figure 67. Sample activity implementation (Java) (Part 23 of 25)

```
  catch(FmcException e)
  {
    int c;

    // Catch and report details about the FmcException
    System.out.println("FmcException occured");
    System.out.println("  RC         : " + e.rc);
    System.out.println("  Origin     : " + e.origin);
    System.out.println("  MessageText: " + e.messageText);
    System.out.println("  Exception  : " + e.getMessage());
    System.out.println("  Parameters : ");
    for ( c = 0; c < e.parameters.length ; c++)
    {
      System.out.println("    " + e.parameters[c] );
    }
    System.out.println("  StackTrace : ");
    e.printStackTrace();

    out.write("FmcException occured");
    out.write("\n  RC         : " + e.rc);
    out.write("\n  Origin     : " + e.origin);
    out.write("\n  MessageText: " + e.messageText);
    out.write("\n  Exception  : " + e.getMessage());
    out.write("\n  Parameters : ");
    for ( c = 0; c < e.parameters.length ; c++)
    {
      out.write("    " + e.parameters[c] );
    }
    out.write("\n  StackTrace : ");
    e.printStackTrace(out);
    out.flush();
    out.close();

  }
```

Figure 67. Sample activity implementation (Java) (Part 24 of 25)

```
  catch(Exception e)
  {
    // Catch and report any exception that occurred.
    e.printStackTrace();
    e.printStackTrace(out);
    out.flush();
    out.close();
  }

} // end of void main

} // end of ActivityImplementation
```

Figure 67. Sample activity implementation (Java) (Part 25 of 25)

Examples

Programming an activity implementation (COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "EXEC".

 DATA DIVISION.


   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcrcs.

   01 stringBuffer  PIC X(4097).
   01 fieldName     PIC X(39) VALUE z"aFieldInTheOutput".

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      DISPLAY "-- COBOL-API Activity Implementation called --"
      PERFORM FmcjGlobalConnect.

* retrieve the input container
      PERFORM FmcjCInCtnr.
      MOVE intReturnValue TO retCode.
      DISPLAY "Get Input Container - rc: " retCode.
      IF retCode NOT = FMC-OK
         MOVE 1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING stringBuffer containerTypeBuffer.
      MOVE 4097 TO bufferLength.
      SET hdlContainer TO inputValue.
      PERFORM FmcjROCType.
      DISPLAY "Input Container Name: " stringBuffer.

* retrieve the output container
      PERFORM FmcjCOutCtnr.
      MOVE intReturnValue TO retCode.
      DISPLAY "Get Output Container - rc: " retCode.
      IF retCode NOT = FMC-OK
         MOVE 1 TO retCode
         GOBACK
      END-IF

      SET hdlContainer TO outputValue.
      PERFORM FmcjRWCType.
      DISPLAY "Output Container Name: " stringBuffer.
```

Figure 68. Sample activity implementation (COBOL, via PERFORM) (Part 1 of 2)

```
* modify output values
      MOVE 42 TO intValue.
      CALL "SETADDR" USING fieldName qualifiedName.
      PERFORM FmcjRWCSetLongValue.
      MOVE intReturnValue TO retCode.
      DISPLAY "Setting long value returns rc: " retCode.

* return the output container
      PERFORM FmcjCSetOutCtnr.
      MOVE intReturnValue TO retCode.
      DISPLAY "Set Output Container - rc: " retCode.

      PERFORM FmcjGlobalDisconnect.
      MOVE FMC-OK TO retCode.
      GOBACK.

      COPY fmcperf.
```

Figure 68. Sample activity implementation (COBOL, via PERFORM) (Part 2 of 2)

Examples

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. "EXEC".

 DATA DIVISION.

   WORKING-STORAGE SECTION.

   COPY fmcvars.
   COPY fmcrcs.

   01 stringBuffer  PIC X(4097).
   01 fieldName     PIC X(39) VALUE z"aFieldInTheOutput".

   LINKAGE SECTION.

   01 retCode            PIC S9(9) BINARY.

 PROCEDURE DIVISION USING retCode.

      DISPLAY "-- COBOL-API Activity Implementation called --"
      CALL "FmcjGlobalConnect".

* retrieve the input container
      CALL "FmcjContainerInContainer"
         USING BY REFERENCE inputValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode.
      DISPLAY "Get Input Container - rc: " retCode.
      IF retCode NOT = FMC-OK
         MOVE 1 TO retCode
         GOBACK
      END-IF

      CALL "SETADDR" USING stringBuffer containerTypeBuffer.
      MOVE 4097 TO bufferLength.
      SET hdlContainer TO inputValue.
      CALL "FmcjContainerType"
         USING BY VALUE hdlContainer
                        containerTypeBuffer
                        bufferLength
         RETURNING pointerReturnValue.
      DISPLAY "Input Container Name: " stringBuffer.

* retrieve the output container
      CALL "FmcjContainerOutContainer"
         USING BY REFERENCE outputValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode.
      DISPLAY "Get Output Container - rc: " retCode.
      IF retCode NOT = FMC-OK
         MOVE 1 TO retCode
         GOBACK
      END-IF

      SET hdlContainer TO outputValue.
      CALL "FmcjContainerType"
         USING BY VALUE hdlContainer
                        containerTypeBuffer
                        bufferLength
         RETURNING pointerReturnValue.
      DISPLAY "Output Container Name: " stringBuffer.

* modify output values
      MOVE 42 TO intValue.
      CALL "SETADDR" USING fieldName qualifiedName.
```

Figure 69. Sample activity implementation (COBOL, via CALL) (Part 1 of 2)

```
      CALL "FmcjReadWriteContainerSetLongValue"
         USING BY VALUE hdlContainer
                        qualifiedName
                        intValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode.
      DISPLAY "Setting long value returns rc: " retCode.

* return the output container
      CALL "FmcjContainerSetOutContainer"
         USING BY VALUE outputValue
         RETURNING intReturnValue.
      MOVE intReturnValue TO retCode.
      DISPLAY "Set Output Container - rc: " retCode.

      CALL "FmcjGlobalDisconnect".
      MOVE FMC-OK TO retCode.
      GOBACK.

      COPY fmcperf.
```

Figure 69. Sample activity implementation (COBOL, via CALL) (Part 2 of 2)

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER

EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples can include the names of individuals, companies, brands, or products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information in an online form, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

- AIX
- CICS
- DB2
- FlowMark
- IBM
- IMS
- Language Environment
- MQSeries
- MVS
- OS/2
- OS/390
- RISC System/6000
- z/OS

Lotus Notes is a registered trademark, and Domino and Lotus Go Webserver are trademarks of Lotus Development Corporation.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations used in MQSeries Workflow for OS/390 publications. If you do not find the term you are looking for, refer to the index of the appropriate manual or view the *IBM Dictionary of Computing* located at:

`http://www.ibm.com/networking/nsg/nsgmain.htm`

A

administration server. The MQ Workflow component that performs administration functions within an MQ Workflow system. Functions include starting and stopping of the MQ Workflow system, performing error management, and participating in administrative functions for a system group.

activity. One of the steps that make up a process model. This can be a program activity, process activity, or block activity.

activity information member. A predefined data structure member associated with the operating characteristics of an activity.

API. Application Programming Interface.

applet. An application program, written in the Java programming language, that can be retrieved from a Web server and executed by a Web browser. A reference to an applet appears in the markup for a Web page, in the same way that a reference to a graphics file appears; a browser retrieves an applet in the same way that it retrieves a graphics file. For security reasons, an applet's access rights are limited in two ways: the applet cannot access the file system of the client upon which it is executing, and the applet's communication across the network is limited to the server from which it was downloaded. Contrast with servlet.

application programming interface. An interface provided by the MQ Workflow workflow manager that enables programs to request services from the MQ Workflow workflow manager.

asynchronous API call. A particular service of the API that allows programs to register functions with the API that are invoked when a defined event is found by MQ Workflow.

audit trail. A relational table in the database that contains an entry for each major event during execution of a process instance.

authorization. The attributes of a user's staff definition that determine the user's level of authority in MQ Workflow. The system administrator is allowed to perform all functions.

B

backward mapping. Conversion of output data created by an OS/390 legacy application into an MQSeries Workflow container. This conversion is performed by the *program execution server's program mapper*.

backward mapping definition. Part of the *MDL* which connects an interface definition and structure definition.

bend point. A point at which a connector starts, ends, or changes direction.

block activity. A composite activity that consists of a group of activities, which can be connected with control and data connectors. A block activity is used to implement a Do-Until loop; all activities within the block activity are processed until the exit condition of the block activity evaluates to true. See also *composite activity.*

Buildtime. An MQ Workflow component with a graphical user interface for creating and maintaining workflow models, administering resources, and the system network definitions.

C

cardinality. (1) An attribute of a relationship that describes the membership quantity. There are four types of cardinality: One-to-one, one-to-many, many-to-many, and many-to-one. (2) The number of rows in a database table or the number of different values in a column of a database table.

child organization. An organization within the hierarchy of administrative units of an enterprise that has a parent organization. Each child organization can have one parent organization and several child organizations. The parent is one level above in the hierarchy. Contrast with *parent organization*.

cleanup server. The MQ Workflow component that physically deletes information in the MQ Workflow Runtime database, which had only been deleted logically.

composite activity. An activity which is composed of other activities. Composite activities are block activities and bundle activities.

container API. An MQ Workflow API that allows programs executing under the control of MQ Workflow to obtain data from the input and output container of the activity and to store data in the output container of the activity.

control connector. Defines the potential flow of control between two nodes in the process. The actual flow of control is determined at run time based on the truth value of the transition conditions associated with the control connector.

coordinator. A predefined role that is automatically assigned to the person designated to coordinate a role.

CPIC. An invocation type that allows the program execution server to run an application synchronously on an IMS service. CPIC is based on IMS/APPC.

D

data connector. Defines the flow of data between containers.

data container. Storage for the input and output data of an activity or process. See *input container* and *output container*.

data mapping. Specifies, for a data connector, which fields from the associated source container are mapped to which fields in the associated target container.

data structure. A named entity that consists of a set of data structure members. Input and output containers are defined by reference to a data structure and adopt the layout of the referenced data structure type.

data structure member. One of the variables of which a data structure is composed.

default control connector. The graphical representation of a standard control connector, shown in the process diagram. Control flows along this connector if no other control path is valid.

document type definition (DTD). The rules, determined by an application, that apply SGML to the markup language of documents of a particular type. SGML provides the syntax for the markup language, and the DTD provides the vocabulary for the markup language.

domain. A set of MQ Workflow system groups which have the same meta-model, share the same staff information, and topology information. Communication between the components in the domain is via message queuing.

dynamic staff assignment. A method of assigning staff to an activity by specifying criteria such as role, organization, or level. When an activity is ready, the users who meet the selection criteria receive the activity to be worked on. See also *level*, *organization*, *process administrator*, and *role*.

E

end activity. An activity that has no outgoing control connector.

EXCI. An invocation type that allows the program execution server to run an application synchronously on a CICS service. EXCI is based on the CICS External CICS Interface provided by CICS Version 4.1 and higher to allow non-CICS applications to call programs running under CICS.

execution server. The MQ Workflow component that performs the processing of process instances at runtime.

exit condition. A logical expression that specifies whether an activity is complete.

export. An MQ Workflow utility program for retrieving information from the MQ Workflow database and making it available in MQ Workflow Definition Language (FDL) or HTML format. Contrast with *import*.

F

fixed member. A predefined data structure member that provides information about the current activity. The value of a fixed member is set by the MQ Workflow workflow manager.

(FDL) MQ Workflow Definition Language. The language used to exchange MQ Workflow information between MQ Workflow system groups. The language is used by the import and export function of MQ Workflow and contains the workflow definitions for staff, programs, data structures, and topology. This allows non-MQ Workflow components to interact with MQ Workflow. See also *export* and *import*.

FlowMark. The predecessor of MQSeries Workflow.

fork activity. An activity that is the source of multiple control connectors.

forward mapping. Conversion of MQSeries Workflow containers into a format accepted by an OS/390 legacy application. This conversion is performed by the *program execution server's program mapper*.

forward mapping definition. Part of the *MDL* which connects a structure definition and interface definition.

fully-qualified name. A qualified name that is complete; that is, one that includes all names in the

hierarchical sequence above the structure member to which the name refers, as well as the name of the member itself.

I

import. An MQ Workflow utility program that accepts information in the MQ Workflow definition language (FDL) format and places it in an MQ Workflow database. Contrast with *export*.

input container. Storage for data used as input to an activity or process. See also *source* and *data mapping*.

interface. The definition of the data structure accepted by an OS/390 CICS or IMS legacy application. This definition is used by the *program mapper* to convert data to (and from) an MQSeries Workflow program's *structure*.

interface definition. Part of the *MDL* which defines the interface used by a legacy application.

interface element. Part of an *interface* definition. An interface element has a name, a type and a cardinality. It is mapped onto a *structure* element by a *mapping rule*.

invocation exit. The DLL specified by the invocation type. The exit is based on an invocation protocol like CICS External CICS Interface, IMS/APPC or the MQSeries CICS and IMS bridges.

invocation protocol. The way the PES connects to a service like CICS or IMS in order to invoke a program on that service.

invocation type. The way the program execution server connects to a service system (like CICS or IMS) in order to invoke a program on that service. The invocation type is part of a program mapping execution request sent to the PES. To invoke a program, the PES loads the appropriate invocation exit as defined for the invocation type. Invocation types include *EXCI* and *CPIC*.

L

level. A number from 0 through 9 that is assigned to each person in an MQ Workflow database. The person who defines staff in Buildtime can assign a meaning to these numbers such as rank and experience. Level is one of the criteria that can be used to dynamically assign activities to people.

local user. Identifies a user during staff resolution whose home server is in the same system group as the originating process.

local subprocess. A subprocess that is processed in the same MQ Workflow system group as the originating process.

logical expression. An expression composed of operators and operands that, when evaluated, gives a result of true, false, or an integer. (Nonzero integers are equivalent to false.) See also *exit condition* and *transition condition*.

M

manager. A predefined role that is automatically assigned to the person who is defined as head of an organization.

mapping definition language. The language used to define *mapping definitions* for the *program mapping exit*.

mapping exit. Used by the PES to convert data between MQSeries Workflow and legacy applications. The exit is defined by a mapping type defined in the *PES directory* and in *Buildtime*. The exit is only called if mapping has been enabled in *Buildtime*.

mapping rules. Part of a *forward mapping* or *backward mapping* definition that defines the mapping between individual *interface elements* and *structure elements*. Mapping rules are defined using the *mapper definition language*.

mapping type. The name used to identify which mapping exit to use. The mapping type is defined in the *PES directory* and must match the *Buildtime* definitions for the legacy application. The mapping type provided with MQSeries Workflow for OS/390 is named **DEFAULT**.

MDL. See *mapping definition language*.

message queuing. A communication technique that uses asynchronous messages for communication between software components.

MQCICS. An invocation type that allows the program execution server to run an application asynchronously on a CICS service. The corresponding invocation exit uses the MQSeries CICS Bridge as invocation protocol.

MQIMS. An invocation type that allows the program execution server to run an application asynchronously on an IMS service. The corresponding invocation exit uses the invocation protocol MQSeries IMS Bridge.

N

navigation. Movement from a completed activity to subsequent activities in a process. The paths followed are determined by control connectors, their associated transition conditions, and by the start conditions of activities. See also *control connector*, *exit condition*, *transition condition*, and *start condition*.

node. (1) The generic name for activities within a process diagram. (2) The operating system image that hosts MQ Workflow systems.

notification. An MQ Workflow facility that can notify a designated person when a process or activity is not completed within the specified time.

notification work item. A work item that represents an activity or process notification.

O

organization. An administrative unit of an enterprise. Organization is one of the criteria that can be used to dynamically assign activities to people. See *child organization* and *parent organization*.

output container. Storage for data produced by an activity or process for use by other activities or for evaluation of conditions. See also *sink*.

P

parent organization. An organization within the hierarchy of administrative units of an enterprise that has one or more child organizations. A child is one level below its parent in the hierarchy. Contrast with child *child organization*.

parent process. A process instance that contains the process activity which started the process as a subprocess.

pattern activity. A single and simple activity in a bundle activity from which multiple instances, called pattern activity instances, are created at run time.

person (pl. people). A member of staff in an enterprise who has been defined in the MQ Workflow database.

PES. See *program execution server*.

PES directory. See *program execution server directory*.

predefined data structure member. A data structure member predefined by MQ Workflow and used for communication between user applications and MQ Workflow Runtime.

process. Synonymously used for a process model and a process instance. The actual meaning is typically derived from the context.

process activity. An activity that is part of a process model. When a process activity is executed, an instance of the process model is created and executed.

process administrator. A person who is the administrator for a particular process instance. The administrator is authorized to perform all operations on a process instance. The administrator is also the target for staff resolution and notification.

process category. An attribute that a process modeler can specify for a process model to limit the set of users who are authorized to perform functions on the appropriate process instances.

process definition. Synonym for *process model*.

process diagram. A graphical representation of a process that shows the properties of a process model.

process instance. An instance of a process to be executed in MQ Workflow Runtime.

process instance list. A set of process instances that are selected and sorted according to user-defined criteria.

process instance monitor. An MQ Workflow client component that shows the state of a particular process instance graphically.

process management. The MQ Workflow Runtime tasks associated with process instances. These consist of creating, starting, suspending, resuming, terminating, restarting, and deleting process instances.

process model. A set of processes represented in a process model. The processes are represented in graphical form in the process diagram. The process model contains the definitions for staff, programs, and data structures associated with the activities of the process. After having translated the process model into a process template, the process template can be executed over and over again. *Workflow model* and *process definition* are synonyms.

process monitor API. An application programming interface that allows applications to implement the functions of a process instance monitor.

process-relevant data. Data that is used to control the sequence of activities in a process instance.

process status. The status of a process instance.

process template. A fixed form of a process model from which process instances can be created. It is the translated form in MQ Workflow Runtime. See also *process instance*.

process template list. A set of process templates that have been selected and sorted according to user-defined criteria.

program. A computer-based application that serves as the implementation of a program activity or as a support tool. Program activities reference executable programs using the logical names associated with the programs in MQ Workflow program registrations. See also *program registration*.

program activity. An activity that is executed by a registered program. Starting this activity invokes the program. Contrast with *process activity*.

program execution agent (PEA). An MQ Workflow component that manages the implementations of program activities for a user in a LAN environment. Each instance of a program execution agent services a single user. MQSeries Workflow for OS/390 does not support program execution agents but rather employs a program execution server.

program execution server (PES). An MQ Workflow component that can manage the implementations of program activities for multiple clients. OS/390 employs a program execution server to implement CICS and IMS programs and to support mapping of data formats between MQSeries Workflow and legacy applications. Multiple instances of a program execution server are possible. In the LAN environment, program execution agents are used instead of program execution servers.

program mapping. Program mapping definitions passed and supported into the mapping database and used by the program mapper at runtime to transform data from legacy applications.

program mapping DB. Database used by the PES exit which contains program mappings imported by the program mapping import tool. Used at runtime by the exit to perform the forward and backward mapping.

program mapping exit. PES exit used to transform MQSeries Workflow for OS/390 containers into a format acceptable by legacy applications and vice versa.

program mapping import tool. Component of the MQSeries Workflow program mapping exit which reads the result of the program mapping parser and inputs the compiled program mapping definitions into the program mapping DB.

program mapping parser. Component of the MQSeries Workflow for OS/390 program mapping exit which parses the MDL and creates an intermediate file which is used by the program mapping import tool.

program registration. Registering a program in MQ Workflow so that sufficient information is available for managing the program when it is executed by MQ Workflow.

R

role. A responsibility that is defined for staff members. Role is one of the criteria that can be used to dynamically assign activities to people.

S

scheduling server. The MQ Workflow component that schedules actions based on time events, such as resuming suspended work items, or detecting overdue processes.

server. The servers that make up an MQ Workflow system are called Execution Server, Administration Server, Scheduling Server, and Cleanup Server.

servlet. An application program, written in the Java programming language, that is executed on a Web server. A reference to a servlet appears in the markup for a Web page, in the same way that a reference to a graphics file appears. The Web server executes the servlet and sends the results of the execution (if there are any) to the Web browser. Contrast with applet.

sink. The symbol that represents the output container of a process or a block activity.

source. The symbol that represents the input container of a process or a block activity.

specific resource assignment. A method of assigning resources to processes or activities by specifying their user IDs.

standard client. The MQ Workflow component, which enables creation and control of process instances, working with worklists and work items, and manipulation of personal data of the logged-on user.

start activity. An activity that has no incoming control connector.

start condition. The condition that determines whether an activity with incoming control connectors can start after all of the incoming control connectors are evaluated.

structure. The definition of the MQSeries Workflow structure passed into or out of an *activity* implementation.

structure definition. Part of the *MDL* which defines the structure used by a program activity.

structure element. Part of an *structure* definition. A structure element has a name, a type and a cardinality. It is mapped onto an *interface* element by a *mapping rule*.

subprocess. A process instance that is started by a process activity.

substitute. The person to whom an activity is automatically transferred when the person to whom the activity was originally assigned is declared as absent.

support tool. A program that end users can start from their worklists in the MQ Workflow MQ Workflow Client to help complete an activity.

symbolic reference. A reference to a specific data item, the process name, or activity name in the description text of activities or in the command-line parameters of program registrations. Symbolic references are expressed as pairs of percent signs (%)

that enclose the fully-qualified name of a data item, or either of the keywords _PROCESS or _ACTIVITY.

system. The smallest MQ Workflow unit within an MQ Workflow domain. It consists of a set of the MQ Workflow servers.

system group. A set of MQ Workflow systems that share the same database.

system administrator. (1) A predefined role that conveys all authorizations and that can be assigned to exactly one person in an MQ Workflow system. (2) The person at a computer installation who designs, controls, and manages the use of the computer system.

T

thin client. A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a PC.

top-level process. A process instance that is not a subprocess and is started from a user's process instance list or from an application program.

transition condition. A logical expression associated with a conditional control connector. If specified, it must be true for control to flow along the associated control connector. See also *control connector*.

translate. The action that converts a process model into a Runtime process template.

trusted program. A program that has been assigned such a characteristic in the FDL (via Buildtime), which enables a PEA or PES to divulge the program ID.

U

Unicode. A system of 16 bit binary codes for text or script characters. Officially called the Unicode Worldwide Character Standard, it is a system for ″the interchange, processing, and display of the written texts of the diverse languages of the modern world″. Unicode is used by the Java programming language for character representation.

user-defined program execution server (UPES). A program listening on a user-defined MQSeries queue and effectively acting as a program execution server by accepting and implementing program invocation requests received from external programs in the form of XML messages.

user ID. An alphanumeric string that uniquely identifies an MQ Workflow user.

user type definition. Part of the *MDL* which defines the interface used by a user type.

user type interface . A user defined interface type. If you need to map a data type that is not supported by the default mapper type, you can define a user type, and write a type conversion program which handles the conversion of the particular data type. This must use the user type exit.

V

verify. The action that checks a process model for completeness.

W

workflow. The sequence of activities performed in accordance with the business processes of an enterprise.

Workflow Management Coalition (WfMC). A non-profit organization of vendors and users of workflow management systems. The Coalition's mission is to promote workflow standards for workflow management systems to allow interoperability between different implementations.

workflow model. Synonym for *process model*.

work item. Representation of work to be done in the context of an activity in a process instance.

work item set of a user. All work items assigned to a user.

worklist. A list of work items and notifications assigned to a user and retrieved from a workflow management system.

worklist view. List of work items selected from a work item set of a user according to filter criteria which are an attribute of a worklist. It can be sorted according to sort criteria if specified for this worklist.

Bibliography

To order any of the following publications, contact your IBM representative or IBM branch office.

MQSeries Workflow for z/OS publications

This section lists the publications included in the MQSeries Workflow for z/OS library.

- *MQSeries Workflow for z/OS: Customization and Administration*, SC33-7030, explains how to customize and administer an MQSeries Workflow for z/OS system.
- *MQSeries Workflow for z/OS: Programming Guide*, SC33-7031, explains the application programming interfaces (APIs) available on z/OS, including program execution server exits.
- *MQSeries Workflow for z/OS: Messages and Codes*, SC33-7032, explains the MQSeries Workflow for z/OS system messages.
- *MQSeries Workflow for z/OS: Program Directory*, GI10-0483, explains how to install MQSeries Workflow for z/OS.

MQ Workflow publications

This section lists the publications included in the MQSeries Workflow library.

- *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 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.

Workflow publications

- *Frank Leymann, Dieter Roller, Production Workflow: Concepts and Techniques* (New Jersey: Prentice Hall PTR, 1999)
- *IBM Systems Journal, Vol. 36. No. 1, 1997 by Frank Leymann, Dieter Roller.* You can also refer to the Internet:

  `http://www.almaden.ibm.com/journal/sj361/leymann.html`
- *Workflow Handbook 1997, published in association with WfMC*. Edited by Peter Lawrence.

MQSeries publications

- *MQSeries System Administration*, SC33-1873, explains administration tasks related to MQSeries.
- *MQSeries Installation*, SH12-6288, discusses the installation of MQSeries.
- *MQSeries System Administration*, SC33-0807, discusses topics related to the application programming interface of MQSeries.

Other useful publications

- *MQSeries Clients*, GC22-1632
- *DB2 for OS/390 Administration Guide*, SC26-8957
- *DB2 for OS/390 SQL Reference*, SC26-8966
- *DB2 for OS/390 Application Programming and SQL Guide*, SC26-8958
- *DB2 for OS/390 Command Reference*, SC26-8960
- *DB2 for OS/390 Utility Guide and Reference*, SC26-8967
- *OS/390 MVS Planning: Workload Management*, GC28-1761
- *OS/390 MVS Programming: Workload Management Services*, GC28-1773
- *W3C Recommendation: Extensible Markup Language (XML) 1.0*, REC-xml-19980210

Licensed books

The licensed books that were declassified in OS/390 Version 2 Release 4 appear on the OS/390 Online Library Collection, SK2T-6700. The remaining licensed books for OS/390 Version 2 appear on the OS/390 Licensed Product library, LK2T-2499, in unencrypted form.

Index

A

B

C

D

E

F

G

N

O

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments — We'd Like to Hear from You

IBM MQ Workflow for z/OS
Programming Guide
Version 3.3

Publication No. SC33-7031-05

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
SC33-7031-05

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC33-7031-05

Cut or Fold
Along Line

Program Number: 5655-BPM

Printed in Denmark by IBM Danmark A/S

SC33-7031-05

Spine information:

IBM MQ Workflow for z/OS MQSeries Workflow for z/OS Programming Guide Version 3.3