The Workflow Management Coalition Specification

Workflow Management Application Programming Interface (Interface 2&3) Specification

Document Number WFMC-TC-1009

July-98
Version 2.0

Copyright (C) 1993, 1999, The Workflow Management Coalition

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the Workflow Management Coalition except that reproduction, storage or transmission without permission is permitted if all copies of the publication (or portions thereof) produced thereby contain a notice that the Workflow Management Coalition and its members are the owners of the copyright therein.

This Specification has been authored by Workflow Management Coalition members.
7.2 Additional Issues .......................................................................................................................... 92
7.1 Additional API Areas ...................................................................................................................... 92
7. APPENDIX A: FUTURE WORK ....................................................................................................... 92
6.8 WAPI Application Invocation Functions ....................................................................................... 86
6.7 WAPI Administration Functions .................................................................................................. 79
6.6 WAPI Worklist Functions ............................................................................................................. 63
6.5 WAPI Activity Status Functions ................................................................................................. 58
6.4 WAPI Process Status Functions .................................................................................................. 53
6.3.6 WMFetchActivityInstanceAttribute ....................................................................................... 49
6.3.7 WMCloseActivityInstanceAttributesList ............................................................................... 50
6.3.8 WMGetActivityInstanceAttributeValue .................................................................................. 51
6.3.9 WMAssignActivityInstanceAttribute .................................................................................... 52
6.6.15 WMAssignWorkItemAttribute ............................................................................................. 78
6.6.14 WMGetWorkItemAttributeValue ........................................................................................... 77
6.6.12 WMFetchWorkItemAttribute ................................................................................................. 75
6.6.11 WMOpenWorkItemAttributesList ......................................................................................... 74
6.6.10 WMReassignWorkItem ........................................................................................................... 73
6.6.9 WMChangeWorkItemState ....................................................................................................... 72
6.6.8 WMCloseWorkItemStatesList ................................................................................................. 71
6.6.7 WMFetchWorkItemState ........................................................................................................... 70
6.6.6 WMOpenWorkItemStatesList .................................................................................................. 68
6.6.5 WMCompleteWorkItem ............................................................................................................ 68
6.6.4 WMGetWorkItem ....................................................................................................................... 67
6.6.3 WMCloseWorkList .................................................................................................................... 66
6.6.2 WMFetchWorkList ..................................................................................................................... 65
6.6.1 WMOpenWorkList .................................................................................................................... 64
6.5.4 WMGetActivityInstance ........................................................................................................... 62
6.5.3 WMCloseActivityInstancesList ............................................................................................... 61
6.5.2 WMFetchActivityInstancesList ............................................................................................... 60
6.5.1 WFMData API calls .................................................................................................................. 59
6.4.1 WMOpenProcessInstancesList ............................................................................................... 54
6.4.2 WMFetchProcessInstance ........................................................................................................ 55
6.4.3 WMCloseProcessInstancesList ............................................................................................... 56
6.4.4 WMGetProcessInstance ........................................................................................................... 57
6.3.8 WMAssignProcessInstancesAttribute .................................................................................... 83
6.3.7 WMChangeProcessInstancesState ......................................................................................... 81
6.3.6 WMGetActivityInstance ........................................................................................................... 79
6.3.5 WMOpenActivityInstancesList ............................................................................................... 61
6.3.4 WMFetchActivityInstance ........................................................................................................ 60
6.3.3 WMCloseActivityInstanceAttributesList ............................................................................... 50
6.3.2 WMGetActivityInstanceAttributeValue .................................................................................. 49
6.3.1 WMAssignActivityInstance ....................................................................................................... 48
6.2.3 Security .................................................................................................................................... 93
6.2.2 Synchpoint processing ............................................................................................................ 92
6.2.1 Error reporting and control ..................................................................................................... 92
6.1.4 Names and Roles ..................................................................................................................... 92
6.1.3 Administration and Maintenance ............................................................................................ 92
6.1.2 WFM Data API calls ................................................................................................................ 92
6.1.1 WFM Data API calls ................................................................................................................ 92
6. APPENDIX A: FUTURE WORK ....................................................................................................... 92
7.1 Additional API Areas ..................................................................................................................... 92
7.2 Additional Issues .......................................................................................................................... 92
7.2.1 Error reporting and control ..................................................................................................... 92
7.2.2 Synchpoint processing ............................................................................................................ 92
7.2.3 Security .................................................................................................................................... 93
11.2 Entity Attribute Manipulation .......................................................... 148
   11.2.1 WMOpenEntityAttributesList ................................................. 149
   11.2.2 WMFetchEntityAttribute ......................................................... 150
   11.2.3 WMCloseEntityAttributesList .................................................. 151
   11.2.4 WMGetEntityAttributeValue ..................................................... 152
   11.2.5 WMOpenEntityAttributeValueList ............................................. 153
   11.2.6 WMFetchEntityAttributeValue .................................................. 154
   11.2.7 WMCloseEntityAttributeValueList ............................................ 155
   11.2.8 WMAssignEntityAttributeValue ............................................... 156
   11.2.9 WMClearEntityAttributeList ................................................... 157
   11.2.10 WMAddEntityAttributeValue ................................................. 158

11.3 Process Modelling Functions .......................................................... 159
   11.3.1 WMOpenWorkflowDefinition .................................................... 160
   11.3.2 WMCloseWorkflowDefinition ................................................... 161
   11.3.3 WMCreateProcessDefinition ................................................. 162
   11.3.4 WMDeleteProcessDefinition ................................................... 163
   11.3.5 WMOpenProcessDefinition ...................................................... 164
   11.3.6 WMCloseProcessDefinition .................................................... 165

11.4 Standard Process Modelling Entity Types ............................................. 165
   11.4.1 Additional Data Types .......................................................... 165

12. APPENDIX G: STATES ........................................................................... 168
    12.1 Process Instance States ............................................................ 168
    12.2 Activity Instance States ............................................................ 169
    12.3 Workitem States ........................................................................... 170
0. Change History

Version 1.0
- Initial version

Version 1.1
- Consistent handling of output parameters as pointers
- Added attributes for WMTProcessDefinition
- Editorial enhancements

Version 1.2
- Added Abstract Object Model
- Added OLE Binding
- Added OMG IDL Binding

Version 2.0 (Beta)
- Added Process Definition functions
- Added States
- Added references to Audit Data
- Added Conformance Specification

Version 2.0 (Beta)
- Added Application Interface Definition
- Added Application Interface functions

Version 2.0e (Beta)
- Added Conformance Profile for WFToolAgent
1. Purpose

The purpose of this document is to specify standard workflow management Application Programming Interfaces (API) which can be supported by WFM products. These API calls provide for a consistent method of access to WFM functions in cross-product WFM Engines. The API set is named Workflow Application Programming Interfaces (WAPI).

This document defines the API specifications of the Workflow Management Coalition for building workflow-enabled applications (Interface 1, 2 and 3 in the Workflow Reference Model).

This document is directly associated to the documents:

- Workflow Management Coalition Glossary
- Workflow Management Coalition Interface 2 WAPI Naming Conventions

The three documents constitute the complete specification.

2. Audience

The intended audience of this document includes all participants in the workflow industry. Comments should be addressed to the Workflow Management Coalition.

3. Overview

The support of these interfaces in WFM products allow the implementation of front-end applications which need to access WFM Engine functions (Workflow services). Such implementations might be written by WFM exploiters or ISVs. Implementation of these API calls are also intended to allow the workflow applications to be adjusted to operate with different WFM Engines using this common API interface.

These API calls should allow a WFM exploiter to have a single end user interface and functions set regardless of the number of WFM products existing in an installation. WAPI calls may be implemented in a number of languages. The first Coalition specification will be for the ‘C’ language. The API operates as CALLS. No assumption is made regarding the underlying implementation of the CALLS in a particular WFM product implementation. The WAPI calls are for use at run-time. That is, when processes are executing or are to be executed. They would normally be used by workflow applications (e.g. worklist handlers, cooperating applications) but may also be used by a WFM Engine when it wishes to interact with another WFM product within the context of the API functions.

Through its set of functions, the WAPI provides a set of workflow services that a Workflow Enactment Service provides. The WAPI does not assume any specific user interface, but rather it specifically assumes that the user interface of the workflow enabled application, that uses these services, provides its own user interface, that depends solely on the application development environment facilities where it is implemented.
The WFM Engine functions can broadly be classified in the following areas:

- WAPI Connection Functions
- WAPI Workflow Definition Functions
- WAPI Process Control Functions
- WAPI Activity Control Functions
- WAPI Process Status Functions
- WAPI Activity Status Functions
- WAPI Worklist Functions
- WAPI Administration Functions

3.1 Application Interface Definition

Introducing a Workflow Management System always implies that at least the existing IT environment has to be integrated, or better “workflow enabled”. Additionally, this interface grants a certain degree of protection on the already installed software systems. The W/MC’s interface to invoke applications does not define a direct application control mechanism. Today, the customers and the vendors are confronted with several different operating systems and application communication mechanisms. Therefore, Workflow Management Systems need an interface to specific application drivers. With the definition of these drivers to invoke and control applications, the Coalition offers an interface which enables a standardized protocol between workflow products and any other software systems.

Currently, a variety of Workflow Management Tools support specialized mechanisms to integrate applications and to exchange information. While all these mechanisms are mostly individually implemented for specific customer requirements, system integration companies and third party vendors have to re-implement these mechanisms, if they intend to use another Workflow Management tool at the same site. Consequently, their interest in supporting the generation of such an interface is, indeed, very high, as it would definitely improve their daily work. It might appear very simple to “workflow enable” common applications, nevertheless, workflow environments typically include a series of different specialized applications, which run in heterogeneous environments. Workflow Management Systems as well as integration platforms are required by the market and require a generalized and standardized application interface.

3.1.1 Purpose & Background

The “Invoking Applications Interface” defines an interface mechanism between Workflow Management Systems and any other application, but it, however, differentiates itself from the other Coalition interface definitions. Invoking an application is not a workflow specific functionality, but a Workflow System would not make much sense without this functionality. Therefore, this interface addresses workflow system vendors as well as any third party software vendor. Based on different communication technologies the so-called “Tool Agents” can handle the application control and information exchange. These Tool Agents represent at least one specific invocation technology. E.g. while one Tool Agent supports DDE commands, others can communicate based on protocols like OLE or CORBA or any other concept. The technology to interact between a Tool Agent and a corresponding application depends on the underlying architecture and on application-specific interfaces, which have to be managed under control of the Tool Agent itself. The suggested interface defines the way a Tool Agent can be used by a workflow application, e.g. a worklist handler or the workflow engine. Finally, the purpose of Tool Agents can be compared with the purpose of standardized software components.
3.2 Design Philosophy
There are a number of design assumptions and constraints that provide a framework or philosophy for the definition of this specification.

3.3 Design Assumptions
Incremental Set of Functions. It is assumed that as the WFM technology evolves, likewise the specifications defined in this document will evolve and will have additions in subsequent versions of this document.

- Strings are defined with buffer sizes allocated in bytes. Strings are assumed to be zero terminated.
- The workflow engine may have security restrictions that may cause an error to be returned to a user for some of the API calls.
- The specific calls to change state have to be supported by all vendors. The generic state changes are reserved for vendor specific states. In the future, it is expected that a common set of states will evolve.
- Each process definition must have a unique ID within an administrative scope.
- Each process instance must have a unique ID within an administrative scope.
- Each activity instance must have a unique ID within a process instance.
- Each work item must have a unique ID within a process instance.
- Process Instance ID is unique to the workflow engines from which it is available. It is the responsibility of the workflow engine to ensure a unique identifier within this scope.

3.4 Design Objectives
Ease of Implementation. The API specification must be easy to implement by a wide range of vendors. This also implies that the specification will be able to be implemented by multiple vendors in a reasonably short period of time.

3.5 Defined Terms and Abbreviations
The terms used in this document are defined in the WFM Coalition Glossary.

3.6 Reference Documents
The following documents are associated with this document and should be used as a reference.
- WFM Coalition Reference Model
- WFM Coalition Glossary
- WFM Coalition WAPI Naming Conventions

3.7 Conformance
A vendor can not claim conformance to this or any other WfMC specification unless specifically authorized to make that claim by the WfMC. The WfMC grants this permission only upon the verification of the particular vendor’s implementation of the published specification, according to the conformance requirements and applicable test procedures defined by the WfMC.

3.8 WAPI Naming Conventions
The Working group has proposed a set of standards for handling the naming conventions of the different implementation of the Workflow API. These naming conventions standards are described in the document.
Workflow Management Coalition Interface 2 WAPI Naming Conventions (Document Number WFMC-TC-1013).
4. WAPI Data Types

This section describes the WAPI data types. These data types are used in the WAPI calls as input and output parameters.

4.1 Basic WAPI Data Types

This subsection contains definitions of the basic Workflow Management types that are operating system or platform dependent.

```c
typedef char WMTInt8;
typedef short WMTInt16;
typedef long WMTInt32;
typedef unsigned char WMTUInt8;
typedef unsigned short WMTUInt16;
typedef unsigned long WMTUInt32;
typedef WMTInt8 WMTText;
typedef WMTText *WMTPText;
typedef WMTInt8 *WMTPInt8;
typedef WMTInt16 *WMTPInt16;
typedef WMTInt32 *WMTPInt32;
```

```c
typedef WMTInt8 WMTBoolean;
typedef WMTUInt8 *WMTPointer;
typedef WMTText *WMTPPrivate;
#define WMNULL ((WMTPointer)0)
#define WMFalse 0
#define WMTrue (!WMFalse)
```

4.2 Other WAPI Data Types

This subsection contains definitions of the Workflow Management types that are specific to the structures and objects defined in this specification.

Strings in this specification, are assumed to be zero terminated. The maximum string length for names, keywords and identifiers in this specification is 63 characters hosted in a 64 byte text array. The following macro definition specifies this typical size:

```c
#define NAME_STRING_SIZE 64
```

All strings in this specification are defined as text arrays, such as:

```c
WMTText user_identification[NAME_STRING_SIZE];
```

Given this, in the example above the string can include up to a maximum of 63 real characters.

In some other cases, the fixed size structures for data reference and unique ids are also defined through the following macro definitions:

```c
#define UNIQUE_ID_SIZE 64
```
All WAPI function calls have a uniform error return datatype:

```c
typedef struct
{
    WMTInt16 main_code;
    WMTInt16 sub_code;
} WMTErrRetType;
```

This data type is shared among all API calls. All other data types are shown along with the WAPI description for each individual call.

This error return datatype is an Int32 word that has two Int16 elements for error returns. The main_code element contains the main error return code, while the sub_code element contains a code that further specifies the nature of the error. For example, the main_code error code `WM_INVALID_PROCESS_INSTANCE` (see Error Return Codes below), would include in its sub_code set of codes a further, more detailed reason why the process instance is invalid.

This specification assumes that the Coalition will specify a subset of the main_code codes, leaving for vendor specific implementation the remaining main_code codes and the set of sub_code codes to provide extensibility and specialization of error codes.

```c
typedef struct
{
    char user_identification[NAME_STRING_SIZE];
    // The identification of the workflow participant on whose behalf the Workflow Application will be operating. The value specified may represent a human, a device, etc. This identification is normally used for security checking, accounting, etc.
    char password[NAME_STRING_SIZE];
    char engine_name[NAME_STRING_SIZE];
    // The identification of the WFM Engine to whom the subsequent API calls are to be directed. This information would not be required for some WFM products in the normal case. However, it is required for those Workflow Applications which interact with multiple WFM Engines. This would be a symbolic name which is resolved through a lookup facility.
    char scope[NAME_STRING_SIZE];
    // Identification of scope for the application. If scope is not relevant, then this field would be empty and ignored.
} WMTConnectInfo;
```

```c
typedef WMTConnectInfo *WMTPConnectInfo;
```

```c
typedef struct
{
    WMTUInt32 session_id; // locally unique ID for the session
    WMTPPrivate pprivate; // pointer to a private structure containing vendor specific information.
} WMTSessionHandle;
```

```c
typedef WMTSessionHandle *WMTPSessionHandle;
```
typedef struct {
    WMTInt32 filter_type; // Includes basic types and SQL String
    WMTInt32 filter_length; // Length (in bytes) of value
    WMTText attribute_name [NAME_STRING_SIZE]
    WMTUInt32 comparison; // one of: <, >, =, !=, <=, <=
    WMTText filter_string;
} WMTFilter;

typedef WMTFilter *WMTPFilter;

// The first 255 filter types will be reserved. These will be used for filtering on
attributes of process control data and process relevant data. The specific code values
for these codes are included in the WFM Coalition Interface 2 WAPI Naming Conventions
specification document.

// In this specification there are two types of filters. One type is useful for
comparisons with and between attribute values. In this case, the filter_string
includes the attribute value that the attribute is compared against. The second type
is a more general mechanism in which the filter_string represents the whole argument
(typically a full SQL argument). If filter_type is a SQL string, the filter_string
will point to a SQL clause with the syntax of a WHERE clause in the SQL 92 standard
language specification.

typedef struct {
    WMTUInt32 query_handle;
} WMTQueryHandle;

typedef WMTQueryHandle *WMTPQueryHandle;

typedef struct {
    WMTText wf_participant[NAME_STRING_SIZE];
} WMTWflParticipant;

typedef WMTWflParticipant *WMTPWflParticipant;

typedef struct {
    WMTText proc_def_id[UNIQUE_ID_SIZE];
} WMTProcDefID;

typedef WMTProcDefID *WMTPProcDefID;

typedef struct {
    WMTText activity_id[NAME_STRING_SIZE];
} WMTActivityID;

typedef WMTActivityID *WMTPActivityID;

typedef struct {
    WMTText proc_def_state[NAME_STRING_SIZE];
} WMTProcDefState;

typedef WMTProcDefState *WMTPProcDefState; // pointer to a 63-byte string

typedef struct {
    // This is the minimum list of elements at this time. Future versions to provide
    // extensibility for this structure.
    WMTText process_name[NAME_STRING_SIZE];
    WMTPProcDefID proc_def_id;
    WMTPProcDefState state;
} WMTProcDef;
typedef struct
{
    WMTText proc_inst_id[UNIQUE_ID_SIZE];
} WMTProcInstID;
typedef WMTProcInstID *WMTPProcInstID;

typedef struct
{
    WMTText proc_inst_state[NAME_STRING_SIZE];
} WMTProcInstState;
typedef WMTProcInstState *WMTPProcInstState; // pointer to a 63-byte string

typedef struct
{
    // This is the minimum list of elements at this time. Future versions to provide
    // extensibility for this structure.
    WMTText process_name[NAME_STRING_SIZE];
    WMTProcInstID proc_inst_id;
    WMTProcDefID proc_def_id;
    WMTProcInstState state;
    WM TInt32 priority;
    WMTWflParticipant proc_participants[20];
    // up to 20 63 character long participant identifiers
} WMTProcInst;
typedef WMTProcInst *WMTPProcInst;

typedef struct
{
    WMTText activity_inst_id[UNIQUE_ID_SIZE];
} WMTActivityInstID;
typedef WMTActivityInstID *WMTPActivityInstID;

typedef struct
{
    WMTText activity_inst_state[NAME_STRING_SIZE];
} WMTActivityInstState;
typedef WMTActivityInstState *WMTPActivityInstState;

typedef struct
{
    // This is the minimum list of elements at this time. Future versions to provide
    // extensibility for this structure.
    WMTText activity_name[NAME_STRING_SIZE];
    WMTActivityInstID activity_inst_id;
    WMTProcInstID proc_inst_id;
    WMTActivityInstState state;
    WM TInt32 priority;
    WMTWflParticipant activity_participants[10];
    // up to 10 63 character long participant identifiers
} WMTActivityInst;
typedef WMTActivityInst *WMTPActivityInst;
typedef struct
{
    WMTText work_item_id[UNIQUE_ID_SIZE];
} WMTWorkItemID;

typedef WMTWorkItemID *WMTPWorkItemID;

typedef struct
{
    // This is the minimum list of elements at this time. Future versions to provide extensibility for this structure.
    WMTText workitem_name[NAME_STRING_SIZE];
    WMTWorkItemID workitem_id;
    WMTActivityInstID activity_inst_id;
    WMTProcInstID proc_inst_id;
    WMTInt32 priority;
    WMTWF1Participant participant;
} WMTWorkItem;

typedef WMTWorkItem *WMTPWorkItem;

typedef struct
{
    WMTText attribute_name[NAME_STRING_SIZE];
    WMTInt32 attribute_type; // type of the attribute
    WMTInt32 attribute_length; // length of the attribute value
    WMTPText pattribute_value; // pointer to the attribute value
} WMTAttribute;

typedef WMTAttribute *WMTPAttribute;

typedef struct
{
    WMTInt32 attribute_number;
    WMTPAttribute pattribute;
    WMTPNextAttr *WMTPAttributeList
} WMTAttributeList;

typedef WMTAttributeList *WMTPAttributeList;

4.3 Attributes

This specification does not make any assumption about the binding that workflow applications will make of retrieved attributes and their values. It is up to the specific application to manage this binding. The API manages attributes as a set of four elements:

WMTText attribute_name[NAME_STRING_SIZE];
WMTInt32 attribute_type; // type of the attribute
WMTInt32 attribute_length; // length of the attribute value
WMTPText pattribute_value; // pointer to the attribute value

All API calls in this specification that deal with attributes, take each individual element as separate parameter for the call.

The following type definitions are used for attribute name:

typedef WMTText WMTPAttribName[NAME_STRING_SIZE];
typedef WMTPAttribName *WMTPAttribName;

These attributes are of the kind called Process Control and Process Relevant Data. Some attributes of process instances, activity instances and work items could be: priority, state, start_time, description, instance_name, workflow_participant.
5. WAPI Error Return Codes

This section describes the minimal set of WAPI error return codes. These error codes correspond to the main_code element of the WMTErrRetType datatype defined above. The specific code values for these codes are included in the WFM Coalition WAPI Naming Conventions specification document.

The minimal set of main_code error return codes are:

- **WM_SUCCESS**: Indicates that the API call completed successfully.
- **WM_CONNECT_FAILED**: Indicates that the WMConnect call failed.
- **WM_INVALID_PROCESS_DEFINITION**: Indicates that the process definition ID that was passed as parameter to an API call was not valid, or it was not recognized by the servicing workflow engine.
- **WM_INVALID_ACTIVITY_NAME**: Indicates that the activity name that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_PROCESS_INSTANCE**: Indicates that the process instance ID that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_ACTIVITY_INSTANCE**: Indicates that the process instance ID that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_WORKITEM**: Indicates that the work item ID that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_ATTRIBUTE**: Indicates that the attribute that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_ATTRIBUTE_ASSIGNMENT_FAILED**: Indicates that the workflow engine was not able to complete the attribute assignment requested.
- **WM_INVALID_STATE**: Indicates that a state was not valid, or was not recognized by the servicing workflow engine.
- **WM_TRANSITION_NOT_ALLOWED**: Indicates that the state transition requested was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_SESSION_HANDLE**: Indicates that the session ID that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_QUERY_HANDLE**: Indicates that the query handle ID that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.
- **WM_INVALID_SOURCE_USER**
Indicates that the participant “source user” that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.

**WM_INVALID_TARGET_USER**
Indicates that the participant “target user” that was passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.

**WM_INVALID_FILTER**
Indicates that the filter structure or values that were passed as parameter to an API call was not valid, or was not recognized by the servicing workflow engine.

**WM_LOCKED**
Reserved for situations in which the servicing workflow engine implements “locking” of workflow entities (process definitions, process instances, activities, work items, etc.) to indicate that the entity is locked at the moment in which its access is requested.

**WM_NOT_LOCKED**
Reserved for situations in which the servicing workflow engine implements “locking” of workflow entities (process definitions, process instances, activities, work items, etc.) to indicate that the entity is not locked at the moment in which its access is requested.

**WM_NO_MORE_DATA**
Indicates that a *fetch* query call has reached the end of the list of valid entities to be returned. This error return code is used to implement queries of lists of workflow entities, it indicates that all the entities of the list that matched the selection criterion have already been returned.

**WM_INSUFFICIENT_BUFFER_SIZE**
Indicates that the buffer size that was passed to an API call is insufficient to hold the data that it is supposed to receive.

**WM_APPLICATION_BUSY**
Indicates that the corresponding application is currently busy and cannot return a status of work progress.

**WM_INVALID_APPLICATION**
Indicates that an invalid application has been requested by the calling interface.

**WM_INVALID_WORK_ITEM**
Indicates that an invalid work item has been referenced to by the calling interface.

**WM_APPLICATION_NOT_STARTED**
Indicates that the requested application did not start up successfully.

**WM_APPLICATION_NOT_DEFINED**
Indicates that the application is not installed or configured.

**WM_APPLICATION_NOT_STOPPED**
Indicates that the corresponding application did not stop orderly.
6. WAPI Descriptions

This section describes the WAPI calls. They are grouped as follows:

- WAPI Connection Functions
- WAPI Process Control Functions
- WAPI Activity Control Functions
- WAPI Process Status Functions
- WAPI Activity Status Functions
- WAPI Worklist Functions
- WAPI Administration Functions
- WAPI Application Invocation Functions

The specification of the WAPI calls that follows includes a specification of parameters with indications of the direction of data passing:

- **in** for parameters with data being passed to the API from the calling application
- **out** for parameters with data being passed from the API to the calling application.

It should be noted, that in the “C” language interface, parameters that are specified as **out** require a pointer to be passed from the calling application to the API. The API in turn will return the appropriate data in the space pointed to by the pointer. The specification of these **in** and **out** parameters is provided to clarify the specific purpose of these parameters in the calls.

6.1 WAPI Connection Functions

Connected/Connectedless Overview

The Coalition WMConnect /WMDisconnect API commands are intended to bound a set of related work by the application using them. When issued, the WMConnect returns a handle whose value is used on all other Coalition API calls. The handle value is unique and relates API calls which are issued between a WMConnect /WMDisconnect pair instance. The WMConnect command allows information to be supplied once and to remain valid until a WMDisconnect occurs.

Information supplied during the WMConnect (see the ConnectInfo structure in the WMConnect call) includes identification information relating to who/what is requesting services from the WFM Engine for use by an authentication service. The structure of the session handle that is returned by the WMConnect call is a pointer to a structure that contains a session ID and another structure pointer containing vendor specific information. (See the Session Handle structure in the WMConnect call.)

For those workflow servers that establish a connection, the session ID and the pointer to the vendor specific information would be returned by the workflow engine. For those workflow servers that do not establish a connection, the session ID would be set to 0, and a pointer to the connection information that was passed in by the user will be stored in the private structure contained in the session handle structure.

Operation between the API and the Engine

The construction of the Coalition API calls are intended to have little impact on the operational structure of how a WFM product supports them. The API calls are considered to be protocol neutral in that once the API boundary is crossed, different types of mechanisms may be employed to deliver the request to the WFM engine. A particular WFM product's method of interacting between the API calls and the WFM Engine functions may be RPC, conversational, messaging (connectedless) or others.
If a messaging mechanism is used by a WFM product, the receipt of a WMConnect may result in the determination of what messaging queue is to be used for interaction between its API support and the WFM engine functions, plus establishing control information to link that queue to subsequent API calls which use a particular handle. If the WFM engine is remote, it may also send a setup type of message to the engine.

If a conversational mechanism is used by a WFM product, and the WFM engine is remote, the receipt of a WMConnect may result in the establishment of a communications session between the code supporting the API calls and the WFM engine.

If a data base is being used, one of the results of the WMConnect may be the establishment of a connection to the appropriate data store facility.

A particular WFM product may choose to accept the WMConnect command, return a handle, and ignore the fact that it occurred.

The above are examples of possible operations performed by different WFM products in support of a WMConnect command. Obviously, more are possible.

In some cases, a product will be required to connect a single workstation to multiple WFM engines. It is possible that multiple WMConnect commands are active concurrently and the subsequent API commands be directed to the correct WFM engine. The WMConnect command may be used to designate a particular engine. The handle returned from the WMConnect command may be used on subsequent API calls to link those which relate to a engine.

The results of a WMDisconnect command is may vary, again depending upon a particular WFM product implementation. Its purpose is to indicate that the application issuing the preceding API calls will no longer be accessing the WFM engine functions within the previous context. In some products, upon receipt of a WMDisconnect command, communications and other resource types may be released.

Application Operation when using the API calls

The operational structure of an application as it relates to the use of the Coalition API calls is affected by the way the API calls are constructed. The current construction of the Coalition API calls result in the code segment of the application making the API call to run in blocked mode. That is, the application will issue an API command and 'wait' for a response from what it perceives as the WFM engine. When making the API call, the application code segment gives up control to the API and does not regain control until the API command is satisfied.

Much of the time, the API commands will be issued due to a workflow participant's direction via the application's End User Interface (EUI). Most of the current API commands are not such that a workflow participant would be interested in making the request, doing something else, and then sometime later (via a process/queue/whatever) viewing the real response to the request. With the request types supported by the API set, it would normally be the case that a workflow participant would want to see the response to the request as soon as possible.

The API calls could be constructed in such a way to allow the application code segment making the API call to run in unblocked mode. That is, to make the API call 'immediate return' rather than waiting for the actual response to the requested action. If this were done, the Coalition would need to define additional functions to support connectedless mode of operation (in some manner, get the asynchronous response when it did arrive and get it to the workflow participant).

The WMConnect / WMDisconnect API commands themselves have nothing to do with the ability of an application to run connected or connectedless as they are now defined.
Synchronous vs Asynchronous Calls

Most API calls in the WAPI call set are synchronous calls. In particular all the query related API calls are synchronous. Other calls may have some asynchronous behavior in that the call itself will return synchronously to the caller program, but the work specified by the call may be executed by the Workflow Engine at a later time, letting the application proceed. This set of API calls will not include any Call-Back mechanism to synchronize asynchronous calls.
6.1.1 WMConnect

NAME

WMConnect - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMConnect command informs the WFM Engine that other commands will be originating from this source.

```c
WMTErrRetType WMConnect (in WMTPConnectInfo pconnect_info, out WMTPSessionHandle psession_handle)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pconnect_info</td>
<td>Pointer to structure containing the information required to create a connection.</td>
</tr>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information which can be passed to the WFM Engine on all subsequent API calls which would identify interactions within the WMConnect / WMDisconnect bounds, that define a participant’s session interaction with the Engine. These handles are opaque so that in connectedless environments the handles include participants identities and passwords rather than session identification. There will be a special value for a handle to indicate failure of the function.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

```c
WM_SUCCESS  
WM_CONNECT_FAILED
```

WMDisconnect

NAME

WMDisconnect - Disconnect from the WFM Engine for this series of interactions

DESCRIPTION

The WMDisconnect command tells the WFM Engine that no more API calls will be issued from this source using the named handle. The WFM Engine could discard state data being held or take other closure actions.

```c
WMTErrRetType WMDisconnect (in WMTPSessionHandle psession_handle)
```

ERROR RETURN VALUE

```c
WM_SUCCESS  
WM_INVALID_SESSION_HANDLE
```
6.2 WAPI Process Control Functions

Process Control Functions can be defined as those which change the operational state of one or more process instances. These API calls are intended for use by the WFM end user application. However, some of the API calls, or parameters within some of the API calls, may affect multiple users and would normally be restricted to the use of a process administrator.

6.2.1 WMOpenProcessDefinitionsList

NAME

WMOpenProcessDefinitionsList - Specifies and opens the query to produce a list of all process definitions that meet the selection criterion of the filter.

DESCRIPTION

This command may also be used by a manager or process administrator to get a list of process definitions so they may view which processes are startable by particular persons. This command directs the WFM Engine to open the query to provide a list of process definitions which are available to a particular workflow participant, some of which may be startable by the participant. It is assumed that not all processes in an organization may be started by all workflow participants. One of the uses of this API is to allow a workflow participant to view which processes he/she can start with the expectation that the next action by the workflow participant would be to pick one to be started.

This command will return a query handle for a list of process definitions that match the specified value for the attribute. The command will also return, optionally, the total count of definitions available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1. If pproc_def_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL process definitions.

(Note: This API does not change the state of process or activity instances per the definition above of Process Control Functions. It is included in this section because it might normally lead to the execution of other API calls which would cause operational state changes.)

WMTErrRetType WMOpenProcessDefinitionsList (  
in WMTPSessionHandle psession_handle,  
in WMTPFilter pproc_def_filter,  
in WMTPBoolean count_flag,  
out WMTPQueryHandle pquery_handle,  
out WMTPInt32 pcount)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_def_filter | Filter associated with the process definition.
count_flag | Boolean flag that indicates if the total count of definitions should be returned.
pquery_handle | Pointer to a structure containing a unique query information.
pcount | Total number of process definitions that fulfill the filter condition.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_FILTER

Version 2.0  
Copyright © 1993, 1999, The Workflow Management Coalition
**REQUIREMENTS**

No requirements are assumed to exist with regard to the type of process model.

No requirements are assumed to exist with regard to how workflow participant’s are identified within the WFM Engine.

**RATIONALE FOR API**

This command and the corresponding fetch calls allows a workflow participant to retrieve the process definition ids which a workflow participant is authorized to start. They might be used in conjunction with the WMCreateProcessInstance and WMStartProcess API calls to start a particular named process.
6.2.2 WMFetchProcessDefinition

NAME

WMFetchProcessDefinition - Returns the next process definition from the set of process definitions that met the selection criterion stated in the WMOpenProcessDefinitionsList call.

DESCRIPTION

This command directs the WFM Engine to provide one process definition from the list of process definitions which are available to a particular workflow participant, some of which may be startable by the participant. It is assumed that not all processes in an organization may be started by all workflow participants. One of the uses of this API is to allow a workflow participant to view which processes he/she can start with the expectation that the next action by the workflow participant would be to pick one to be started. This fetch function, as well as all other fetch functions in this API, will return subsequent items after every call, one at a time. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The sort order in which the items are returned is specific of the workflow engine servicing the call, no specific order should be assumed.

WMTErrRetType WMFetchProcessDefinition (in WMTPSessionHandle psession_handle, in WMTPQueryHandle pquery_handle, out WMTPProcDef pproc_def_buf_ptr)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle Identification of the specific query handle returned by the WMOpenProcessDefinitionsList query command.
pproc_def_buf_ptr Pointer to a buffer area provided by the client application where the process definition structure will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_DEFINITION
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.2.3 WMCloseProcessDefinitionsList

NAME

WMCloseProcessDefinitionsList - Closes the query of process definitions.

DESCRIPTION

WMTErrRetType WMCloseProcessDefinitionsList(
    in WMTPSessionHandle psession_handle,
    In WMTPQueryHandle pquery_handle)

Argument Name        Description
psession_handle      Pointer to a structure containing information about the context for this action.
pquery_handle        Identification of the specific query handle returned by the WMOpenProcessDefinitionsList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.2.4 WMOpenProcessDefinitionStatesList

NAME

WMOpenProcessDefinitionStatesList - Specifies and opens the query to produce the list of states of the process definition that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of states for a process definition. The command will also return, optionally, the total count of definitions available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available states of the process definition that match the filter criterion, in order to offer this list to the application user. For example, process definitions can be in states such as disabled (thus disallowing temporarily the creation of new process definitions), or enabled (thus allowing again the creation of new process definitions based on the named definition). If pproc_def_state_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL states available for the definition.

WMTErrRetType WMOpenProcessDefinitionStatesList (in WMTPSessionHandle psession_handle,
  in WMTPProcDefID pproc_def_id,
  in WMTPFilter pproc_def_state_filter,
  in WMTBoolean count_flag,
  out WMTPQueryHandle pquery_handle,
  out WMTUInt32 pcount)

Argument Name                  Description
psession_handle                Pointer to a structure containing information about the context for this action.
pproc_def_id                   Pointer to a structure containing the unique process definition ID.
pproc_def_state_filter         Filter associated with the process definition state.
count_flag                     Boolean flag that indicates if the total count of process definition states should be returned.
pquery_handle                 Pointer to a structure containing a unique query information.
pcount                         Total number of states for this process definition.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_DEFINITION
6.2.5 WMFetchProcessDefinitionState

NAME

WMFetchProcessDefinitionState - Returns the next process definition state, from the list of states of the process definition that match the filter criterion.

DESCRIPTION

This command returns a process definition state. This fetch function will return subsequent process definition states after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchProcessDefinitionState (  
  in  WMTPSessionHandle psession_handle,  
  in  WMTPQueryHandle pquery_handle,  
  out WMTPProcDefState pproc_def_state)

Argument Name   Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle  Identification of the specific query handle returned by the WMOpenProcessDefinitionStatesList query command.
pproc_def_state Pointer to a buffer area provided by the client application where the state name will be placed.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_QUERY_HANDLE  
WM_NO_MORE_DATA
6.2.6 WMCloseProcessDefinitionStatesList

NAME

WMCloseProcessDefinitionStatesList - Closes the query for process definition states.

DESCRIPTION

WMTErrRetType WMCloseProcessDefinitionStatesList ( 
   In WMTPSessionHandle psession_handle, 
   In WMTPQueryHandle pquery_handle) 

Argument Name     Description
psession_handle   Pointer to a structure containing information about the context for this action.
pquery_handle     Identification of the specific query handle returned by the 
                  WMOpenProcessDefinitionStatesList query command.

ERROR RETURN VALUE

WM_SUCCESS 
WM_INVALID_SESSION_HANDLE 
WM_INVALID_QUERY_HANDLE
### 6.2.7 WMChangeProcessDefinitionState

**NAME**

**WMChangeProcessDefinitionState** - Changes the state of the named process definition.

**DESCRIPTION**

This command is defined to allow a process definition to be changed temporarily to a specific state such as *disabled* (thus disallowing temporarily the creation of new process definitions), or *enabled* (thus allowing again the creation of new process definitions based on the named definition).

```c
WMTErrRetType WMChangeProcessDefinitionState (  
    in WMTPSessionHandle psession_handle,  
    in WMTPProcDefID pproc_def_id,  
    in WMTPProcDefState pproc_def_state)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this</td>
</tr>
<tr>
<td></td>
<td>action.</td>
</tr>
<tr>
<td>pproc_def_id</td>
<td>Pointer to a structure containing a unique process definition ID.</td>
</tr>
<tr>
<td>pproc_def_state</td>
<td>Pointer to a structure that contains the name of the state to change the</td>
</tr>
<tr>
<td></td>
<td>process definition to.</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_DEFINITION
- WM_INVALID_STATE
- WM_TRANSITION_NOT_ALLOWED

**REQUIREMENTS**

Each process definition must have a unique ID within an administrative scope.

**RATIONALE FOR API**

This API allows the possible intervention of a process administrator in a running process. This might be for the purpose of changing the process definition and having all subsequently created definitions reflect the new definition.

**AUDIT INFORMATION**

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

- **Audit Data Type:** Change Process Definition State
- **Event Code:** WMChangedProcessDefinitionState
6.2.8 WMCreateProcessInstance

NAME

WMCreateProcessInstance - Create an instance of a previously defined process.

DESCRIPTION

An operational instance of the named process definition will be created by a WFM Engine as the result of this command. A call to WMStartProcess would then start the process.

To assign attributes to the process instance, you will make multiple calls to WMAssignProcessInstanceAttribute.

The process instance ID returned by this call is valid and reliable until WMStartProcess is called, at which time it may be reassigned to a new value.

```c
WMTErrRetType WMCreateProcessInstance(  
    in  WMTPSessionHandle psession_handle,  
    in  WMTPProcDefID  pproc_def_id,  
    in  WMTPText  pproc_inst_name,  
    out WMTPProcInstID  pproc_inst_id)
```

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_def_id | Pointer to a structure containing a unique process definition ID.
pproc_inst_name | Pointer to the name for the process instance created by this call.
pproc_inst_id | Pointer to a structure containing the process instance ID created by this call.

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_DEFINITION

REQUIREMENTS

No requirements exist with regard to process model type.

RATIONALE FOR API

This API allows a workflow participant to create an instance of a process. It is anticipated that vendor’s implementations will be of at least 2 types: one in which the creation of a process instance and the starting of the same are a single functionality and another in which this functionality is separate. The calls in this API definition are thus separated to accommodate both types of implementation. Vendors that provide the single functionality will implement the creation and start of a process through the creation of a temporary (possibly local) pproc_inst_id through WMCreateProcessInstance, assign attributes to it and then call WMStartProcess.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

- Audit Data Type: Create / Start Process Instance
- Event Code: WMCreatedProcessInstance
6.2.9 WMStartProcess

NAME
WMStartProcess - Start the named process.

DESCRIPTION
The WMStartProcess command directs the WFM Engine to begin executing a process, for which an instance has been created. When a process is started through this command, the first activity(s) of the process will be started. The process instance ID returned by this call will be valid for the life of the process instance.

Note: The programmer needs to maintain the association between the new process instance ID and the session in order to identify which session they need to connect to for future calls.

WMTErrRetType WMStartProcess {
   in WMTPSessionHandle psession_handle,
   in WMTPProcInstID pproc_inst_id,
   out WMTPProcInstID pnew_proc_inst_id)

Argument Name         Description
psession_handle       Pointer to a structure containing information about the context for this action.
pproc_inst_id         Pointer to a structure containing the process instance ID returned by the WMCreateProcessInstance call.
pnew_proc_inst_id     Pointer to a structure containing the process instance ID created by this call. This ID will be valid for the life of the process instance.

ERROR RETURN VALUE
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ATTRIBUTE

REQUIREMENTS
The process instance to be started has a unique id within an administrative scope.
No requirements exist with regard to process model type.

RATIONALE FOR API
This API allows a workflow participant to start a created process instance. It is anticipated that vendor’s implementations will be of at least 2 types: one in which the creation of a process instance and the starting of the same are a single functionality and another in which this functionality is separate. The calls in this API definition are thus separated to accommodate both types of implementation. Vendors that provide the single functionality will implement the creation and start of a process through the creation of a temporary (possibly local) proc_inst_id through WMCreateProcessInstance, assign attributes to it and then call WMStartProcess.

AUDIT INFORMATION
The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Create / Start Process Instance
Event Code: WMStartedProcessInstance
6.2.10 WMTerminateProcessInstance

NAME
WMTerminateProcessInstance - Terminate a process instance.

DESCRIPTION
This command provides the capability of gracefully terminating a process without aborting the process instance. Return from this call does not imply that the process instance has terminated, for example, the process instance could be stopped when currently running activities are complete. The exact behavior of currently running activities is system dependent.

WMErrRetType WMTerminateProcessInstance ( in WMTPSessionHandle psession_handle, in WMTPProcInstID pproc_inst_id)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pproc_inst_id A pointer to a structure that indicates the process instance that you want to terminate.

ERROR RETURN VALUE
The error return value for this function will include one or more of the following error codes (see Error Return Codes section):

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE

REQUIREMENTS
None

RATIONALE FOR API
To allow a process instances to be terminated.

AUDIT INFORMATION
The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Process Instance State
Event Code: WMTerminatedProcessInstance
6.2.11 WMOpenProcessInstanceStatesList

NAME

WMOpenProcessInstanceStatesList - Specifies and opens the query to produce the list of states of the process instance that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of states for a process instance. The command will also return, optionally, the total count of states available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1. The meaning of states is dependent upon the particular WFM Engine implementation. For example, the process instance can have states such as suspended or in-progress.

One of the uses of this API, together with the corresponding fetch and close calls, is to allow a workflow application to query the Workflow Engine for the available states of the process instance that match the filter criterion, in order to offer this list to the application user. If pproc_inst_state_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL states available for the process instance.

```c
WMTErrRetType WMOpenProcessInstanceStatesList (
    in WMTPSessionHandle psession_handle,
    in WMTPProcInstID pproc_inst_id,
    in WMTPFilter pproc_inst_state_filter,
    in WMTBoolean count_flag,
    out WMTPQueryHandle pquery_handle,
    out WMTPInt32 pcount)
```

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the unique process instance ID.
pproc_inst_state_filter | Filter associated with the process instance state.
count_flag | Boolean flag that indicates if the total count of process instance states should be returned.
pquery_handle | Pointer to a structure containing a unique query information.
pcount | Total number of states for this process instance.

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_INSTANCE
6.2.12 WMFetchProcessInstanceState

NAME

WMFetchProcessInstanceState - Returns the next process instance state from the list of states of the process instance that match the filter criterion.

DESCRIPTION

This command returns a process instance state. This fetch function will return subsequent process instance states after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchProcessInstanceState (in WMTPSessionHandle psession_handle, in WMTPQueryHandle pquery_handle, out WMTPProcInstState pproc_inst_state)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pquery_handle | Identification of the specific query handle returned by the WMOpenProcessInstanceStatesList query command.
pproc_inst_state | Pointer to a buffer area provided by the client application where the state name will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
### 6.2.13 WMCloseProcessInstanceStatesList

**NAME**
WMCloseProcessInstanceStatesList - Closes the query for process instance states.

**DESCRIPTION**

```c
WMErrRetType WMCloseProcessInstanceStatesList (  
    in WMTPSessionHandle psession_handle,  
    in WMTPQueryHandle pquery_handle)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pquery_handle</td>
<td>Identification of the specific query handle returned by the WMOpenProcessInstanceStatesList query command.</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_QUERY_HANDLE
6.2.14 WMChangeProcessInstanceState

NAME

WMChangeProcessInstanceState - Changes the state of the named process instance.

DESCRIPTION

This command is defined to allow a process instance to be changed temporarily to a specific state such as suspended.

Execution of this command will cause the single process instance that is named to be transitioned to a new state. In this case, the meaning of all states is dependent upon the particular WFM Engine implementation. This command will set the state attribute of the process instance to a state such as suspended or running.

```c
WMTErrRetType WMChangeProcessInstanceState(
    in WMTPSessionHandle psession_handle,
    in WMTPProcInstID pproc_inst_id,
    in WMTPProcInstState pproc_inst_state)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pproc_inst_id</td>
<td>Pointer to a structure containing a unique process instance ID.</td>
</tr>
<tr>
<td>pproc_inst_state</td>
<td>Pointer to a structure that contains the name of the process state that you want to change the instance to.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_INSTANCE
- WM_INVALID_STATE
- WM_TRANSITION_NOT_ALLOWED

REQUIREMENTS

Each process instance must have a unique ID within an administrative scope.

RATIONALE FOR API

This API allows the possible intervention of a workflow participant in a running process.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

- Audit Data Type: Change Process Instance State
- Event Code: WMChangedProcessInstanceState
6.2.15 WMOpenProcessInstanceAttributesList

NAME

WMOpenProcessInstanceAttributesList - Specifies and opens the query to produce the list of attributes that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of attributes for a process instance. The command will also return, optionally, the total count of attributes available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available attributes that can be assigned to the process instance, in order to offer this list to the application user. Attribute values can be obtained as well provided that a buffer of enough size is passed in the fetch call. Individual attribute values can also be retrieved with the WMGetProcessInstanceAttributeValue call. If pproc_inst_attr_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL attributes available for the process instance.

WMTErrRetType WMOpenProcessInstanceAttributesList ( 
  in WMTPSessionHandle psession_handle, 
  in WMTPProcInstID pproc_inst_id, 
  in WMTPFilter pproc_inst_attr_filter, 
  in WMITBoolean count_flag, 
  out WMTPQueryHandle pquery_handle, 
  out WMTPInt32 pcount)

Argument Name  Description
psession_handle  Pointer to a structure containing information about the context for this action.
pproc_inst_id  Pointer to a structure containing the unique process instance ID.
pproc_inst_attr_filter  Filter associated with the process instance attributes.
  Boolean flag that indicates if the total count of process instance attributes should be returned.
pquery_handle  Pointer to a structure containing a unique query information.
pcount  Total number of attributes for this process instance.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
6.2.16 WMFetchProcessInstanceAttribute

NAME

WMFetchProcessInstanceAttribute - Returns the next process instance attribute from the list of attributes that match the filter criterion.

DESCRIPTION

This command returns a process instance attribute. This fetch function will return subsequent process instance attributes after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The fetch function will return the attribute value as well in a buffer specified in the call. If buffer_size is NULL then the attribute value will not be returned. If buffer_size is not large enough to hold the attribute value then the function will return as much of the attribute value as can be fit in the buffer. The proper length of the attribute value is available in the attribute_length field. The application can compare the attribute_length with the buffer_size to determine if the full value was returned.

WMTErrRetType WMFetchProcessInstanceAttribute (  
in WMTPSessionHandle psession_handle,  
in WMTPQueryHandle pquery_handle,  
out WMTPAttrName pattribute_name,  
out WMTPInt32 pattribute_type,  
out WMTPInt32 pattribute_length,  
out WMTPText pattribute_value,  
in WMTInt32 buffer_size)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pquery_handle | Identification of the specific query handle returned by the WMOpenProcessInstanceAttributesList query command.
pattribute_name | Pointer to the name of the attribute.
pattribute_type | Pointer to the type of the attribute.
pattribute_length | Pointer to the length of the attribute value.
pattribute_value | Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size | Size of the buffer.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_QUERY_HANDLE  
WM_NO_MORE_DATA
6.2.17  WMCloseProcessInstanceAttributesList

NAME

WMCloseProcessInstanceAttributesList - Closes the query for process instance attributes.

DESCRIPTION

WMErrRetType WMCloseProcessInstanceAttributesList ( 
    In WMTPSessionHandle psession_handle,
    In WMTPQueryHandle pquery_handle)

Argument Name   Description
psession_handle  Pointer to a structure containing information about the context for this action.
pquery_handle   Identification of the specific query handle returned by the WMOpenProcessInstanceAttributesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.2.18 WMGetProcessInstanceAttributeValue

NAME

WMGetProcessInstanceAttributeValue - Returns the value, type and length of a process instance attribute specified by the proc_inst_id and attribute_name parameters.

DESCRIPTION

This command will return the value of a process instance attribute in the buffer specified in the call.

```c
WMTErrRetType WMGetProcessInstanceAttributeValue (  
    in WMTPSessionHandle psession_handle,  
    in WMTPProcInstID pproc_inst_id,  
    in WMTPAttrName pattribute_name,  
    out WMTPInt32 pattribute_type,  
    out WMTPInt32 pattribute_length,  
    out WMTPText pattribute_value,  
    in WMTInt32 buffer_size)  
```

Argument Name  | Description
---|---
psession_handle  | Pointer to a structure containing information about the context for this action.
pproc_inst_id  | Pointer to a structure containing the unique process instance ID.
attribute_name  | Pointer to the name of the attribute.
attribute_type  | Pointer to the type of the attribute.
attribute_length  | Pointer to the length of the attribute value.
attribute_value  | Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size  | Size of the buffer to be filled.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ATTRIBUTE
WM_INSUFFICIENT_BUFFER_SIZE
6.2.19 WMAssignProcessInstanceAttribute

NAME

WMAssignProcessInstanceAttribute - Assign the proper attribute to process instance(s)

DESCRIPTION

This command tells the WFM Engine to assign an attribute, change an attribute or to change the value of an attribute of a process instance.

This command changes the value of an attribute of a process instance. Attributes of process instances are of the kind called Process Control and Process Relevant Data. These attributes are specified as quadruplets of name, type, length and value.

WMTErrRetType WMAssignProcessInstanceAttribute ( 
  in WMTPSessionHandle psession_handle,
  in WMTPProcInstID pproc_inst_id,
  in WMTPAttrName pattribute_name,
  in WMTInt32 attribute_type,
  in WMTInt32 attribute_length,
  in WMTPText pattribute_value)

Argument Name | Description
---------------|--------------------------------------------------------
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the process instance ID that indicates the process for which the attribute will be assigned.
pattribute_name | Pointer to the name of the attribute.
attribute_type | Type of the attribute.
attribute_length | Length of the attribute value.
pattribute_value | Pointer to a buffer area provided by the client application where the attribute value will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_ASSIGNMENT_FAILED

REQUIREMENTS

None

RATIONALE FOR API

For various business reasons, certain pieces of work are required to be handled with particular attributes (e.g. priority) relative to other pieces of like work. This command allows attributes to be set on those pieces of work. In some cases, these attributes are determined by the WFM product based upon data values existing during process execution. The setting of these attributes through the use of this API is provided to cover the cases where applications set them upon requests from users.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:
Audit Data Type: Change Process Instance Attributes
Event Code: WMAssignProcessInstanceAttributes
6.3 WAPI Activity Control Functions

Activity Control Functions can be defined as those which change the operational state of one or more activity instances. These API calls are intended for use by the WFM end user. However, some of the API calls, or parameters within some of the API calls, may affect multiple users and would normally be restricted to the use of a process administrator.

6.3.1 WMOOpenActivityInstanceStatesList

NAME

WMOOpenActivityInstanceStatesList - Specifies and opens the query to produce the list of states of the activity instance that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of states for an activity instance. The command will also return, optionally, the total count of states available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available states of the activity instance that match the filter criterion, in order to offer this list to the application user. If pact_inst_state_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL states available for the activity instance.

WMErrMsgRetType WMOOpenActivityInstanceStatesList { in WMTPSessionHandle psession_handle, in WMTPProcInstID pproc_inst_id, in WMTPActivityInstID pactivity_inst_id, in WMTPFilter pact_inst_state_filter, in WMTPBoolean count_flag, out WMTPQueryHandle pquery_handle, out WMTPInt32 pcount }

Argument Name Description

psession_handle Pointer to a structure containing information about the context for this action.
pproc_inst_id Pointer to a structure containing a unique process instance ID.
pactivity_inst_id Pointer to a structure containing the unique activity instance ID.
pact_inst_state_filter Filter associated with the activity instance state.
count_flag Boolean flag that indicates if the total count of activity instance states should be returned.
pquery_handle Pointer to a structure containing a unique query information.
pcount Total number of states for this activity instance.

ERROR RETURN VALUE

WM_SUCCESS WM_INVALID_SESSION_HANDLE WM_INVALID_PROCESS_INSTANCE WM_INVALID_ACTIVITY_INSTANCE
6.3.2 WMFetchActivityInstanceState

NAME

WMFetchActivityInstanceState - Returns the next activity instance state, from the list of states of the activity instance that match the filter criterion.

DESCRIPTION

This command returns an activity state. This fetch function will return subsequent activity states after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchActivityInstanceState (in WMTPSessionHandle psession_handle,
                        in WMTPQueryHandle pquery_handle,
                        out WMTPActivityInstState pactivity_inst_state)

Argument Name | Description
---------------|--------------------------------------------------
psession_handle | Pointer to a structure containing information about the context for this action.
pquery_handle | Identification of the specific query handle returned by the WMOpenActivityInstanceStatesList query command.
pactivity_inst_state | Pointer to a buffer area provided by the client application where the state name will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.3.3 WMCloseActivityInstanceStatesList

NAME

WMCloseActivityInstanceStatesList - Closes the query for activity instance states.

DESCRIPTION

WMTErrRetType WMCloseActivityInstanceStatesList (in WMTPSessionHandle psession_handle, in WMTPQueryHandle pquery_handle)

Argument Name | Description
--- | ---
psessionId | Pointer to a structure containing information about the context for this action.
pQueryHandle | Identification of the specific query handle returned by the WMOpenActivityInstanceStatesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.3.4 WMChangeActivityInstanceState

NAME

WMChangeActivityInstanceState - Changes the state of the named activity instance.

DESCRIPTION

This command directs a WFM Engine to change the state of a single activity instance within a process instance. This allows the state of one activity instance to be changed, without impacting others in the process instance.

For example, this command will be used to change the state of an activity instance to suspended. This command can be used afterwards to change the state of the activity instance back to running. The implementation documentation will provide the names and semantics of the supported activity states for a particular implementation.

WMTErrRetType WMChangeActivityInstanceState ( 
    in WMTPSessionHandle psession_handle, 
    in WMTPProcInstID pproc_inst_id, 
    in WMTPActivityInstID pactivity_inst_id, 
    in WMTPActivityInstState pactivity_inst_state)

Argument Name          Description
psession_handle        Pointer to a structure containing information about the context for this action.
pproc_inst_id          Pointer to a structure containing a unique process instance ID.
pactivity_inst_id      Pointer to structure containing the activity instance ID of the activity whose state to change.
pactivity_inst_state   Pointer to a structure that contains the name of the activity instance state that you want to change to.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_STATE
WM_TRANSITION_NOT_ALLOWED

REQUIREMENTS

Each process instance must have a unique ID within an administrative scope.
Each activity instance must have a unique ID within a process instance.

RATIONALE FOR API

A workflow participant may wish to modify the state attributes associated with a particular activity instance.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Activity Instance State
Event Code: WMChangedActivityInstanceState
6.3.5 WMOpenActivityInstanceAttributesList

NAME

WMOpenActivityInstanceAttributesList - Specifies and opens the query to produce the list of activity attributes that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of attributes for an activity instance. The command will also return, optionally, the total count of attributes available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available attributes that can be assigned to the activity instance, in order to offer this list to the application user. Attribute values can be obtained as well provided that a buffer of enough size is passed in the fetch call. Individual attribute values can also be retrieved with the WMGetActivityInstanceAttributeValue call. If pact_inst_attr_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL attributes available for the activity instance.

WMTErrRetType WMOpenActivityInstanceAttributesList ( 
    in WMTPSessionHandle psession_handle, 
    in WMTPProcInstID pproc_inst_id, 
    in WMTPActivityInstID pactivity_inst_id, 
    in WMTPFilter pact_inst_attr_filter, 
    in WMTBoolean count_flag, 
    out WMTPQueryHandle pquery_handle, 
    out WMTPInt32 pcount)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pproc_inst_id Pointer to a structure containing the unique process instance ID.
pactivity_inst_id Pointer to a structure containing the unique activity instance ID.
pact_inst_attr_filter Filter associated with the activity instance attributes.
count_flag Boolean flag that indicates if the total count of activity instance attributes should be returned.
pquery_handle Pointer to a structure containing a unique query information.
pcount Total number of attributes for this activity instance.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
6.3.6 WMFetchActivityInstanceAttribute

NAME

WMFetchActivityInstanceAttribute - Returns the next activity instance attribute from the list of activity attributes that match the filter criterion.

DESCRIPTION

This command returns a activity instance attribute. This fetch function will return subsequent activity instance attributes after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The fetch function will return the attribute value as well in a buffer specified in the call. If buffer_size is NULL then the attribute value will not be returned. If buffer_size is not large enough to hold the attribute value then the function will return as much of the attribute value as can be fit in the buffer. The proper length of the attribute value is available in the attribute_length field. The application can compare the attribute_length with the buffer_size to determine if the full value was returned.

WMErrMsgRetType WMFetchActivityInstanceAttribute (
  WMTPSessionHandle psession_handle,
  WMTPQueryHandle pquery_handle,
  WMTPAttrName pattribute_name,
  WMTPInt32 pattribute_type,
  WMTPInt32 pattribute_length,
  WMTPText pattribute_value,
  WMInt32 buffer_size)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle Identification of the specific query handle returned by the WMOpenActivityInstanceAttributesList query command.
pattribute_name Pointer to the name of the attribute.
pattribute_type Pointer to the type of the attribute.
pattribute_length Pointer to the length of the attribute value.
pattribute_value Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size Size of the buffer.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.3.7 WMCloseActivityInstanceAttributesList

NAME

WMCloseActivityInstanceAttributesList - Closes the query for activity instance attributes.

DESCRIPTION

WMTErrRetType WMCloseActivityInstanceAttributesList ( 
    In WMTPSessionHandle psession_handle, 
    In WMTPQueryHandle pquery_handle)

Argument Name | Description
---------------|--------------------------------------------------
p/session_handle | Pointer to a structure containing information about the context for this action.
p/query_handle | Identification of the specific query handle returned by the

WMOpenActivityInstanceAttributesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.3.8 WMGetActivityInstanceAttributeValue

NAME

WMGetActivityInstanceAttributeValue - Returns the value, type and length of an activity instance attribute specified by the pproc_inst_id, pactivity_inst_id and attribute_name parameters.

DESCRIPTION

This command will return the value of an activity instance attribute in the buffer specified in the call.

WMTErrRetType WMGetActivityInstanceAttributeValue (  
    in WMTPSessionHandle psession_handle,  
    in WMTPProcInstID pproc_inst_id,  
    in WMTPActivityInstID pactivity_inst_id,  
    in WMTPAttrName pattribute_name,  
    out WMTPInt32 pattribute_type,  
    out WMTPInt32 pattribute_length,  
    out WMTPText pattribute_value,  
    in WM TInt32 buffer_size)

Argument Name | Description
---------------|--------------------------------------------------
p/session_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the unique process instance ID.
pactivity_inst_id | Pointer to a structure containing the unique activity instance ID.
pattribute_name | Pointer to the name of the attribute.
pattribute_type | Pointer to the type of the attribute.
pattribute_length | Pointer to the length of the attribute value.
pattribute_value | Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size | Size of the buffer to be filled.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ATTRIBUTE
WM_INSUFFICIENT_BUFFER_SIZE
6.3.9 WMAssignActivityInstanceAttribute

NAME

WMAssignActivityInstanceAttribute - Assign an attribute to an activity instance.

DESCRIPTION

This command tells the WFM Engine to assign an attribute, to change an attribute or to change the value of an attribute of the activity instance within a named process definition.

This command changes the value of the attributes of a activity instance. These attributes of activity instances are of the kind called Process Control and Process Relevant Data. These attributes are specified as quadruplets of name, type, length and value.

WMTErrRetType WMAssignActivityInstanceAttribute (  
  in WMTPSessionHandle psession_handle,  
  in WMTPProcDefID pproc_def_id,  
  in WMTPActivityInstID pactivity_inst_id,  
  in WMTPAttrName pattribute_name,  
  in WMTInt32 attribute_type,  
  in WMTInt32 attribute_length,  
  in WMTPText pattribute_value)

Argument Name Description  
psession_handle Pointer to a structure containing information about the context for this action.  
pproc_inst_id Pointer to a structure containing the unique process instance ID.  
pactivity_inst_id Pointer to a structure containing the activity instance identification for which the attribute will be assigned.  
pattribute_name Pointer to the name of the attribute.  
attribute_type Type of the attribute.  
attribute_length Length of the attribute value.  
pattribute_value Pointer to a buffer area provided by the client application where the attribute value will be placed.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_PROCESS_INSTANCE  
WM_INVALID_ACTIVITY_INSTANCE  
WM_INVALID_ATTRIBUTE  
WM_ATTRIBUTE_ASSIGNMENT_FAILED

REQUIREMENTS

None

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Assign Activity Instance Attributes  
Event Code: WMAssignedActivityInstanceAttributes
6.4 **WAPI Process Status Functions**

The process status functions are intended to provide a view of the work done, work to be done, work associated with a workflow participant or group of workflow participants, etc. The status queries may be requested by a normal workflow participant or may be requested by a manager or process administrator who wishes to view the progress of work within his/her domain.

The status API calls are structured such that they provide views ranging from a view of global work to a view of work within a single process instance. These views are as follows:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All the process instances associated with a process definition.</td>
</tr>
<tr>
<td>2</td>
<td>A view of a single process instance.</td>
</tr>
</tbody>
</table>

In addition, various filters (parameters) are provided with the calls such that the information returned may be tailored.

The API functions associated with these API calls are described in this section.
6.4.1 WMOpenProcessInstancesList

NAME

WMOpenProcessInstancesList - Specifies and opens the query to produce a list of process instances that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of process instances that match the specified value for the attribute. The command will also return, optionally, the total count of instances available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

This command will be used to set up a wide variety of queries of process instances. For example, this command will be used to set up the query for a list of completed or suspended process instances. If pproc_inst_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL accessible process instances.

WMTErrRetType WMOpenProcessInstancesList (  
in WMTPSessionHandle psession_handle,  
in WMTPFilter pproc_inst_filter,  
in WMTBoolean count_flag,  
out WMTPQueryHandle pquery_handle,  
out WMTPInt32 pcount)

Argument Name      Description
psession_handle    Pointer to a structure containing information about the context for this action.
pproc_inst_filter  Pointer to a structure containing the information for this request.
count_flag         Boolean flag that indicates if the total count of process instances should be returned.
pquery_handle      Pointer to a structure containing a unique query information.
pcount             Total number of process instances that fulfill the filter condition.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_FILTER

REQUIREMENTS

None

RATIONALE FOR API

The requester of the information needs to know what work of a particular type is in process or needs to know what work has completed.
6.4.2 WMFetchProcessInstance

NAME

WMFetchProcessInstance - Returns the next process instance from the list of process instances that met the selection criterion stated in the corresponding WMOpenProcessInstancesList call.

DESCRIPTION

This command returns a process instance. This fetch function will return subsequent process instances after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchProcessInstance (  
    in WMTPSessionHandle psession_handle,  
    in WMTPQueryHandle pquery_handle,  
    out WMTPProcInst pproc_inst_buf_ptr)

Argument Name  Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle Identification of the specific query handle returned by the WMOpenProcessInstancesList query command.
pproc_inst_buf_ptr Pointer to a buffer area provided by the client application where the set of process instances will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA

REQUIREMENTS

None
6.4.3 WMCloseProcessInstancesList

**NAME**

WMCloseProcessInstancesList - Closes the query of process instances.

**DESCRIPTION**

This command will close the query of process instances that match the specified query attribute, specified in the WMOpenProcessInstancesList command. The query handle can no longer be used.

```c
WMErrMsg WMCloseProcessInstancesList (  
in WMTPSessionHandle psession_handle,  
In WMTPQueryHandle pquery_handle)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pquery_handle</td>
<td>Identification of the specific query handle returned by the WMOpenProcessInstancesList query command.</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_QUERY_HANDLE
6.4.4 WMGetProcessInstance

NAME

WMGetProcessInstance - Return a specific process instance record.

DESCRIPTION

The WMGetProcessInstance provides information about what work has been done within a process instance and what is the current work being done within the process instance.

```c
WMErrRetType WMGetProcessInstance (
    in  WMTPSessionHandle psession_handle,
    in  WMTPProcInstID pproc_inst_id,
    out WMTPProcInst pproc_inst)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pproc_inst_id</td>
<td>Pointer to the process instance identification.</td>
</tr>
<tr>
<td>pproc_inst</td>
<td>Pointer to a structure containing the requested process instance information. Includes the state and other attributes of the process instance.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_INSTANCE

REQUIREMENTS

None
6.5 **WAPI Activity Status Functions**

The process status functions are intended to provide a view of the work done, work to be done, work associated with a workflow participant or group of workflow participants, etc. The status queries may be requested by a normal workflow participant or may be requested by a manager or process administrator who wishes to view the progress of work within his/her domain.

The status API calls are structured such that they provide views ranging from a view of global work to a view of work within a single activity instance. These views are as follows:

<table>
<thead>
<tr>
<th></th>
<th>All the activity instances associated to a process definition or instance</th>
<th>WM(Open+Fetch+Close)ActivityInstancesList</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A view of a single activity within a process instance.</td>
<td>WMGetActivityInstance</td>
</tr>
</tbody>
</table>

In addition, various filters (parameters) are provided with the calls such that the information returned may be tailored.

The API functions associated with these API calls are described in this section.
6.5.1 WMOpenActivityInstancesList

NAME

WMOpenActivityInstancesList - Specifies and opens the query to produce a list of activity instances that match the criterion of the filter.

DESCRIPTION

This command will return a query handle for a list of activity instances that match the criterion of the filter. The command will also return, optionally, the total count of activity instances available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

This command will be used to set up a wide variety of queries of activity instances. For example, this command will be used to set up the query for a list of completed or suspended activity instances. If pactivity_inst_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL accessible activity instances.

WMRetType WMOpenActivityInstancesList (in WMTPSessionHandle psession_handle, in WMTPFilter pactivity_inst_filter, in WMTPBoolean count_flag, out WMTPQueryHandle pquery_handle, out WMTPInt32 pcount)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pactivity_inst_filter | Pointer to a structure containing the information for this request.
count_flag | Boolean flag that indicates if the total count of activity instances should be returned.
pquery_handle | Pointer to a structure containing a unique query information returned by this function.
pcount | Total number of activity instances that fulfill the filter condition.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_FILTER

REQUIREMENTS

None

RATIONALE FOR API

The requester of the information needs to know what work of a particular type is in process or needs to know what work has completed.
6.5.2 WMFetchActivityInstance

NAME

WMFetchActivityInstance - Returns the next activity instance from the list of activity instances that met the selection criterion in the corresponding WMOpenActivityInstancesList call.

DESCRIPTION

This command returns an activity instance. This fetch function will return subsequent activity instances after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchActivityInstance (  
in WMTPSessionHandle psession_handle,  
lp WMTPQueryHandle pquery_handle,  
out WMTPActivityInst pactivity_inst)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle Identification of the specific query handle returned by the WMOpenActivityInstancesList query command.
pactivity_inst Pointer to a buffer area provided by the client application where the set of activity instances will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA

REQUIREMENTS

None
6.5.3 WMCloseActivityInstancesList

NAME
WMCloseActivityInstancesList - Closes the query of activity instances.

DESCRIPTION
This command will close the query of activity instances that match the specified query attribute, specified in the WMOpenActivityInstancesList command. The query handle can no longer be used.

WMTErrRetType WMCloseActivityInstancesList (in WMTPSessionHandle psession_handle, in WMTPQueryHandle pquery_handle)

Argument Name | Description
--- | ---
psessionId | Pointer to a structure containing information about the context for this action.
pqueryHandle | Identification of the specific query handle returned by the WMOpenActivityInstancesList query command.

ERROR RETURN VALUE
- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_QUERY_HANDLE

REQUIREMENTS
None
6.5.4 WMGetActivityInstance

NAME

WMGetActivityInstance - Returns the record of a specific activity instance.

DESCRIPTION

The WMGetActivityInstance command provides status about an activity within a process instance.

WMErrRetType WMGetActivityInstance ( 
  in  WMTPSessionHandle psession_handle, 
  in  WMTPProcInstID pproc_inst_id, 
  in  WMTPActivityInstID pactivity_inst_id, 
  out WMTPActivityInst pactivity_inst )

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the process instance identification.
pactivity_inst_id | Pointer to a structure containing the identification of the activity instance.
pactivity_inst | Pointer to a structure containing the activity instance information.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE

REQUIREMENTS

None
6.6 WAPI Worklist Functions

The WAPI worklist API calls provide workflow participants access to information about work to which they have been assigned. As described by the WFM Coalition reference model, a process consists of a set of activities connected in such a way to control the sequencing of application invocation. An activity is associated with one or more applications to be invoked and also, during run time, is associated with the person(s) who has been assigned to do the work. Depending upon a WFM product’s implementation, a workflow participant may be assigned one or more pieces of work at any one time. Each piece of work assigned to a workflow participant is called a ‘work item’ and the collection of all work items assigned to a workflow participant is called that workflow participant’s ‘worklist’.

(Note: To clarify the difference between an ‘activity’ and a ‘work item’ the following discussion is included. When a process is being defined (build time), an ‘activity’ is the construct used to define a piece of work to be done. It serves as a type of anchor point for further descriptions of that work to be done (i.e., the name of the application to be invoked, possibly a reference to skills needed to do the work, a symbolic name denoting the network address where the application is to be executed, etc.). During run time, when the activity is ready to be executed and one or more candidate persons are assigned to do the work, a work item is created and placed on that person(s) worklist. So, even though an activity and a work item both represent a piece of work, they come into existence at different points in time, there may be more than one work item for an activity and some operational characteristics may be different.)

A worklist then is defined as: the result of an implementation-defined query against the work item space. It is a list of work items and a work item is one element in a worklist.

The API calls in this section exist for the manipulation of work items and worklists.
6.6.1 WMOpenWorkList

NAME

WMOpenWorkList - Specifies and opens the query to produce the worklist that matches the criterion of the filter.

DESCRIPTION

This command provides the capability of returning a list of work items assigned to a specified workflow participant or a workgroup. The requester may be making the request on behalf of himself or may be a manager wanting to know what work has been assigned to a particular person or a workgroup.

A query handle will be returned for the list of work items that match the specified value for the attribute. The command will also return, optionally, the total count of work items available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1. If pworklist_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL accessible work items.

WMErrRetType WMOpenWorkList (
    in WMTPSessionHandle psession_handle,
    in WMTPFilter pworklist_filter,
    in WMTBoolean count_flag,
    out WMTPQueryHandle pquery_handle,
    out WMTPInt32 pcount)

Argument Name   Description
psession_handle Pointer to a structure containing information about the context for this action.
pworklist_filter Pointer to a structure containing the filter information for this request.
count_flag Boolean flag that indicates if the total count of work items should be returned.
pquery_handle Pointer to a structure containing a unique query information returned by this function.
pcount Total number of work items that fulfill the filter condition.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_FILTER

REQUIREMENTS

None

RATIONALE FOR API

A workflow participant must be able to determine what work has been assigned. A manager must be able to determine who has work and what work is to be done within a department.
6.6.2 WMFetchWorkItem

NAME

WMFetchWorkItem - Returns the next work item from the worklist that met the selection criterion in the corresponding WMOpenWorkList call.

DESCRIPTION

This command returns a work item. This fetch function will return subsequent work items after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchWorkItem (  
in  WMTPSessionHandle psession_handle,  
  in  WMTPQueryHandle pquery_handle,  
out  WMTPWorkItem pwork_item)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pquery_handle | Identification of the specific query handle returned by the WMOpenWorkList query command.
pwork_item | Pointer to a buffer area provided by the client application where the set of work item will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.6.3 WMCloseWorkList

NAME

WMCloseWorkList - Closes the query of work items.

DESCRIPTION

This command will close the query of work items that match the specified query filter, specified in the WMOpenWorkList command. The query handle can no longer be used.

WMTErrRetType WMCloseWorkList (  
          in WMTPSessionHandle psession_handle,  
          in WMTPQueryHandle pquery_handle)

Argument Name          Description
psession_handle        Pointer to a structure containing information about the context for this action.
pquery_handle          Identification of the specific query handle returned by the WMOpenWorkList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.6.4 WMGetWorkItem

NAME

WMGetWorkItem - Returns the record of a specific work item

DESCRIPTION

This command allows a workflow participant to designate which piece of work he wishes to do. The viewer may be selecting a work item from a list obtained by the WMOpenWorkList command.

This command operates on a single work item basis. This command execution need not imply that the work item is reserved or locked.

```c
WMTErrRetType WMGetWorkItem ( 
    in WMTPSessionHandle psession_handle,  
    in WMTPProcInstID pproc_inst_id,   
    in WMTPWorkItemID pwork_item_id,   
    out WMTPWorkItem pwork_item )
```

Argument Name          Description
---------------------   -----------------------------------------------
psession_handle        Pointer to a structure containing information about the context for this action.
pproc_inst_id          Pointer to a structure containing the unique process instance ID.
pwork_item_id          Pointer to a structure containing the work item identification for this request.
pwork_item             Pointer to a structure containing the work item being returned by this function.

ERROR RETURN VALUE

The error return value for this function will include one or more of the following error codes (see Error Return Codes section):

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_INSTANCE
- WM_INVALID_WORKITEM

REQUIREMENTS

The application issuing the command must have sufficient identification information to select the work item desired.

RATIONALE FOR API

A workflow participant must be able to tell the WFM Engine which piece of work is to be selected.

AUDIT INFORMATION

The following audit information is directly related to this function and might be recorded by an implementation of this specification that complies with the Audit Profile:

- Audit Data Type: Change Work Item State
- Event Code: WMSel ectedWorkItem

In this particular case it is left to the implementation to realize a state change of the Work Item when a WMGetWorkItem operation is invoked.
6.6.5 WMCompleteWorkItem

NAME

WMCompleteWorkItem - Tell the WFM Engine that this work item has been completed.

DESCRIPTION

This command allows a workflow participant to tell the WFM Engine that a work item has been completed.

To change a work item's attributes, multiple calls to WMAssignWorkItemAttribute.

```
WMTErrRetType WMCompleteWorkItem ( 
    in WMTPSessionHandle psession_handle, 
    in WMTPProcInstID pproc_inst_id, 
    in WMTPWorkItemID pwork_item_id)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pproc_inst_id</td>
<td>Pointer to a structure containing the unique process instance ID.</td>
</tr>
<tr>
<td>pwork_item_id</td>
<td>Pointer to a structure containing the work item identification for this request.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM

REQUIREMENTS

None

RATIONALE FOR API

WFM products implement various ways to determine when an activity is complete. The use of the API may range from just a successful/unsuccessful indication to placing values in the completion state which might cause the WFM Engine to select a future model navigation path from among many.

Typically, a work item will correspond to an activity instance. However the API should allow the existence of multiple work items per activity, executed one at a time. So completion of a work item does not necessarily mean that all work for an activity instance is completed. Completion of a work item could trigger the start of the next work item that corresponds to that activity instance. The Workflow Engine will determine the next work item based on the process definition.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Work Item State
Event Code: WMCompletedWorkItem

6.6.6 WMOpenWorkitemStatesList

NAME

WMOpenWorkItemStatesList - Specifies and opens the query to produce the list of states of workitem that match the filter criterion.
DESCRIPTION

This command will return a query handle for a list of states for a workitem. The command will also return, optionally, the total count of definitions available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available states of the workitem that match the filter criterion, in order to offer this list to the application user. For example, workitems can be in states such as disabled (thus disallowing temporarily the creation of new process definitions), or enabled (thus allowing again the creation of new process definitions based on the named definition). If pworkitem_state_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL states available for the definition.

WMTErrRetType WMOpenWorkitemStatesList ( 
    in WMTSessionHandle psession_handle,  
    in WMTPProcDefID pworkitem_id,  
    in WMTPFilter pworkitem_state_filter,  
    in WMITBoolean count_flag,  
    out WMTQueryHandle pquery_handle,  
    out WMTUInt32 pcount)

Argument Name | Description
--- | ---
p/session_handle | Pointer to a structure containing information about the context for this action.
p/workitem_id | Pointer to a structure containing the unique workitem ID.
p/workitem_state_filter | Filter associated with the workitem state.
count_flag | Boolean flag that indicates if the total count of process definition states should be returned.
p/query_handle | Pointer to a structure containing a unique query information.
pcount | Total number of states for this process definition.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_PROCESS_DEFINITION
6.6.7 WMFetchWorkitemState

NAME

WMFetchWorkitemState - Returns the next workitem state, from the list of states of the workitem that match
the filter criterion.

DESCRIPTION

This command returns a workitem state. This fetch function will return subsequent workitem states after
every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA.

WMTErrRetType WMFetchWorkitemState (  
    in WMTPSessionHandle psession_handle,  
    in WMTPQueryHandle pquery_handle,  
    out WMTPProcDefState pworkitem_state)

Argument Name  Description
psession_handle  Pointer to a structure containing information about the context for this action.
pquery_handle   Identification of the specific query handle returned by the
pworkitem_state   WMOpenWorkitemStatesList query command.
                  Pointer to a buffer area provided by the client application where the state name
                  will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.6.8 WMCloseWorkitemStatesList

NAME

WMCloseWorkitemStatesList - Closes the query for workitem states.

DESCRIPTION

WMTErrRetType WMCloseWorkitemStatesList ( In WMTPSessionHandle psession_handle, In WMTPQueryHandle pquery_handle)

Argument Name    Description
psession_handle   Pointer to a structure containing information about the context for this action.
pquery_handle    Identification of the specific query handle returned by the WMOpenWorkitemStatesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.6.9 WMChangeWorkitemState

NAME

WMChangeWorkitemState - Changes the state of the named workitem.

DESCRIPTION

This command is defined to allow a workitem to be changed temporarily to a specific state such as notRunning, or running. See Appendix G for a discussion of states.

```c
WMTErrRetType WMChangeWorkitemState ( 
    in WMTPSessionHandle psession_handle, 
    in WMTPProcDefID pworkitem_id, 
    in WMTPProcDefState pworkitem_state)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this</td>
</tr>
<tr>
<td></td>
<td>action.</td>
</tr>
<tr>
<td>pworkitem_id</td>
<td>Pointer to a structure containing a unique workitem ID.</td>
</tr>
<tr>
<td>pworkitem_state</td>
<td>Pointer to a structure that contains the name of the state to change the</td>
</tr>
<tr>
<td></td>
<td>workitem to.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_DEFINITION
WM_INVALID_STATE
WM_TRANSITION_NOT_ALLOWED

REQUIREMENTS

Each workitem must have a unique ID within an administrative scope.

RATIONALE FOR API

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Work Item State
Event Code: WMChangedWorkItemState
6.6.10 WMReassignWorkItem

NAME
WMReassignWorkItem

DESCRIPTION
This command allows a work item from one workflow participant’s worklist to be reassigned to another workflow participant’s worklist.

(Note: Possible future releases of the API specification may provide for an entire worklist to be reassigned in total.)

WMTP:: WMReassignWorkItem (in WMTPSessionHandle psession_handle, in WMTPWflParticipant psoure_user, in WMTPWflParticipant ptarget_user, in WMTPProcInstID pproc_inst_id, in WMTPWorkItemID pwork_item_id)

Argument Name | Description
---------------|----------------------------------------
psession_handle | Pointer to a structure containing information about the context for this action.
psource_user | The identification of a workflow participant from which work is to be reassigned.
ptarget_user | The identification of the workflow participant to whom work is to be assigned.
pproc_inst_id | Pointer to a structure containing the unique process instance ID.
pwork_item_id | Pointer to a structure containing the work item identification being reassigned.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM
WM_INVALID_SOURCE_USER
WM_INVALID_TARGET_USER

REQUIREMENTS
The workflow participant making the reassignment request has the authority to do so.

RATIONALE FOR API
A workflow participant having work assigned may be away from work for various reasons and the work must be given to another workflow participant to get it accomplished. A WFM Engine may direct all work items to a single worklist (departmental worklist for example).

With the reassignment API, workflow participants in that department may reassign work to themselves after they finish a current work item and become available for more work. This creates a possible de facto people load balancing scheme.

AUDIT INFORMATION
The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Assign / Reassign Work Item
Event Code: WMReassignedWorkItem
6.6.11 WMOpenWorkItemAttributesList

NAME

WMOpenWorkItemAttributesList - Specifies and opens the query to produce the list of work item attributes that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of attributes for a work item. The command will also return, optionally, the total count of attributes available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available attributes that can be assigned to the work item, in order to offer this list to the application user. Attribute values can be obtained as well provided that a buffer of enough size is passed in the fetch call. Individual attribute values can also be retrieved with the WMGetWorkItemAttributeValue call. If pwork_item_attr_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL attributes available for the work item.

WMTErrRetType WMOpenWorkItemAttributesList ( 
   in WMTPSessionHandle psession_handle, 
   in WMTPProcInstID pproc_inst_id, 
   in WMTPWorkItemID pwork_item_id, 
   in WMTFFilter pwork_item_attr_filter, 
   in WMTBoolean count_flag, 
   out WMTPQueryHandle pquery_handle, 
   out WMTPInt32 pcount)

Argument Name | Description
--- | ---
p/session_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the unique process instance ID.
pwork_item_id | Pointer to a structure containing the unique work item ID.
pwork_item_attr_filter | Filter associated with the work item attributes.
count_flag | Boolean flag that indicates if the total count of work item attributes should be returned.
pquery_handle | Pointer to a structure containing a unique query information.
pcount | Total number of attributes for this work item.

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_INSTANCE
- WM_INVALID_WORKITEM
6.6.12 WMFetchWorkItemAttribute

NAME

WMFetchWorkItemAttribute - Returns the next work item attribute from the list of work item attributes that match the filter criterion.

DESCRIPTION

This command returns a work item attribute. This fetch function will return subsequent work item attributes after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The fetch function will return the attribute value as well in a buffer specified in the call. If buffer_size is NULL then the attribute value will not be returned. If buffer_size is not large enough to hold the attribute value then the function will return as much of the attribute value as can be fit in the buffer. The proper length of the attribute value is available in the attribute_length field. The application can compare the attribute_length with the buffer_size to determine if the full value was returned.

WMTErrRetType WMFetchWorkItemAttribute ( 
   in WMTPSessionHandle psession_handle, 
   in WMTPQueryHandle pquery_handle, 
   out WMTPAttrName pattribute_name, 
   out WMTPInt32 pattribute_type, 
   out WMTPInt32 pattribute_length, 
   out WMTPText pattribute_value, 
   in WMTInt32 buffer_size)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pquery_handle Identification of the specific query handle returned by the WMOpenWorkItemAttributesList query command.
pattribute_name Pointer to the name of the attribute.
pattribute_type Pointer to the type of the attribute.
pattribute_length Pointer to the length of the attribute value.
pattribute_value Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size Size of the buffer.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
6.6.13 WMCloseWorkItemAttributesList

NAME

WMCloseWorkItemAttributesList - Closes the query for work item attributes.

DESCRIPTION

WMTErrRetType WMCloseWorkItemAttributesList (
    [in] WMTPSessionHandle psession_handle,
    [in] WMTPQueryHandle pquery_handle)

Argument Name   Description
psession_handle  Pointer to a structure containing information about the context for this action.

pquery_handle   Identification of the specific query handle returned by the
                WMOpenWorkItemAttributesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
6.6.14 WMGetWorkItemAttributeValue

NAME

WMGetWorkItemAttributeValue - Returns the value, type and length of a work item attribute specified by the pwork_item_id parameter.

DESCRIPTION

This command will return the value of a work item attribute in the buffer specified in the call.

WMErrRetType WMGetWorkItemAttributeValue (in WMTPSessionHandle psession_handle, in WMTPProcInstID pproc_inst_id, in WMTPWorkItemID pwork_item_id, in WMTPAttrName pattribute_name, out WMTPInt32 pattribute_type, out WMTPInt32 pattribute_length, out WMTPText pattribute_value, in WMTInt32 buffer_size)

Argument Name | Description
--- | ---
psession_handle | Pointer to a structure containing information about the context for this action.
pproc_inst_id | Pointer to a structure containing the unique process instance ID.
pwork_item_id | Pointer to a structure containing the unique work item ID.
pattribute_name | Pointer to the name of the attribute.
pattribute_type | Pointer to the type of the attribute.
pattribute_length | Pointer to the length of the attribute value.
pattribute_value | Pointer to a buffer area provided by the client application where the attribute value will be placed.
buffer_size | Size of the buffer to be filled.

ERROR RETURN VALUE

WM_SUCCESS
WN_INVALID_SESSION_HANDLE
WN_INVALID_ATTRIBUTE
WN_INSUFFICIENT_BUFFER_SIZE
6.6.15 WMAssignWorkItemAttribute

NAME

WMAssignWorkItemAttribute - Assign the proper attribute to a work item.

DESCRIPTION

This command tells the WFM Engine to assign an attribute, to change an attribute or to change the value of an attribute of a work item.

WMErrRetType WMAssignWorkItemAttribute ( 
  in WMTPSessionHandle psession_handle, 
  in WMTPProcInstID pproc_inst_id, 
  in WMTPWorkItemID pwork_item_id, 
  in WMTPAttrName pattribute_name, 
  in WMITInt32 attribute_type, 
  in WMITInt32 attribute_length, 
  in WMTPText pattribute_value)

Argument Name  Description
psession_handle  Pointer to a structure containing information about the context for this action.
pproc_inst_id  Pointer to a structure containing the unique process instance ID.
pwork_item_id  Pointer to a structure containing the work item ID for which an attribute will be added or changed.
pattribute_name  Pointer to the name of the attribute.
attribute_type  Type of the attribute.
attribute_length  Length of the attribute value.
pattribute_value  Pointer to a buffer area provided by the client application where the attribute value will be placed.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_PROC_INST
WM_INVALID_ATTRIB
WM_ATTRIBUTE_ASSIGNMENT_FAILED

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Assign Work Item Attributes
Event Code: WMAssignWorkItemAttributes
6.7 WAPI Administration Functions

The set of administration functions provide the functionality needed to perform administration and maintenance functions of a workflow system. This set includes the minimal services contemplated for this client application interface. The set includes functions to change state of a set of process or activity instances, terminating and aborting process instances, and for assigning attributes to a set of process and activity instances.

6.7.1 WMChangeProcessInstancesState

NAME

WMChangeProcessInstancesState - Change the state of the instances of the named process definition that match the specified filter criterion.

DESCRIPTION

This command is defined to allow a set of process instances in the named process definition to move to a specific new state.

Execution of this command will cause a set of process instances of the named process definition change their state. If the filter pointer pproc_inst_filter is NULL, then the command is applied to all process instances. Specific state names and their semantics are dependent upon the particular WFM Engine implementation.

This call will be executed when a set of process instances of a process must have a new state, such as suspended, disabled or enabled. Specific state names and semantics must be included in implementation documentation.

Since this command operates on a set of process instances of a named process definition, it is expected to be issued by a person having the authority to do so. The scope of this operation may be different depending on the vendor’s implementation.

WMTErrRetType WMChangeProcessInstancesState (  
  in WMTPSessionHandle psession_handle,  
  in WMTPProcDefID pproc_def_id,  
  in WMTPFilter pproc_inst_filter,  
  in WMTPProcInstState pproc_inst_state)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
Pproc_def_id Pointer to a structure containing a unique process definition ID.
pproc_inst_filter Pointer to a structure containing the filter information for this request.
pproc_inst_state An ID that indicates the process state that you want to change to.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_PROCESS_DEFINITION  
WM_INVALID_FILTER  
WM_INVALID_STATE  
WM_TRANSITION_NOT_ALLOWED

REQUIREMENTS

Each process instance must have a unique ID within an administrative scope.
Each process definition must have a unique ID within an administrative scope.
RATIONALE FOR API

This API allows the possible intervention of a process administrator in a running process. This might be for the purpose of changing the process definition and having all subsequently created instances reflect the new definition. It provides the capability of halting running process instances while changes in roles, activities, etc. are put into effect. It allows instances to be stopped while problem determination can be done on a malfunctioning process.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

<table>
<thead>
<tr>
<th>Audit Data Type</th>
<th>Change Process / Subprocess Instance State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Code:</td>
<td>WMChangedProcessInstanceState</td>
</tr>
</tbody>
</table>
6.7.2 WMChangeActivityInstancesState

NAME

WMChangeActivityInstancesState - Change the state of the activity instances of a particular name associated to a process definition that match the specified filter criterion.

DESCRIPTION

This command directs a WFM Engine to change the state of the named activity for a set of activity instances. It is assumed that a person who can change the state of the set of activity instances corresponding to a process definition has special authorization to do so. If the implementation supports a state such as suspended, and resumed or in-progress, then the functions for suspend and resume are implemented as state change calls. If the filter pointer pact_inst_filter is NULL, then the command is applied to all activity instances of the given activity definition.

WMTErrRetType WMChangeActivityInstancesState (  
  in WMTPSessionHandle psession_handle,  
  in WMTPProcDefID pproc_def_id,  
  in WMTPActivityID pactivity_def_id,  
  in WMTPFilter pact_inst_filter,  
  in WMTPActivityInstState pactivity_inst_state)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
Pproc_def_id Pointer to a structure containing a unique process definition ID.
Pactivity_def_id Pointer to the activity definition ID.
Pact_inst_filter Pointer to a structure containing the filter information for this request.
Pactivity_inst_state An ID that indicates the activity instance state that you want to change to.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_DEFINITION
WM_INVALID_ACTIVITY_NAME
WM_INVALID_FILTER
WM_INVALID_STATE
WM_TRANSITION_NOT_ALLOWED

REQUIREMENTS

Each process definition must have a unique ID within an administrative scope.
Each activity must have a unique ID within a process definition.

RATIONALE FOR API

A workflow participant may wish to modify the states of activity instances of a particular activity. Other situations might involve the malfunctioning of an application associated with an activity. A process containing the activity may be a frequently used one, and it might be issuing dumps each time it is invoked. The use of this API would allow the calling of the application to be stopped while remedial measures were taken.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Activity Instance State
Event Code: WMChangedActivityInstanceState
6.7.3 WMTerminateProcessInstances

NAME

WMTerminateProcessInstances - Terminate the process instances of the named process definition that match the specified filter criterion.

DESCRIPTION

This command provides the capability of terminating a set of process instances associated with a process definition. Execution of this command will cause a set of process instances of the named process definition to be terminated. If the filter pointer pproc_inst_filter is NULL, then the command is applied to all process instances.

WMTErrRetType WMTerminateProcessInstances (  
   in WMTPSessionHandle psession_handle,  
   in WMTPProcDefID pproc_def_id,  
   in WMTPFilter pproc_inst_filter)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
Pproc_def_id Pointer to a structure containing the process definition for which all process instances are to be terminated.
Pproc_inst_filter Pointer to a structure containing the filter information for this request.

ERROR RETURN VALUE

WM_SUCCESS  
WM_INVALID_SESSION_HANDLE  
WM_INVALID_PROCESS_DEFINITION  
WM_INVALID_FILTER

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Process / Subprocess Instance State  
Event Code: WMTerminatedProcessInstance
6.7.4 WMAssignProcessInstancesAttribute

NAME

WMAssignProcessInstancesAttribute - Assign the proper attribute to a set of process instances within a process definition that match the specific filter criterion.

DESCRIPTION

This command tells the WFM Engine to assign an attribute, or to change an attribute or to change the values of an attribute of a set of process instances within a named process definition.

This command changes the value of the attribute of a process instance. These attributes of process instances are of the kind called Process Control or Process Relevant Data.

```c
WMTErrRetType WMAssignProcessInstancesAttribute (  
   in WMTPSessionHandle psession_handle,  
   in WMTPProcDefID pproc_def_id,  
   in WMTPFilter pproc_inst_filter,  
   in WMTPAttrName pattribute_name,  
   in WM TInt32 attribute_type,  
   in WM TInt32 attribute_length,  
   in WM TText pattribute_value)  
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>Pproc_def_id</td>
<td>Pointer to a structure containing the process definition ID for which the attribute of all process instances will be changed.</td>
</tr>
<tr>
<td>Pproc_inst_filter</td>
<td>Pointer to a structure containing the filter information for this request.</td>
</tr>
<tr>
<td>Pattribute_name</td>
<td>Pointer to the name of the attribute.</td>
</tr>
<tr>
<td>Attribute_type</td>
<td>Type of the attribute.</td>
</tr>
<tr>
<td>Attribute_length</td>
<td>Length of the attribute value.</td>
</tr>
<tr>
<td>Pattribute_value</td>
<td>Pointer to a buffer area provided by the client application where the attribute value will be placed.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_DEFINITION
- WM_INVALID_FILTER
- WM_INVALID_ATTRIBUTE

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

- Audit Data Type: Change Process Instance Attribute
- Event Code: WMAssignedProcessInstanceAttributes
**WMAssignActivityInstancesAttribute**

**NAME**

WMAssignActivityInstancesAttribute - Assign the proper attribute to set of activity instances within a process definition that match the specific filter criterion.

**DESCRIPTION**

This command tells the WFM Engine to assign an attribute, or to change an attribute or to change the value of an attribute of a set of activity instances within a named process definition. These attributes of activity instances are of the kind called Process Control or Process Relevant Data. If pact_inst_filter is NULL, then the function is applied to ALL accessible activity instances of the given activity definition.

```c
WMTErrRetType WMAssignActivityInstancesAttribute (  
    in WMTPSessionHandle psession_handle,  
    in WMTPProcDefID pproc_def_id,  
    in WMTPActivityID pactivity_def_id,  
    in WMTPFilter pact_inst_filter,  
    in WMTPAttrName pattribute_name,  
    in WMTInt32 attribute_type,  
    in WMTInt32 attribute_length,  
    in WMTPText pattribute_value)
```

**Argument Name**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pproc_def_id</td>
<td>Pointer to a structure containing the process definition ID. In the case that the attribute will be changed for all activity instances that correspond to the process definition. This parameter will be NULL otherwise.</td>
</tr>
<tr>
<td>pactivity_def_id</td>
<td>Pointer to a structure containing the activity definition identification for which the attribute will be assigned.</td>
</tr>
<tr>
<td>pact_inst_filter</td>
<td>Pointer to a structure containing the filter information for this request.</td>
</tr>
<tr>
<td>pattribute_name</td>
<td>Pointer to the name of the attribute.</td>
</tr>
<tr>
<td>attribute_type</td>
<td>Type of the attribute.</td>
</tr>
<tr>
<td>attribute_length</td>
<td>Length of the attribute value.</td>
</tr>
<tr>
<td>pattribute_value</td>
<td>Pointer to a buffer area provided by the client application where the attribute value will be placed.</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

- WM_SUCCESS
- WM_INVALID_SESSION_HANDLE
- WM_INVALID_PROCESS_DEFINITION
- WM_INVALID_ACTIVITY_NAME
- WM_INVALID_FILTER
- WM_INVALID_ATTRIBUTE

**AUDIT INFORMATION**

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

- Audit Data Type: Assign Activity Instance Attributes
- Event Code: WMAssignedActivityInstanceAttributes
6.7.6 WMAbortProcessInstances

NAME

WMAbortProcessInstances - Abort the set of process instances that correspond to the named process definition, that match the specific filter criterion, regardless of its state.

DESCRIPTION

This command allows a set of process instances of a process definition to be aborted. All current activities within these process instances will be stopped when possible. The instances will be terminated. If pproc_inst_filter is NULL, then the function will be applied to ALL accessible process instances.

```c
WMErrRetType WMAbortProcessInstances(  
    in WMTPSessionHandle psession_handle, 
    in WMTPProcDefID pproc_def_id, 
    in WMTPFilter pproc_inst_filter)
```

Argument Name  Description

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pproc_def_id</td>
<td>Pointer to a structure containing the process definition for who all processes instances is being aborted.</td>
</tr>
<tr>
<td>pproc_inst_filter</td>
<td>Pointer to a structure containing the filter information for this request.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

<table>
<thead>
<tr>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_SUCCESS</td>
</tr>
<tr>
<td>WM_INVALID_SESSION_HANDLE</td>
</tr>
<tr>
<td>WM_INVALID_PROCESS_DEFINITION</td>
</tr>
<tr>
<td>WM_INVALID_FILTER</td>
</tr>
</tbody>
</table>

REQUIREMENTS

None

RATIONALE FOR API

This command is for use in catastrophic circumstances where nothing except clearing the process away can be done.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

<table>
<thead>
<tr>
<th>Audit Data Type</th>
<th>Event Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Process Instance State</td>
<td>WMAbortedProcessInstance</td>
</tr>
</tbody>
</table>
6.7.7 WMAbortProcessInstance

NAME

WMAbortProcessInstance - Abort the process instance specified regardless of its state.

DESCRIPTION

This command allows a process instance to be aborted. All current activities within the process instance will be stopped when possible. The instance will be terminated.

WMErrRetType WMAbortProcessInstance (in WMTPSessionHandle psession_handle, in WMTPProcInstID pproc_inst_id)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pproc_inst_id Pointer to a structure containing the process instance being aborted.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE

REQUIREMENTS

None

RATIONALE FOR API

This command is for use in catastrophic circumstances where nothing except clearing the process away can be done.

AUDIT INFORMATION

The following audit information is directly related to this function and would be recorded by an implementation of this specification that complies with the Audit Profile:

Audit Data Type: Change Process / Subprocess Instance State
Event Code: WMAbortedProcessInstance

6.8 WAPI Application Invocation Functions

The set of application interface functions provides services to Tool-Agents, to invoke and control applications associated with specific work items.

The Invoked Application Interface defines an API set, which is highly recommended to be used by Workflow System components (engine and client applications) to control specialized application drivers called Tool Agents. These Tool Agents finally start up and stop applications, pass workflow and application relevant information to and from the application and control the application’s run level status.

Therefore, the Invoked Application Interface WAPIs are only directed against a Tool Agent. Nevertheless, additional workflow information could be requested by an application via the Tool Agent using standard WAPI functions. As the Invoked Application Interface should handle bi-directional requests (requests to and from applications), it depends on the interfaces and architecture of applications how to interact with an Tool Agent.
This interface will allow the request and update of application data and more run-time relevant functionalities.

![Diagram]

Fig. 1: The localization of the Invoking Application Interface.

The Workflow System itself has to know about the installed Tool Agents. The basic architecture of Tool Agents could be compared with a driver - interface, i.e. ODBC, etc..

Within this interface definition, no further communication mechanism between the Tool Agents and the Workflow System is necessary.

### 6.8.1 WMTAConnect() & WMTADisconnect()

**DESCRIPTION**

These commands create and terminate connections to Tool Agent interfaces. The commands are already defined in section “WAPI Connection Functions”. Applications might require login procedures, therefore user authentication should be passed to a Tool Agent to provide single-login mechanisms.

Note: The value for engine_name in WMTConnectInfo represents the name of the Tool Agent implementation as defined in the process definition.
6.8.2 WMTAInvokeApplication()

NAME

WMTAInvokeApplication - Force a Tool Agent to start or activate a specific application.

DESCRIPTION

The workflow application or engine activates a specified application associated with a work item by calling this Tool Agent API. Applications could be already active (started) or have to be invoked (started) by the Tool Agent. Invoking an application always includes passing of additional options like application parameters and modes.

```c
int WMTAInvokeApplication (  
    in int tool_agent_handle,  
    in string application_name,  
    in WMTPProcInstID pproc_inst_id,  
    in WMTPWorkItemID pwork_item_id,  
    in WMTPAttributeList pattribute_list,  
    in int app_mode)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool_agent_handle</td>
<td>This handle represents one connection to a specific Tool Agent</td>
</tr>
<tr>
<td>application_name</td>
<td>This parameter represents the name of the executable file or component. The application name must be passed without the path name. (The Tool Agent implementation and configuration has to handle the local configuration.)</td>
</tr>
<tr>
<td>pproc_inst_id</td>
<td>Process instance, to identify the relation between the application and a process instance. This ID allows the System to reference to a specific application handle of the Tool Agent.</td>
</tr>
<tr>
<td>pwork_item_id</td>
<td>Work Item associated with invoked application</td>
</tr>
<tr>
<td>pattribute_list</td>
<td>Pointer to a list of parameters and attributes which are required by the application. These parameters could be either application relevant, or dynamic, or workflow relevant data. (e.g. filename, record identifier, etc.)</td>
</tr>
<tr>
<td>app_mode</td>
<td>Represents a possible application mode like “CREATE”, “UPDATE”, “READ-ONLY”, ”PRINT”, etc..</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

- WM_SUCCESS
- WM_APPLICATION_NOT_STARTED
- WM_APPLICATION_NOT_DEFINED
- WM_APPLICATION_BUSY

REQUIREMENTS

None

RATIONALE FOR API

This command invokes a specific application associated with a work item. A Tool Agent might control one or multiple applications, which have to be started or activated. Also, an application have to be started in a specific mode like “open” or “update”.

AUDIT INFORMATION

None
6.8.3 WMTARequestAppStatus()

NAME

WMTARequestAppStatus - allows the Workflow System to check for open applications and their status (running, pending, etc.).

DESCRIPTION

WMTARequestAppStatus() defines how the Workflow System has to check the status of an application and retrieves workflow relevant data from the application. To retrieve workflow relevant data from an invoked application, the workflow application or engine has to request the application status and information from a Tool Agent. Due to some asynchronous requirements of integrated applications, Tool Agents can request additional information by use of other WAPI interfaces.

```c
int WMTARequestAppStatus (
  in WM TInt32 tool_agent_handle,
  in WM TPProcInstID proc_inst_id,
  in WM TPWorkItemID pwork_item_id,
  out WM TInt32 app_status,
  out WM TAttributeList WFRelevantData)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool_agent_handle</td>
<td>This handle represents one specific Tool Agent</td>
</tr>
<tr>
<td>proc_inst_id</td>
<td>Workflow relevant data belong to this process instance and should be updated after the application is finished.</td>
</tr>
<tr>
<td>pwork_item_id</td>
<td>Work Item associated with invoked application.</td>
</tr>
<tr>
<td>app_status</td>
<td>Information about the invoked application. (I.e. “RUNNING”, “ACTIVE”, “WAITING”, “TERMINATED”, “FINISHED”, etc.)</td>
</tr>
<tr>
<td>WFRelevantData</td>
<td>A list or structure of workflow relevant data, which could be accessed by the Tool Agent mechanisms.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_SUCCESS
WM_APPLICATION_BUSY
WM_INVALID_TOOL_AGENT_HANDLE
WM_INVALID_WORKITEM
WM_INVALID_PROCESS_INSTANCE

REQUIREMENTS

None

RATIONALE FOR API

To check the status of an active work item this command might be used to control the status of an invoked application.

AUDIT INFORMATION

None
6.8.4 WMTATerminateApp()

NAME
WMTATerminateApp - Forces the Tool Agent to terminate an application.

DESCRIPTION
This API allows the Workflow System to stop an application, which relates to a specific process instance. Also, an application can be terminated by any other event. Therefore, WMTerminateApp is not mandatory within the application control APIs, but it allows the ToolAgent to free application relevant information.

```c
int WMTATerminateApp (in WM TInt32 tool_agent_handle,
                         in WM TPProcInstID pproc_inst_id,
                         in WM TPWorkItemID pwork_item_id)
```

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tool_agent_handle</td>
<td>This handle represents one specific Tool Agent</td>
</tr>
<tr>
<td>pproc_inst_id</td>
<td>Workflow relevant data belong to this process instance and should be updated after the application is finished.</td>
</tr>
<tr>
<td>pwork_item_id</td>
<td>Work Item associated with invoked application.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

- WM_SUCCESS
- WM.APPLICATION_NOT_STOPPED
- WM.INVALID_PROCESS_INSTANCE
- WM.INVALID_WORKITEM
- WM.APPLICATION_BUSY

REQUIREMENTS

None

RATIONALE FOR API

This command is to close a connection to an application and to stop it. It might be used before system shutdown, or to terminate invoked applications to allow better control of system resources used by integrated applications.

AUDIT INFORMATION

None
7. Appendix A: Future Work

7.1 Additional API Areas

The WFM Coalition API specification work will address the following areas. It will be determined whether API calls should be created for these areas or whether they are the sole domain of particular WFM product implementations.

7.1.1 WFM Data API calls

The types of data that applications need to manipulate through this API specification are process control data, process relevant data, and application data. The current specification addresses the access to these data through the definition and manipulation of attributes of processes, activities and work items. It is currently believed that some additional new API calls or parameter additions to existing API calls will be required for complete data manipulation.

7.1.2 Ad hoc activities

In a future release of API specifications, the API working group will consider the functionality to allow applications to add activities to an instance of a process that are not part of its definition. These ad-hoc additions will be done on an instance basis.

7.1.3 Administration and Maintenance

The API working group believes that the functions in this area correspond to interface 5. Services should include functions for:

- Purging
- Backup
- Archiving
- Download and Upload instances (for remote users)

7.1.4 Names and Roles

The API working group believes that a Workflow Engine should also provide services for definition, assignment, mapping and maintenance of roles and names (identities). The working group also believes that these services should be provided through interface 5 as well.

7.2 Additional Issues

The WFM Coalition API specification work will be expanded to take care of the following issues for future releases.

7.2.1 Error reporting and control

All WAPI function calls have a uniform error return datatype. This data type is shared among all API calls. This specification assumes that the Coalition will specify a subset of the main error return codes, leaving for vendor specific implementation the remaining main error return codes and the set of subcode codes to provide extensibility and specialization of error codes. (See section WAPI Data Types, and WAPI Error Return Codes sections).

7.2.2 Synchpoint processing

Synchpoint processing deals with recoverability. The API working group believes that this area is extremely important to WFM exploiters. However, it is also believed that it would be one of the more
difficult areas to deal with in terms of member agreement. Work in this area is being deferred to the second release of the API specifications.

7.2.3 Security
The current version of the WFM API specification does not include any specific requirements or provisions for security mechanisms, except for the inclusion of user password in the WMTConnectInfo structure. Implementation of security mechanisms are left up to the specific implementations.

7.2.4 Locking
The current version of the WFM API specification does not include any specific requirements or provisions for locking mechanisms. Implementation of locking mechanisms are left up to the specific implementations.

7.2.5 Process Integrity
The current version of the WFM API specification does not include any specific requirements or provisions for mechanisms to guarantee process integrity. Implementation of process integrity mechanisms are left up to the specific implementations.
8. Appendix B: Object Bindings

This chapter describes the object bindings for the WAPI functions described in this document\(^1\). Bindings are defined for OLE and in terms of OMG IDL. Both bindings realize a common object model that is described in the next section; the rest of the chapter describes the binding specifications.

8.1 Abstract Object Definition

The following diagram shows the primary objects used in the definition of the Workflow Application Client interface.

![Diagram of object relationships]

The WorkflowServer provides the context for communication with the Workflow Enactment Service. It allows for filtered queries on objects owned by the specific Enactment Service. An executable workflow model itself is represented by the ProcessDefinition; the Process Definition serves as a Factory for creating instances of the Workflow Model that are enacted by the Workflow Management System. To execute a specific process, a ProcessInstance of the ProcessDefinition is created. During execution of the ProcessInstance, the Enactment Service creates instances (ActivityInstance) of the Activity Definitions contained in the Process Definition. Assignment of an activity instance to a participant creates a WorkItem.

The next diagram shows the auxiliary constructs that are used to complete the Object Model.

---

\(^1\) The new Process Definition functions are not covered here at the moment.
A set of standardized Attributes is defined for each of the objects - attributes specific to a particular Enactment Service or user-defined attributes that determine the specifics of a Workflow object in a particular Workflow Model. Access to filtered lists of objects owned by the Enactment Service is managed via Collection-type interfaces; Filter objects support definition of selection criteria for those lists.

Workflow Object Collections are realized as OLE-Collections in the OLE binding; in the OMG IDL binding an Iterator-type interface is defined for each of the fundamental Workflow Object interfaces.

### 8.1.1 Mapping WAPI to the OLE and IDL Bindings

The following table describes how the ingredients of the Common Object Model described above map to the WAPI constructs defined in this specification.

<table>
<thead>
<tr>
<th>WAPI Element</th>
<th>OLE Object</th>
<th>IDL Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMTSession</td>
<td>Server</td>
<td>ApplicationClientServer</td>
</tr>
<tr>
<td>WMTFilter</td>
<td>Filter</td>
<td>Filter</td>
</tr>
<tr>
<td>WMTQueryHandle</td>
<td>OLE-Collection</td>
<td>ProcessDefinitionsList</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProcessInstancesList</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ActivityInstancesList</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WorkList</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AttributeList</td>
</tr>
<tr>
<td>WMTProcessDefinition</td>
<td>ProcessDefinition</td>
<td>ProcessDefinition</td>
</tr>
<tr>
<td>WMTProcessInstance</td>
<td>ProcessInstance</td>
<td>ProcessInstance</td>
</tr>
<tr>
<td>WMTActivityInstance</td>
<td>ActivityInstance</td>
<td>ActivityInstance</td>
</tr>
<tr>
<td>WMTWorkItem</td>
<td>WorkItem</td>
<td>WorkItem</td>
</tr>
<tr>
<td>WMTAttributeName</td>
<td>Attribute</td>
<td>Attribute</td>
</tr>
<tr>
<td>WMTAttributeType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMTAttributeLength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMTAttributeValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMTEntity of type Process</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Workflow Object Collections are realized as OLE-Collections in the OLE binding; in the OMG IDL binding an Iterator-type interface is defined for each of the fundamental Workflow Object interfaces.
8.2 OLE Automation Binding

This appendix describes the OLE automation binding for the Workflow Management Coalition Interface 2 (WAPI2). It is based on:

- the WAPI specified in this document, and
- *Microsoft Visual Basic 4.0, Professional Features, Creating OLE Servers, Chapter 3, Standards and Guidelines*

This binding has two goals:

1. To accurately reflect the functionality specified by WAPI2.
2. To conform to the standards and guidelines for OLE automation interfaces.

Note that this version of the binding does not yet include the “entity” functions.

8.2.1 Expressing WAPI2 as an OLE Automation Interface

WAPI2 is defined in terms of data structures and functions. An OLE automation interface consists of object classes, each with properties and methods. The OLE automation binding for WAPI2 was derived using the following rules:

1. Define an OLE automation object class for each WAPI2 data structure. However, if a WAPI2 data structure consists of a single WMTText field, use the OLE automation String class.
2. Define a read-only OLE automation property for each field in each WAPI2 data structure, on the object class corresponding to the data structure.
3. For each WAPI2 function, define a method on the appropriate object class. Omit the session handle parameter from the methods (except for the Server methods).
4. Use OLE automation collections for each Open/Fetch/Close...List combination of functions, and for fields in data structures that hold multiple values (e.g. participants).
5. Errors are reported via exceptions.

8.2.1.1 Object Classes

The OLE automation binding defines an OLE automation object (class) for each WAPI2 data structure. For example, WAPI2 defines a process instance data structure as follows:

| WMTEntity of type Transition Information | TransitionDefinition |
| WMTEntity of type Application Definition | ApplicationDefinition |
| WMTEntity of type Participant Definition | ParticipantDefinition |
| WMTEntity of type Process Relevant Data | DataDefinition |
typedef struct
{
    // This is the minimum list of elements at this time. Future versions to provide
    // extensibility for this structure.
    WMTText process_name[NAME_STRING_SIZE];
    WMTProcInstID proc_inst_id;
    WMTProcDefID proc_def_id;
    WMTProcInstState state;
    WMTInt32 priority;
    WMTText data_reference[DATA_REFERENCE_SIZE];
        // private element containing vendor specific
    // information
    WMTWflParticipant proc_participants[20];
        // up to 20 63 character long participant identifiers
} WMTProcInst;

The OLE automation binding defines a ProcessInstance object class with properties Name, ID,
ProcessDefinitionID, State, Priority, DataReference, and Participants. This ProcessInstance object class
defines Start and Terminate methods, corresponding to the WMStartProcess and
WMTerminateProcessInstance WAPI2 functions.

The table below lists the object classes in the WAPI2 OLE automation binding, and the corresponding
WAPI2 data structures. Note that there are exceptions to the rules stated above. There is no
ConnectionInfo object class - the information is passed as separate parameters to the Connect method of the
Server class. There is an Attribute object class - its properties are passed as separate parameters in the
WAPI2 attribute functions.

<table>
<thead>
<tr>
<th>OLE Automation Object</th>
<th>WAPI 2 Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>WMTSessionHandle</td>
</tr>
<tr>
<td>Filter</td>
<td>WMTFilter</td>
</tr>
<tr>
<td>Collection</td>
<td>WMTQueryHandler</td>
</tr>
<tr>
<td>ProcessDefinition</td>
<td>WMTProcDefID</td>
</tr>
<tr>
<td>ProcessInstance</td>
<td>WMTProcInst</td>
</tr>
<tr>
<td>ActivityInstance</td>
<td>WMTActivityInst</td>
</tr>
<tr>
<td>WorkItem</td>
<td>WMTWorkItem</td>
</tr>
<tr>
<td>Attribute</td>
<td>WMTWflParticipant</td>
</tr>
<tr>
<td>String</td>
<td>WMTProcDefState</td>
</tr>
<tr>
<td></td>
<td>WMTProcInstState</td>
</tr>
</tbody>
</table>

The table below lists the entities in the Workflow Process Definition Language (WPDL), and the
corresponding WAPI2 OLE automation binding objects.

<table>
<thead>
<tr>
<th>OLE Automation Object</th>
<th>WPDL Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessDefinition</td>
<td>Workflow Process Definition</td>
</tr>
<tr>
<td>ActivityDefinition</td>
<td>Workflow Process Activity</td>
</tr>
<tr>
<td>TransitionDefinition</td>
<td>Transition Information</td>
</tr>
<tr>
<td>ParticipantDefinition</td>
<td>Workflow Participant Definition</td>
</tr>
<tr>
<td>ApplicationDefinition</td>
<td>Workflow Application Definition</td>
</tr>
<tr>
<td>DataDefinition</td>
<td>Workflow Process Relevant Data</td>
</tr>
</tbody>
</table>

8.2.1.2 Object Hierarchy

The object classes in an OLE automation interface are organized into an object hierarchy. This is not an
inheritance hierarchy based on “is a” relationships. Rather, it is a navigational hierarchy that “organizes the
objects in a way that makes programming easier”. The top level objects in the hierarchy are “externally
creatable”, which means that a program can obtain such objects directly. All other objects in the OLE automation interface are obtained indirectly, through the properties and methods of the top level objects. Here is the object hierarchy for the WAPI2 OLE automation interface:

```
Server
  Process Definitions
    States
    ActivityDefinitions
    TransitionDefinitions
    ParticipantDefinitions
    ApplicationDefinitions
    ProcessDataDefinitions
  Process Instances
    Attributes
    Participants
    States
  Activity Instances
    Attributes
    Participants
    States
  Work Items
    Attributes
    Participant
    ParticipantDefinitions
    ApplicationDefinitions
Filter
```

WAPI2 requires a program to first obtain a session handle, and then use it to get process, activity, and work item handles. In the OLE automation binding, Server and Filter are the top level objects. The Server object class has methods for listing process definitions, process instances, activity instances, and work items.

### 8.2.1.3 Collections and Queries

WAPI2 supports several retrieval operations that return multiple values:

- a list of process definitions, process instances, activity instance, or work items,
- the states of a process definition, process instance, or activity instance
- the attributes of a process instance, activity instance, or work item

For each such retrieval operation, WAPI2 defines three functions:

- `WMOpen...List`
- `WMFetch...`
- `WMClose...List`

The open functions take a filter parameter. The fetch functions are used to iterate through the values retrieved.

OLE automation uses the Collection object class to navigate such one-to-many relationships in the object hierarchy. The Server object class has list methods which take a Filter object as a parameter and return a collection of ProcessDefinition, ProcessInstance, ActivityInstance, or WorkItem objects. The ProcessDefinition, ProcessInstance, and ActivityInstance object classes have a States property whose value is a collection of states. The ProcessInstance, ActivityInstance, and WorkItem object classes have an Attributes property whose value is a collection of attributes. These properties have a Filter parameter.
The Collection object class has a Count property (the number of elements in the collection) and provides methods for iterating through its elements. For example, here is the VBA code to populate a list box with a user’s work items:

```vba
Dim mySession As Session
Dim myWorkList As Filter
Dim myWorkItem As WorkItem

Set mySession = CreateObject("WAPI2.Session")
mySession.Connect(...)  
For Each myWorkItem In Session.ListWorkItems(myWorkList)
    ListBox.AddItem myWorkItem.Name
Next myWorkItem
```

The ProcessDefinition, ProcessInstance, ActivityInstance, and WorkItem classes each have collection-valued properties for their states, attributes, and participants. The elements of the state and participant collections are strings. The elements of the attribute collections are Attribute objects, which have two properties: Name and Value. The Value property is expressed as an OLE automation Variant, which provides methods for determining its data type and length.

### 8.2.1.4 Exceptions

OLE automation supports exceptions. OLE automation servers can report errors by raising an exception rather than returning an error code. This allows chaining calls to an OLE automation interface in a single expression. For example, the following expression

```vba
```

makes three calls to the OLE automation interface to return the name of the process definition for the process instance that contains the work item. These expressions commonly appear in programs or macros that call an OLE automation interface. Any one of the calls could encounter an error, which would be reported to the calling program through an exception.

The OLE automation binding for WAPI2 uses exceptions to report errors. The exception object carries a text description of the error with it. The Server object also has ErrorCode and ErrorSubCode properties. When a program calls the WAPI2 OLE automation interface, and the server encounters an error, it sets the Server properties to the error codes in the WMTErrRetType data structure, and raises an exception.

### 8.2.2 Attributes

Most workflow objects can have a collection of attributes, where each attribute has a name, data type, and value. The WAPI C binding provides functions for

- iterating through the attributes of an object: WMOpen…AttributesList, WMFetch…Attribute, WMClose…AttributesList, and
- getting and setting attribute values: WMGet…AttributeValue, WMAssign…AttributeValue.

In the OLE binding, each object has an Attributes property whose value is a collection of Attribute objects. The OLE collection object provides methods for iterating through the attributes of an object. An Attribute object has name, type and value properties corresponding to the attribute_name, attribute_type, attribute_length, and attribute_value parameters to the WMGet…AttributeValue function. The Attributes collection is indexed by attribute name. Getting the value of an attribute object has the same effect as calling WMGet…AttributeValue; setting the value of an attribute object has the same effect as calling WMAssign…AttributeValue. For example, the following expression

```vba
activity.Attributes("Size").Value
```
evaluates to the value of the “Size” attribute of an activity object (referenced by the activity variable), and

```
activity.Attributes("Size").Value = 15
```

evaluates to the value of the “Size” attribute of an activity object. The data type of the Value property is the OLE automation Variant type. This data type provides functions for determining the data type of its value, and converting its value to a basic data type.

### 8.2.3 Server

The Server object class corresponds to the WMTSessionHandle data type. Server objects are externally creatable. A program must successfully call the Connect method on a Server object before it can be used to access other objects.

#### 8.2.3.1 Properties

A Server object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td>String</td>
<td><code>pconnect_info.engine_name</code></td>
</tr>
<tr>
<td>ErrorCode</td>
<td>Integer</td>
<td><code>WMTErrRetType.main_code</code></td>
</tr>
<tr>
<td>ErrorSubCode</td>
<td>Integer</td>
<td><code>WMTErrRetType.sub_code</code></td>
</tr>
<tr>
<td>Scope</td>
<td>String</td>
<td><code>pconnect_info.scope</code></td>
</tr>
</tbody>
</table>

These properties are read-only. They are set when the OLE automation interface raises an exception.

#### 8.2.3.2 Methods

A Server object has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>WMConnect</td>
</tr>
<tr>
<td>in String User</td>
<td></td>
</tr>
<tr>
<td>in String Password</td>
<td></td>
</tr>
<tr>
<td>ProcessDefinition CreateProcessDefinition</td>
<td>WMCreateProcessDefinition</td>
</tr>
<tr>
<td>DeleteProcessDefinition</td>
<td>WMDeleteProcessDefinition</td>
</tr>
<tr>
<td>in ProcessDefinition ProcDef</td>
<td></td>
</tr>
<tr>
<td>Disconnect</td>
<td>WMDisconnect</td>
</tr>
<tr>
<td>Collection ListProcessDefinitions in Filter</td>
<td>WMOpenProcessDefinitionsList</td>
</tr>
<tr>
<td>ProcDefFilter</td>
<td></td>
</tr>
<tr>
<td>Collection ListProcessInstances in Filter</td>
<td>WMOpenProcessInstancesList</td>
</tr>
<tr>
<td>ProcInstFilter</td>
<td></td>
</tr>
<tr>
<td>ProcessInstanceId GetProcessInstanceId in</td>
<td>WMGetProcessInstance</td>
</tr>
<tr>
<td>ProcInstID</td>
<td></td>
</tr>
<tr>
<td>Collection ListActivityInstances in Filter</td>
<td>WMOpenActivityInstancesList</td>
</tr>
<tr>
<td>ProcInstID ProcInstInstFilter</td>
<td></td>
</tr>
<tr>
<td>ActivityInstanceId GetActivityInstanceId in</td>
<td>WMGetActivityInstance</td>
</tr>
<tr>
<td>ProcInstID ProcInstInstId</td>
<td></td>
</tr>
<tr>
<td>Collection ListWorkItems in Filter WorkItem</td>
<td>WMOpenWorkItemsList</td>
</tr>
<tr>
<td>ProcInstID WorkItemIdFilter</td>
<td></td>
</tr>
<tr>
<td>WorkItem GetWorkItem</td>
<td>WMGetWorkItem</td>
</tr>
<tr>
<td>in String ProcInstID</td>
<td></td>
</tr>
<tr>
<td>in String WorkItemID</td>
<td></td>
</tr>
</tbody>
</table>
ApplicationDefinition
CreateApplicationDefinition in String Name     WMCreateEntity
DeleteApplicationDefinition     WMDeleteEntity
ParticipantDefinition
CreateParticipantDefinition in String Name     WMCreateEntity
DeleteParticipantDefinition     WMDeleteEntity

8.2.3.3 Connect
This method is the binding for the WMConnect function. Note that the engine name and scope parameters to the WMConnect function are omitted here. This information is encoded in the parameters to the call to the OLE function (CreateObject or GetObject) which returns the server object.

```
Connect (
    in String User,
    in String Password)
```

Argument Description (WMConnect Argument)
User pconnect_info.user_identification
Password pconnect_info.password

8.2.3.4 CreateProcessDefinition
This method is the binding for the WMCreateProcessDefinition function.

```
ProcessDefinition CreateProcessDefinition ()
```

Argument Description (WMCreateProcessInstance Argument)
ProcessInstance pproc_def_id

8.2.3.5 DeleteProcessDefinition
This method is the binding for the WMDeleteDefinition function.

```
DeleteProcessDefinition (
    in ProcessDefinition ProcDef)
```

Argument Description (WMDeleteProcessDefinition Argument)
ProcDef pproc_def_id

8.2.3.6 WMDisconnect
This method is the binding for the WMDisconnect function.

```
Disconnect ()
```

8.2.3.7 ListProcessDefinitions
This method is the binding for the WMOpenProcessDefinitionsList function.
Collection ListProcessDefinitions (  
  in Filter ProcDefFilter)

Argument Description (WMOpenProcessDefinitionsList Argument)
ProcDefFilter pproc_def_filter
Collection pquery_handle

8.2.3.8 ListProcessInstances
This method is the binding for the WMOpenProcessInstancesList function.

Collection ListProcessInstances (  
  in Filter ProcInstFilter)

Argument Description (WMOpenProcessInstancesList Argument)
ProcInstFilter pproc_inst_filter
Collection pquery_handle

8.2.3.9 GetProcessInstance
This method is the binding for the WMGetProcessInstance function.

ProcessInstance GetProcessInstance (  
  in String ProcInstID)

Argument Description (WMGetProcessInstance Argument)
ProcInstID pproc_inst_id
ProcessInstance pproc_inst

8.2.3.10 ListActivityInstances
This method is the binding for the WMOpenActivityInstancesList function.

Collection ListProcessInstances (  
  in Filter ActivityInstFilter)

Argument Description (WMAActivityInstancesList Argument)
ActivityInstFilter pactivity_inst_filter
Collection pquery_handle

8.2.3.11 GetActivityInstance
This method is the binding for the WMGetActivityInstance function.

ActivityInstance GetActivityInstance (  
  in String ProcInstID,  
  in String ActivityInstID)

Argument Description (WMGetActivityInstance Argument)
ProcInstID pproc_inst_id
ActivityInstID pactivity_inst_id
ActivityInstance pactivity_inst

8.2.3.12 ListWorkItems
This method is the binding for the WMOpenWorkList function.
Collection ListWorkItems {
    in Filter WorkListFilter
}

Argument Description (WMOpenWorkList Argument)
WorkListFilter pworklist_filter
Collection pquery_handle

8.2.3.13 GetWorkItem
This method is the binding for the WMGetWorkItem function.

WorkItem GetWorkItem {
    in String ProcInstID,
    in String WorkItemID
}

Argument Description (WMGetWorkItem Argument)
ProcInstID pproc_inst_id
WorkItemID pwork_item_id
WorkItem pwork_item

8.2.3.14 CreateApplicationDefinition
This method is the binding for the WMCreateEntity function, when used to create a workflow application definition outside of any process definition.

ApplicationDefinition CreateParticipantDefinition ()

Argument Description (WMCreateEntity Argument)
Name entity_name
ApplicationDefinition Entity

8.2.3.15 DeleteApplicationDefinition
This method is the binding for the WMDeleteEntity function, when used to delete a workflow application definition that is not part of a process definition.

DeleteApplicationDefinition {
    in ApplicationDefinition AppDef
}

Argument Description (WMDeleteEntity Argument)
AppDef entity_id

8.2.3.16 CreateParticipantDefinition
This method is the binding for the WMCreateEntity function, when used to create a workflow participant definition outside of any process definition.

ParticipantDefinition CreateParticipantDefinition ()

Argument Description (WMCreateEntity Argument)
Name entity_name
ParticipantDefinition Entity
8.2.3.17 DeleteParticipantDefinition

This method is the binding for the WMDeleteEntity function, when used to delete a workflow participant definition that is not part of a process definition.

```
DeleteParticipantDefinition (in ParticipantDefinition PartDef)
```

Argument Description (WMDeleteEntity Argument)

PartDef entity_id

8.2.4 Filter

The filter object class corresponds to the WMTFilter data type. Filter objects are externally creatable.

8.2.4.1 Properties

Filter objects have the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
<td>WMTFilter.filter_type</td>
</tr>
<tr>
<td>Length</td>
<td>Integer</td>
<td>WMTFilter.filter_length</td>
</tr>
<tr>
<td>AttributeName</td>
<td>String</td>
<td>WMTFilter.attribute_name</td>
</tr>
<tr>
<td>Comparison</td>
<td>Integer</td>
<td>WMTFilter.comparison</td>
</tr>
<tr>
<td>FilterString</td>
<td>String</td>
<td>WMTFilter.filter_string</td>
</tr>
</tbody>
</table>

8.2.4.2 Methods

There are no methods for Filter objects.

8.2.5 Process Definition

The process definition class corresponds to the WMTProcDefID data type. Process definition objects are not externally creatable. They are returned by the Server object’s ListProcessDefinitions method, and by the ProcessDefinition property of a ProcessInstance object.

8.2.5.1 Properties

A ProcessDefinition object has the following read-only properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td>Collection</td>
<td>WPDL &lt;Activity List&gt;</td>
</tr>
<tr>
<td>Applications</td>
<td>Collection</td>
<td>WPDL &lt;Workflow Application List&gt;</td>
</tr>
<tr>
<td>Data</td>
<td>Collection</td>
<td>WPDL &lt;Workflow Process Relevant Data List&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WMTProcDefId.proc_def_id</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WPDL &lt;process name&gt;</td>
</tr>
<tr>
<td>Participants</td>
<td>Collection</td>
<td>WPDL &lt;Workflow Participant List&gt;</td>
</tr>
<tr>
<td>States</td>
<td>Collection</td>
<td>WMOpenProcessDefinitionStatesList</td>
</tr>
<tr>
<td>Transitions</td>
<td>Collection</td>
<td>WPDL &lt;Transition Information List&gt;</td>
</tr>
</tbody>
</table>

All of these properties, except name, are read-only. The States property takes a Filter parameter.

8.2.5.2 Methods

A ProcessDefinition object has the following methods:
| ProcessInstance CreateInstance | WMCreateProcessInstance |
| ChangeInstancesState in Filter InstanceFilter in String InstanceState | WMChangeProcessInstancesState |
| ChangeActivityInstancesState in String ActivityDefinitionID in Filter InstanceFilter in String InstanceState | WMChangeActivityInstancesState |
| TerminateInstances in Filter InstanceFilter | WMTerminateProcessInstances |
| AssignInstancesAttribute in Filter InstanceFilter in String Name in Variant Value | WMAssignProcessInstancesAttribute |
| AssignActivityInstancesAttribute in String ActivityInstanceID in Filter InstanceFilter in String Name in String Value | WMAssignActivityInstancesAttribute |
| AbortInstances in Filter InstanceFilter | WMAbortProcessInstances |
| ActivityDefinition AddActivity in String Name | WMAddEntity |
| ApplicationDefinition AddApplication in String Name | WMAddEntity |
| ParticipantDefinition AddParticipant in String Name | WMAddEntity |
| ProcessDataDefinition AddData in String Name | WMAddEntity |
| TransitionDefinition AddTransition in String Name | WMAddEntity |

Note that the Server parameters to these methods is implicit. They use the server from which the process definition was obtained.

### 8.2.5.3 CreateInstance

This method is the binding for the WMCreateProcessInstance function.

```plaintext
ProcessInstance CreateInstance ()
```

**Argument** | **Description (WMCreateProcessInstance Argument)**
--- | ---
ProcessInstance | pproc_inst_id

### 8.2.5.4 ChangeInstancesState

This method is the binding for the WMChangeProcessInstancesState function.

```plaintext
ChangeInstancesState ( in Filter InstanceFilter, in String InstanceState)
```

**Argument** | **Description (WMChangeProcessInstancesState Argument)**
--- | ---
InstanceFilter | pproc_inst_filter
InstanceState | process_inst_state
8.2.5.5 ChangeActivityInstancesState
This method is the binding for the WMChangeActivityInstancesState function.

\[
\text{ChangeActivityInstancesState (}\]
\begin{align*}
in & \text{ ActivityDefID}, \\
\text{in} & \text{ InstanceFilter}, \\
\text{in} & \text{ InstanceState}\end{align*}
\]

Argument Description (WMChangeActivityInstancesState Argument)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivityDefID</td>
<td>pactivity_def_id</td>
</tr>
<tr>
<td>InstanceFilter</td>
<td>pact_inst_filter</td>
</tr>
<tr>
<td>InstanceState</td>
<td>activity_inst_state</td>
</tr>
</tbody>
</table>

8.2.5.6 TerminateInstances
This method is the binding for the WMTerminateProcessInstances function.

\[
\text{TerminateInstances (}\]
\begin{align*}
in & \text{ InstanceFilter}\end{align*}
\]

Argument Description (WMTerminateProcessInstances Argument)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceFilter</td>
<td>pproc_inst_filter</td>
</tr>
</tbody>
</table>

8.2.5.7 AssignInstancesAttribute
This method is the binding for the WMAssignProcessInstancesAttribute function.

\[
\text{AssignInstancesAttribute (}\]
\begin{align*}
in & \text{ InstanceFilter,} \\
\text{in} & \text{ Name,} \\
\text{in} & \text{ Value}\end{align*}
\]

Argument Description (WMAssignProcessInstancesAttribute Argument)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceFilter</td>
<td>pact_inst_filter</td>
</tr>
<tr>
<td>Name</td>
<td>attribute_name</td>
</tr>
<tr>
<td>Value</td>
<td>pattribute_value</td>
</tr>
</tbody>
</table>

8.2.5.8 AssignActivityInstancesAttribute
This method is the binding for the WMAssignActivityInstancesAttribute function.

\[
\text{AssignActivityInstancesAttribute (}\]
\begin{align*}
in & \text{ ActivityDefID,} \\
\text{in} & \text{ InstanceFilter,} \\
\text{in} & \text{ Name,} \\
\text{in} & \text{ Value}\end{align*}
\]

Argument Description (WMAssignActivityInstancesAttribute Argument)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivityDefID</td>
<td>pactivity_def_id</td>
</tr>
<tr>
<td>InstanceFilter</td>
<td>pact_inst_filter</td>
</tr>
<tr>
<td>Name</td>
<td>attribute_name</td>
</tr>
<tr>
<td>Value</td>
<td>pattribute_value</td>
</tr>
</tbody>
</table>

8.2.5.9 AbortInstances
This method is the binding for the WMAbortProcessInstances function.

\[
\text{AbortInstances (}\]
\begin{align*}
in & \text{ InstanceFilter}\end{align*}
\]

Version 2.0
Copyright © 1993, 1999, The Workflow Management Coalition
Argument Description (WMAbortProcessInstances Argument)
InstanceFilter pproc_inst_filter

8.2.5.10 AddActivity
This method is the binding for the WMAddEntity function, when used to add an activity definition to a process definition.

ActivityDefinition AddActivity {
    in String Name
}

Argument Description (WMAddEntity Argument)
Name entity_name
ActivityDefinition entity

8.2.5.11 AddApplication
This method is the binding for the WMAddEntity function, when used to add an application definition to a process definition.

ApplicationDefinition AddApplication {
    in String Name
}

Argument Description (WMAddEntity Argument)
Name entity_name
ApplicationDefinition entity

8.2.5.12 AddData
This method is the binding for the WMAddEntity function, when used to add process relevant data to a process definition.

ProcessDataDefinition AddData {
    in String Name
}

Argument Description (WMAddEntity Argument)
Name entity_name
ProcessDataDefinition entity

8.2.5.13 AddParticipant
This method is the binding for the WMAddEntity function, when used to add a participant to a process definition.

ParticipantDefinition AddParticipant {
    in String Name
}

Argument Description (WMAddEntity Argument)
Name entity_name
ParticipantDefinition entity
8.2.5.14 AddTransition

This method is the binding for the WMAddEntity function, when used to add transition information to a process definition.

```
TransitionDefinition AddTransition (in String Name)
```

Argument Description (WMAddEntity Argument)
Name entity_name
TransitionDefinition entity

8.2.6 Process Instance

The ProcessInstance object class corresponds to the WMTProcessInst data type. Process instance objects are not externally creatable. They are returned by the Server object’s ListProcessInstances method, and by the ProcessInstance property of an ActivityInstance or WorkItem object.

8.2.6.1 Properties

A ProcessInstance object has the following read-only properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenProcessInstanceAttributesList</td>
</tr>
<tr>
<td>DataReference</td>
<td>String</td>
<td>WMTProcessInst.data_reference</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WMTProcessInst.proc_inst_id</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WMTProcessInst.process_name</td>
</tr>
<tr>
<td>Participants</td>
<td>Collection</td>
<td>WMTProcessInst.proc_participants</td>
</tr>
<tr>
<td>Priority</td>
<td>Integer</td>
<td>WMTProcessInst.priority</td>
</tr>
<tr>
<td>ProcessDefinition</td>
<td>ProcessDefinition</td>
<td>WMGetProcessDefinition</td>
</tr>
<tr>
<td>ProcessDefinitionID</td>
<td>String</td>
<td>WMTProcessInst.proc_def_id</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
<td>WMTProcessInst.state</td>
</tr>
<tr>
<td>States</td>
<td>Collection</td>
<td>WMOpenProcessInstanceStatesList</td>
</tr>
</tbody>
</table>

All of these properties are read-only, except for the State property. Updating this property has the same effect as calling the ChangeState method. The Attributes and States properties take a Filter parameter. The ProcessDefinition property is a convenience. It calls the GetProcessDefinition method on the session from which the process instance was obtained, passing the ProcessDefinitionID property value.

8.2.6.2 Methods

A ProcessInstance object has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessInstance Start</td>
<td>WMStartProcess</td>
</tr>
<tr>
<td>Terminate</td>
<td>WMTerminateProcess</td>
</tr>
<tr>
<td>ChangeState in String State</td>
<td>WMChangeProcessInstanceState</td>
</tr>
<tr>
<td>AssignAttribute in String Name in Variant Value</td>
<td>WMAssignProcessInstanceAttribute</td>
</tr>
<tr>
<td>Abort</td>
<td>WMAbortProcessInstance</td>
</tr>
</tbody>
</table>
Note that the Server parameters to these methods is implicit. They use the server from which the process instance was obtained.

### 8.2.6.3 Start
This method is the binding for the WMStartProcess function.

```plaintext
ProcessInstance Start ()
```

**Argument** | **Description (WMStartProcess Argument)**
--- | ---
ProcessInstance | pnew_proc_inst_id

### 8.2.6.4 Terminate
This method is the binding for the WMTerminateProcessInstance function.

```plaintext
Terminate ()
```

### 8.2.6.5 ChangeState
This method is the binding for the WMChangeProcessInstanceState function.

```plaintext
ChangeState ( in String State)
```

**Argument** | **Description (WMChangeProcessInstanceState Argument)**
--- | ---
State | pproc_inst_state

### 8.2.6.6 AssignAttribute
This method is the binding for the WMAssignProcessInstanceAttribute function.

```plaintext
AssignAttribute ( in String Name, in Variant Value)
```

**Argument** | **Description (WMAssignProcessInstanceAttribute Argument)**
--- | ---
Name | attribute_name
Value | pattribute_value

### 8.2.6.7 Abort
This method is the binding for the WMAbortProcessInstance function.

```plaintext
Abort ()
```

### 8.2.7 Activity Definition
The ActivityDefinition class corresponds to the Workflow Process Activity entity in WPDL. Activity definition objects are not externally creatable. They are returned in the Activities property of a ProcessDefinition object.

#### 8.2.7.1 Properties
An ActivityDefinition object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenEntityAttributesList</td>
</tr>
</tbody>
</table>
8.2.8 Activity Instance

The ActivityInstance class corresponds to the WMTActivityInst data type. Activity instance objects are not externally creatable. They are returned by the Server object’s ListActivityInstances method, and by the ActivityInstance property of a WorkItem object.

8.2.8.1 Properties

An ActivityInstance object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenActivityInstanceAttributesList</td>
</tr>
<tr>
<td>DataReference</td>
<td>String</td>
<td>WMTActivityInst.data_reference</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WMTActivityInst.activity_inst_id</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WMTActivityInst.activity_name</td>
</tr>
<tr>
<td>Participants</td>
<td>Collection</td>
<td>WMTActivityInst.proc_participants</td>
</tr>
<tr>
<td>Priority</td>
<td>Integer</td>
<td>WMTActivityInstance.priority</td>
</tr>
<tr>
<td>ProcessInstance</td>
<td>ProcessInstance</td>
<td>WMGetProcessInstance</td>
</tr>
<tr>
<td>ProcessInstanceID</td>
<td>String</td>
<td>WMTActivityInstance.proc_inst_id</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
<td>WMTActivityInstance.state</td>
</tr>
<tr>
<td>States</td>
<td>Collection</td>
<td>WMOpenActivityInstanceStatesList</td>
</tr>
</tbody>
</table>

All of these properties are read-only, except for the State property. Updating this property has the same effect as calling the ChangeState method. The Attributes and States properties take a Filter parameter. The ProcessInstance property is a convenience. It calls the GetProcessInstance method on the server from which the activity instance was obtained, passing the ProcessInstanceID property value.

8.2.8.2 Methods

An ActivityInstance object has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChangeState in String State</td>
<td>WMChangeActivityInstanceState</td>
</tr>
<tr>
<td>AssignAttribute in String Name in Variant Value</td>
<td>WMAssignActivityInstanceAttribute</td>
</tr>
</tbody>
</table>

Note that the Server parameters to these methods is implicit. They use the server from which the activity instance was obtained.

8.2.8.3 ChangeState

This method is the binding for the WMChangeActivityInstanceState function.

```
ChangeState ( in String State)
```
Argument Description (WMChangeActivityInstanceState Argument)
State activity_inst_state

8.2.8.4 AssignAttribute
This method is the binding for the WMAssignActivityInstanceAttribute function.

\[
\text{AssignAttribute} \ (\text{in String Name, in Variant Value})
\]

Argument Description (WMAssignActivityInstanceAttribute Argument)
Name attribute_name
Value pattribute_value

8.2.9 WorkItem
The WorkItem class corresponds to the WMTWorkItem data type. Work item objects are not externally creatable. They are returned by the Server object’s ListWorkItem method.

8.2.9.1 Properties
A WorkItem object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivityInstance</td>
<td>ActivityInstance</td>
<td>WMGetActivityInstance</td>
</tr>
<tr>
<td>ActivityInstanceID</td>
<td>String</td>
<td>WMTWorkItem.activity_inst</td>
</tr>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenWorkItemAttributesList</td>
</tr>
<tr>
<td>DataReference</td>
<td>String</td>
<td>WMTWorkItem.data_reference</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WMTWorkItem.workitem_id</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WMTWorkItem.workitem_name</td>
</tr>
<tr>
<td>Participant</td>
<td>String</td>
<td>WMTWorkItem.proc_participant</td>
</tr>
<tr>
<td>Priority</td>
<td>Integer</td>
<td>WMTWorkItemance.priority</td>
</tr>
<tr>
<td>ProcessInstance</td>
<td>ProcessInstance</td>
<td>WMGetProcessInstance</td>
</tr>
<tr>
<td>ProcessInstanceID</td>
<td>String</td>
<td>WMTWorkItemance.proc_inst_id</td>
</tr>
</tbody>
</table>

All of these properties are read-only. The Attributes property takes a Filter parameter. The ActivityInstance and ProcessInstance properties are a convenience. They call the GetProcessInstance and getActivityInstance methods, respectively, on the server from which the work item was obtained, passing the ProcessInstanceID or ActivityInstanceID property value.

8.2.9.2 Methods
A WorkItem object has the following methods:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignAttribute</td>
<td>WMAssignWorkItemAttribute</td>
</tr>
<tr>
<td>Complete</td>
<td>WMCompleteWorkItem</td>
</tr>
<tr>
<td>Reassign</td>
<td>WMReassignWorkItem</td>
</tr>
</tbody>
</table>

Note that the Server parameters to these methods is implicit. They use the server from which the work item was obtained.
### 8.2.9.3 AssignAttribute

This method is the binding for the `WMAssignWorkItemAttribute` function.

```plaintext
AssignAttribute (  
  in String Name,
  in Variant Value)
```

**Argument Description (WMAssignWorkItemAttribute Argument)**

- Name: attribute_name
- Value: attribute_value

### 8.2.9.4 Complete

This method is the binding for the `WMCompleteWorkItem` function.

```plaintext
Complete ()
```

### 8.2.9.5 Reassign

This method is the binding for the `WMReassignWorkItem` function.

```plaintext
Reassign (  
  in String SourceUser,
  in String TargetUser)
```

**Argument Description (WMReassignWorkItem Argument)**

- SourceUser: psource_user
- TargetUser: ptarget_user

### 8.2.10 Transition Definition

The TransitionDefinition class corresponds to the Transition Information entity in WPDL. Transition definition objects are not externally creatable. They are returned in the Transitions property of a ProcessDefinition object.

#### 8.2.10.1 Properties

A TransitionDefinition object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection WMOpenEntityAttributesList</td>
<td></td>
</tr>
<tr>
<td>From</td>
<td>ActivityDefinition WPDL &lt;trans from&gt;</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>String WPDL &lt;transition id&gt;</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>String WPDL &lt;name&gt;</td>
<td></td>
</tr>
<tr>
<td>To</td>
<td>ActivityDefinition WPDL &lt;trans to&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### 8.2.11 Participant Definition

The ParticipantDefinition class corresponds to the Workflow Participant Definition entity in WPDL. Participant definition objects are not externally creatable. They are returned by the ListParticipantDefinitions method of a Server object, or in the Participants property of a ProcessDefinition object.
8.2.11.1 Properties
A ParticipantDefinition object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenEntityAttributesList</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WPDL &lt;participant id&gt;</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WPDL &lt;name&gt;</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>WPDL &lt;participant type&gt;</td>
</tr>
</tbody>
</table>

8.2.12 Application Definition
The ApplicationDefinition class corresponds to the Workflow Application Definition entity in WPDL. Application definition objects are not externally creatable. They are returned by the ListApplicationDefinitions method of a Server object, or in the Applications property of a ProcessDefinition object.

8.2.12.1 Properties
An ApplicationDefinition object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenEntityAttributesList</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WPDL &lt;tool name&gt;</td>
</tr>
</tbody>
</table>

8.2.13 Process Data Definition
The ProcessDataDefinition class corresponds to the Workflow Process Relevant Data entity in WPDL. Process data definition objects are not externally creatable. They are returned in the Data property of a ProcessDefinition object.

8.2.13.1 Properties
A ProcessDataDefinition object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Collection</td>
<td>WMOpenEntityAttributesList</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>WPDL &lt;data id&gt;</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>WPDL &lt;name&gt;</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>WPDL &lt;data type&gt;</td>
</tr>
</tbody>
</table>

8.2.14 Attribute
The Attribute object class corresponds to a single attribute of a workflow object. Attribute objects are not externally creatable. They are returned in the Attributes property of a workflow object, which is a collection of attribute objects indexed by name.

8.2.14.1 Properties
An Attribute object has the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>WMFetch...Attribute Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataType</td>
<td>Integer</td>
<td>attribute_type</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>attribute_name</td>
</tr>
</tbody>
</table>
8.3 OMG IDL Binding

This chapter provides a detailed description of the Workflow Facility Client Application components in terms of OMG IDL. The specification is split into three modules, the first one providing generic interfaces and operations, the others defining the specific interfaces and functions for the Application Client Interface and the Process Definition Interface.

8.3.1 The Workflow Facility Base Module

The Workflow Facility Base module contains definitions common to all of the various interfaces described in the Workflow Reference Model.

The interfaces defined by this module are:
- Attribute interface, which provides access to attributes of various types of workflow objects.
- AttributeList interface, which provides operations to handle filtered lists of Attributes.
- Filter interface, which is used to define queries for workflow objects issued against the Workflow Enactment Service who owns these objects.
- WorkflowObject interface, which defines generic operations and attributes common to many workflow objects.

The following abbreviated IDL summarizes the interfaces contained in the CfWFBase module.

```idl
module CfWorkflowFacilityBase {
    ... // data type and general exception definitions
    interface Filter {
        ... // query filter object definitions
    };
    interface Attribute {
        ... // workflow object attribute definitions
    };
    interface AttributeList {
        ... // workflow object attribute list definitions
    };
    interface WorkflowObject {
        ... // workflow object definitions
    };
}
```

The following sections describe the contents of the CfWFBase module in detail.

8.3.1.1 Data Types and General Exceptions

The following data types and exceptions are defined in the CfWFBase module and are used in various interfaces of the Workflow Facility.
Each workflow object has a name and a state. WMTName and WMTState define the types to specify the corresponding attributes. In addition, the WMTName type is used to define any kind of name-like attribute, e.g., to specify named references to objects outside the scope of the Workflow Facility specification. WMTStates handles a list of states. The WMTId type is used for identification of persistent object references. The WMTErrorCode type is used to provide additional information with some Exceptions, e.g., the InvalidFilter exception uses the Error Code to indicate the specific problem with the Filter.

The InvalidFilter and NoMoreData exceptions are related to processing of filtered queries and query result lists. InvalidState and TransitionNotAllowed exceptions are raised by state-changing operations on workflow objects.

### 8.3.1.2 Filter Interface

The Filter interface is used to specify the filter criteria for a query against the set of objects of a specific type.

```java
interface Filter {
    attribute long filterType;
    attribute long filterLength;
    attribute WMTName attributeName;
    attribute integer comparison;
    attribute string filterString;
}
```

### 8.3.1.3 Attribute Interfaces

The Attribute interface is used to access attributes of workflow objects. Attribute data are accessed by value; the attributeValue is of type any and is further specified by attributeType.

There are basically three types of attributes of a workflow object that can be accessed via this interface:

- The standard attributes described in this document (e.g., Name)
- Vendor specific attributes associated with a particular object type (e.g., ‘ProcessClass’ attribute of a ProcessDefinition)
- User defined attributes associated with particular object instances (e.g., ‘CustomerNumber’ attribute of a particular ProcessInstance)

```java
interface Attribute {
    attribute WMTName attributeName;
    attribute string attributeType;
    attribute long attributeLength;
    attribute any attributeValue;
}
```

typedef sequence<Attribute> Attributes;

The AttributeList interface provides iterator operations for handling of a list of Attributes; the corresponding factory operation for this interface can be found in an workflow object interface. The
fetchAttribute operation gets the next Attribute from the list, the fetchAttributes operation gets the next howMany Attributes from the list; if the list is empty, the NoMoreData exception is raised.

```java
interface AttributeList {
    Attribute fetchAttribute()
        raises (NoMoreData);
    Attributes fetchAttributes(long howMany)
        raises (NoMoreData);
}
```

### 8.3.1.4 Workflow Object Interface

The Workflow Object interface defines the attributes and operations common to most workflow objects. Each WorkflowObject has a Name, a State and a set of Attributes associated with it. A list of valid states for a particular WorkflowObject can be obtained using the listValidStates operation; the InvalidState exception is raised when a state change to an unknown state is requested. getState obtains the current State of a workflow object and setState changes the State; the TransitionNotAllowed exception is raised when the transition from the current state to the new state is not allowed. OpenAttributesList is the factory operation for an AttributeList, allowing for a query for attributes; getAttributeValue supports access to attributes by name. The AssignAttribute(s) operations assign new values to Attributes. The InvalidAttribute exception is raised on requests for attributes not defined for the workflow object; the AttributeAssignmentFailed exception is raised when the Attribute could not be modified, e.g., is read-only.

```java
interface WorkflowObject {
    attribute WMTName name;
    attribute WMId id;

    void listValidStates (
        in Filter filter,
        in boolean countFlag,
        out WMTStates states,
        out long count);

    void changeState (in WMTState newState)
        raises (TransitionNotAllowed, InvalidState);

    void getState ( out WMTState currentState);

    void openAttributeList (in Filter filter,
        in boolean countFlag,
        out AttributeList attributes,
        out long count)
        raises (InvalidFilter);

    void getAttributeValue (in WMTName name
        out Attribute attribute)
        raises (InvalidAttribute);

    void assignAttribute (in Attribute attribute)
        raises (InvalidAttribute, AttributeAssignmentFailed);

    void assignAttributes (in Attributes attributes)
        raises (InvalidAttribute, AttributeAssignmentFailed);
}
```

### 8.3.2 Workflow Application Client Server Interface
The Workflow ApplicationClientServer interface handles the connection of a particular workflow user to an
Enactment Service and provides access to the workflow objects accessible through this Enactment Service.
The connect operation initializes the ApplicationClientServer; the context of the connection is defined by
the engineName and the scope. The ConnectFailed exception is raised when a connection could not be
established. The disconnect operation serves as a destructor for the ApplicationClientServer.
Access to the workflow objects accessible through the connection is supported by providing factory
methods for interfaces managing access to lists of workflow objects: the Open...sList operation take a filter
as their first argument (see the CfWFBase module description for details), the countFlag parameter
indicates whether the number of elements in the query result should be returned. A NotConnected exception
is raised when no connection was established. The query results are bound to a connection and are
invalidated when the connection is terminated.
Operations are provided to get a ProcessInstance, ActivityInstance or WorkItem object via its identifier.

```java
interface ApplicationClientServer {
    attribute CfWFBase :: WMTName engineName;
    attribute CfWFBase :: WMTName scope;

    void connect(
        in CfWFBase :: WMTName userId,
        in string password)
        raises (ConnectFailed);

    void disconnect()
        raises (NotConnected);

    ProcessDefinitionList openProcessDefinitionsList(
        in CfWFBase :: Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);

    ProcessInstanceList openProcessInstancesList(
        in CfWFBase :: Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);

    ActivityInstanceList openActivityInstancesList(
        in CfWFBase :: Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);

    WorkList openWorkList(
        in CfWFBase :: Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);

    ProcessInstance getProcessInstance(
        in CfWFBase :: WMTId processInstanceId)
        raises (InvalidId);

    ActivityInstance getActivityInstance(
        in CfWFBase :: WMTId processInstanceId,
        in CfWFBase :: WMTId activityInstanceId)
        raises (InvalidId);

    WorkItem getWorkItem(
        in CfWFBase :: WMTId processInstanceId,
        in CfWFBase :: WMTId workItemId)
        raises (InvalidId);
};
```
8.3.2.1 Process Definition Interface

The Process Definition interface provides factory operation for Process Instances and supports Process Management operations on workflow objects related to the Process Definition: change of State and change of a specific Attribute's value for all members of a filtered set of Process Instances and Activity Instances. The ProcessDefinition interface inherits attributes and operations from WorkflowObject.

```
interface ProcessDefinition : CfWFBase :: WorkflowObject {
    ProcessInstance createProcessInstance (in CfWFBase :: WMTName instanceName)
    raises (NotConnected);

    void changeProcessInstancesState (in CfWFBase :: Filter filter,
                                       in CfWFBase :: WMTState newState)
    raises (NotConnected, TransitionNotAllowed, InvalidState);

    void abortProcessInstances (in CfWFBase :: Filter filter)
    raises (NotConnected, TransitionNotAllowed);

    void terminateProcessInstances (in CfWFBase :: Filter filter)
    raises (NotConnected, TransitionNotAllowed);

    void assignProcessInstancesAttribute (in CfWFBase :: Filter filter,
                                         in CfWFBase :: Attribute attribute)
    raises (NotConnected, InvalidFilter, InvalidAttribute, AttributeAssignmentFailed);

    void changeActivityInstancesState (in CfWFBase :: Filter filter,
                                        in CfWFBase :: WMTState newState)
    raises (NotConnected, InvalidFilter, TransitionNotAllowed, InvalidState);

    void assignActivityInstancesAttribute (in CfWFBase :: Filter filter,
                                             in CfWFBase :: Attribute attribute)
    raises (NotConnected, InvalidFilter, InvalidAttribute, AttributeAssignmentFailed);
}
```

8.3.2.2 Process Instance Interface

The ProcessInstance interface provides operations to access and modify the state and the attributes of a Process Instance object.

State changes can be performed using the start, terminate or abort operations. Additional state transitions may be supported by an EnactmentService (see the WorkflowObject :: changeState() operation described above). The getParentProcessDefinition operation returns the ProcessDefinition object that was used to create the specific ProcessInstance. The listAssignedParticipants operation provides the list of workflow Participants associated to the Process Instance. The ProcessInstance interface inherits attributes and operations from WorkflowObject. All operations require an active connection to the Enactment Service.

```
interface ProcessInstance : CfWFBase :: WorkflowObject {
    attribute CfWFBase :: WMTDataRef dataReference;
    attribute long priority;
}
```
8.3.2.3 Activity Instance Interface

The Activity Instance interface provides operations to access and modify the attributes and the state of an ActivityInstance object.

The getParentProcessInstance operation returns the ProcessInstance object that owns the specific ActivityInstance. The listAssignedParticipants operation provides the list of workflow Participants associated to the Activity Instance. The ActivityInstance interface inherits attributes and operations from WorkflowObject. All operations require an active connection to the Enactment Service.

```java
interface ActivityInstance : CfWFBase::WorkflowObject {
    attribute CfWFBase::WMTDataRef dataReference;
    attribute long priority;
    ProcessInstance getParentProcessInstance();
    CfWFBase::WMTWflParticipants listAssignedParticipants() raises (NotConnected);
};
```

8.3.2.4 Work Item Interface

The WorkItem interface provides operations to access and modify the attributes and the state of a WorkItem object.

The get- and completeWorkitem operations change the State of a WorkItem. getAssignedParticipant returns the workflow participant currently assigned to the work item; reassignWorkItem assigns it to another participant.

```java
interface WorkItem : CfWFBase::WorkflowObject {
    attribute CfWFBase::WMTDataRef dataReference;
    attribute long priority;
    ProcessInstance getParentProcessInstance();
    ActivityInstance getParentActivityInstance();
    void reassign(in CfWFBase::WMTWflParticipant sourceUser, in CfWFBase::WMTWflParticipant targetUser) raises (NotConnected, InvalidSourceUser, InvalidTargetUser);
    void get() raises (NotConnected, TransitionNotAllowed);
    void complete();
};
```
8.3.2.5 Filtered List Processing

The following interfaces provide iterators for results returned from filtered list requests; see the section on Attributes for a description of the iterator functions.

```c
typedef sequence<ProcessInstance> ProcessInstances;
typedef sequence<ActivityInstance> ActivityInstances;
typedef sequence<WorkItem> WorkItems;

interface ProcessDefinitionList {
    attribute long count;
    ProcessDefinition fetchProcessDefinition();
    raises (NoMoreData);
    ProcessDefinitions fetchProcessDefinitions(
        in  unsigned long howMany)
    raises (NoMoreData);
};

interface ProcessInstanceList {
    attribute long count;
    ProcessInstance fetchProcessInstance();
    raises (NoMoreData);
    ProcessInstances fetchProcessInstances(
        in  unsigned long howMany)
    raises (NoMoreData);
};

interface ActivityInstanceList {
    attribute long count;
    ActivityInstance fetchActivityInstance();
    raises (NoMoreData);
    ActivityInstances fetchActivityInstances(
        in  unsigned long howMany)
    raises (NoMoreData);
};

interface WorkList {
    attribute long count;
    WorkItem fetchWorkItem();
    raises (NoMoreData);
    WorkItem fetchWorkItems(
        in  unsigned long howMany)
    raises (NoMoreData);
};
```

8.3.3 The Process Definition Module

The Process Definition Module contains the interfaces used to create and modify Process Definitions to be executed by an Enactement Service.

The module defines the following interfaces:
• **ProcessModelServer** interface, which handles connection of a workflow participant with a particular Enactment Service and provides factory interfaces for access to filtered list of workflow definition objects owned by that Enactment Service.

• **ProcessModel** interface, which represents a workflow model; this interface serves as a factory for components of the process model, such as ActivityDefinitions and TransitionDefinitions.

• **ActivityDefinition** interface, which represents a node in a process model

• **TransitionDefinition** interface, which represents a connection between ActivityDefinitions

• **DataDefinition** interface, which defines the Process Relevant Data used by a particular process model

• **ApplicationDefinition** interface, which represents an application that can be used to support processing of an Activity during execution of a process model

• **ParticipantDefinition** interface, which represents a resource that might receive Work Items during execution of a process model

The following abbreviated IDL summarizes the interfaces contained in the CfWFBase module.

```idl
#include "CfWFBase.idl"

module CfWFProcessDefinition {
    ... // Data type and specific exception definitions
    interface ProcessModel; // Forward declaration
    interface ApplicationDefinition; // Forward declaration
    interface ParticipantDefinition; // Forward declaration

    interface ProcessModelList {
        ... // Iterator for process model query result
    };

    interface ApplicationDefinitionList {
        ... // Iterator for application definition query result
    };

    interface ParticipantList {
        ... // Iterator for participant definition query result
    };

    interface ProcessDefinitionServer {
        ... // process definition server object definitions
    };

    interface ProcessModel : CfWFBase::WorkflowObject{
        ... // process model object definitions
    };

    interface ApplicationDefinition : CfWFBase::WorkflowObject {
        ... // application definition object definitions
    };

    interface ParticipantDefinition : CfWFBase::WorkflowObject {
        ... // participant definition object definitions
    };

};
```

### 8.3.3.1 Data Types and Specific Exceptions

The following data types and exceptions are specific to the Process Definition Client module.
The exceptions defined here deal with problems related to management of the connection to the Enactment Service.

### 8.3.3.2 Process Definition Server Interface

The Process Definition Server Interface handles the connection of a particular workflow user to an Enactment Service and provides access to the workflow definition objects accessible through this Enactment Service.

The connect operation initializes the WorkflowEnactmentServer; the context of the connection is defined by the engineName and the scope. The ConnectFailed exception is raised when a connection could not be established. The disconnect operation serves as a destructor for the ApplicationClientServer.

Access to the workflow objects accessible through the connection is supported by providing factory methods for interfaces managing access to lists of workflow objects: the Open...List operation take a filter as their first argument (see the CfWFBase module description for details), the countFlag parameter indicates whether the number of elements in the query result should be returned. A NotConnected exception is raised when no connection was established. The query results are bound to a connection and are invalidated when the connection is terminated.

```plaintext
interface ProcessDefinitionServer {
    attribute CfWFBase::WMTName engineName;
    attribute CfWFBase::WMTName scope;
    void connect(
        in CfWFBase::WMTName userId,
        in string password)
        raises (ConnectFailed);
    void disconnect()
        raises (NotConnected);
    ProcessModel createProcessModel(
        in CfWFBase::WMTName processName)
        raises (NotConnected);
    ProcessModelList openProcessModelsList(
        in CfWFBase::Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);
    ApplicationDefinitionList openApplicationDefinitionsList (n
        in CfWFBase::Filter filter,
        in boolean countFlag)
        raises (InvalidFilter, NotConnected);
    ParticipantDefinitionList openParticipantDefinitionsList (n
        in CfWFBase::Filter filter,
        in boolean countFlag,
        out WMTActivityInstanceList activityInstances,
        out long count)
        raises (InvalidFilter, NotConnected);
    ProcessModel getProcessModel(
        in CfWFBase::WMTId processModelId)
        raises (InvalidId);
}
```
8.3.3.3 Process Model Interface
The Process Model interface provides factory operation for Activity Definitions, Transition Definitions and Data Definitions contained in a Process Model.
The ProcessDefinition interface inherits attributes and operations from WorkflowObject.

```csharp
interface ProcessModel : CfWFBase::WorkflowObject {
    ActivityDefinition addActivityDefinition(
        in CfWFBase::WMTName activityName)
    raises (NotConnected);
    void removeActivityDefinition(
        in CfWFBase::WMTId activityId)
    raises (NotConnected);
    DataDefinition addDataDefinition(
        in CfWFBase::WMTName dataName,
        in CfWFBase::WMTType dataType)
    raises (NotConnected);
    void removeDataDefinition(
        in CfWFBase::WMTId dataId)
    raises (NotConnected);
    TransitionDefinition addTransitionDefinition(
        in CfWFBase::WMTName transitionName,
        in CfWFBase::WMTId sourceActivityDefinitionId,
        in CfWFBase::WMTId targetActivityDefinitionId)
    raises (NotConnected, InvalidId);
    void removeTransitionDefinition(
        in CfWFBase::WMTId transitionId)
    raises (NotConnected);
};
```

8.3.3.4 Application Definition Interface
The ApplicationDefinition interface provides operations to access and modify the attributes of an Application Definition object. All operations require an active connection to the Enactment Service.

```csharp
interface ApplicationDefinition : CfWFBase::WorkflowObject {
};
```

8.3.3.5 Participant Definition Interface
The ParticipantDefinition interface provides operations to access and modify the attributes of a Participant Definition object. All operations require an active connection to the Enactment Service.
8.3.3.6 **Activity Definition Interface**

The `ActivityDefinition` interface provides operations to access and modify the attributes of a `ActivityDefinition` object.

The `getParentProcessModel` operation returns the `ProcessModel` object that was used to create the specific `ActivityDefinition`. All operations require an active connection to the Enactment Service.

```java
interface ActivityDefinition : CfWFBase::WorkflowObject{
    attribute WMTImplementationType implementationType;
    attribute CfWFBase::WMTId implementationId;
    ProcessModel getParentProcessModel();
};
```

8.3.3.7 **Transition Definition Interface**

The `TransitionDefinition` interface provides operations to access and modify the attributes of a `TransitionDefinition` object.

The `getParentProcessModel` operation returns the `ProcessModel` object that was used to create the specific `TransitionDefinition`. All operations require an active connection to the Enactment Service.

```java
interface TransitionDefinition : CfWFBase::WorkflowObject {
    attribute CfWFBase::WMTId sourceActivityId;
    attribute CfWFBase::WMTId targetActivityId;
    ProcessModel getParentProcessModel();
};
```

8.3.3.8 **Filtered List Processing**

The following interfaces provide iterators for results returned from filtered list requests; see the section on Attributes for a description of the iterator functions.

```java
typedef sequence<ProcessModel> ProcessModels;
typedef sequence<ActivityDefinition> ActivityDefinitions;
typedef sequence<TransitionDefinition> TransitionDefinitions;
typedef sequence<ApplicationDefinition> ApplicationDefinitions;
typedef sequence<ParticipantDefinition> ParticipantDefinitions;

interface ProcessModelsList {
    attribute long count;
    ProcessModel fetch();
    raises (NoMoreData);
    ProcessModels fetchN(
in  unsigned long howMany);
    raises (NoMoreData);
};
```
8.3.4 Relationship to WfMC Standards

The C-language description has been converted into an object oriented specification. Where possible, the syntax of C-functions has been preserved when converting to operations on objects. Here is a list of changes:

- The operations dealing with States and Attributes of workflow objects have been moved into the WorkflowObject class. The generic operations replace the object-type specific ones defined in the C-API.
- Processing of filtered lists is done in the same way as in the C-language specification, using an Iterator instead of WMTPQueryHandle. The Iterator might return more than one element at a time.
- The limits on the size of string type attributes have been removed. Same for limit on the number of Participants associated with an ActivityInstance or ProcessInstance.
- The Unique Id attributes of the various workflow entities are replaced by their object Id (not an explicit attribute).
- ReturnCodes have been replaced by exceptions.
9. Appendix D: Audit Data

The following describes the Audit Data related to the functions defined in this specification. The WfMC Audit Data Specification identifies events related to workflow objects (in general changes of state or of attributes) and describes the format of Audit Data to be reported for these events. Events are in general triggered by an external interaction with the Enactment Service, e.g., via an operation defined in this specification. An event can be a directly associated to the operation (e.g., WMStartProcessInstance triggers a WMProcessInstancesStarted event) or indirectly triggered by such an interaction, mediated by the Enactment Service (e.g., WMStartProcessInstance will cause state changes for the start activities of a process, resulting in WMActivityInstanceStateChanged events). An implementation of an Enactment Service complies with the WfMC Audit Data Specification if Audit Records are supported for all events identified in that document. For convenience of the reader we have included references to Audit-relevant events triggered by the functions described in this specification; for each operation the Audit Data Record and the directly associated event is stated. The following description provides pointers to the corresponding definitions in the WfMC Audit Data Specification; please refer to this document for details.

9.1 Auditing Process Definitions

The following table identifies the Audit Data for WAPI functions related to state changes Process Definitions. Operation refers to a WAPI function defined in this specification, Event Set refers to a section in the WfMC Audit Data Specification and Event identifies the event reported in the Audit Data record.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Audit Data Record</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMChangeProcessDefinitionState</td>
<td>Change Process Definition State</td>
<td>WMChangedProcessDefinitionState</td>
</tr>
</tbody>
</table>

9.2 Auditing Process Instances

The following table identifies the Audit Data for WAPI functions related to state changes and changes of attributes of activity instances. Operation refers to a WAPI function defined in this specification, Event Set refers to a section in the WfMC Audit Data Specification and Event identifies the event reported in the Audit Data record.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Audit Data Record</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMCreateProcessInstance</td>
<td>Create/Start Process/Subprocess Instance State</td>
<td>WMCreatedProcessInstance</td>
</tr>
<tr>
<td>WMStartProcessInstance</td>
<td>Create/Start Process/Subprocess Instance State</td>
<td>WMStartedProcessInstance</td>
</tr>
<tr>
<td>WMChangeProcessInstancesState</td>
<td>Change Process/Subprocess Instance State</td>
<td>WMChangedProcessInstanceState</td>
</tr>
<tr>
<td>WMChangeProcessInstanceState</td>
<td>Change Process/Subprocess Instance State</td>
<td>WMChangedProcessInstanceState</td>
</tr>
<tr>
<td>WMTerminateProcessInstances</td>
<td>Change Process/Subprocess Instance State</td>
<td>WMTerminatedProcessInstance</td>
</tr>
<tr>
<td>WMTerminateProcessInstance</td>
<td>Change</td>
<td>WMTerminatedProcessInstance</td>
</tr>
</tbody>
</table>

2 The new Process Definition functions are not covered here at the moment.
### 9.3 Auditing Activity Instances

The following table identifies the Audit Data for WAPI functions related to state changes and changes of attributes of activity instances. Operation refers to a WAPI function defined in this specification, Event Set refers to a section in the WfMC Audit Data Specification and Event identifies the event reported in the Audit Data record.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Audit Data Record</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMChangeActivityInstancesState</td>
<td>Change Activity Instance State</td>
<td>WMChangedActivityInstanceState</td>
</tr>
<tr>
<td>WMChangeActivityInstanceState</td>
<td>Change Activity Instance State</td>
<td>WMChangedActivityInstanceState</td>
</tr>
<tr>
<td>WMAssignActivityInstancesAttribute</td>
<td>Assign Activity Instance Attributes</td>
<td>WMAssignedActivityInstanceAttributes</td>
</tr>
<tr>
<td>WMAssignActivityInstanceAttribute</td>
<td>Assign Activity Instance Attributes</td>
<td>WMAssignedActivityInstanceAttributes</td>
</tr>
</tbody>
</table>

### 9.4 Auditing Workitems

The following table identifies the Audit Data for WAPI functions related to work items. Operation refers to a WAPI function defined in this specification, Event Set refers to a section in the WfMC Audit Data Specification and Event identifies the event reported in the Audit Data record.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Audit Data Record</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMAssignWorkitemAttribute</td>
<td>Assign Workitem Attributes</td>
<td>WMAssignedWorkitemAttributes</td>
</tr>
<tr>
<td>WMChangeWorkitemState</td>
<td>Change Workitem State</td>
<td>WMChangedWorkitemState</td>
</tr>
<tr>
<td>WMGetWorkitem</td>
<td>Change Workitem State</td>
<td>WMSelectedWorkitem (optional)</td>
</tr>
<tr>
<td>WMCompleteWorkitem</td>
<td>Change Workitem State</td>
<td>WMCompletedWorkitem</td>
</tr>
<tr>
<td>WMReassignWorkitem</td>
<td>Assign/Reassign Workitem</td>
<td>WMReassignedWorkitem</td>
</tr>
</tbody>
</table>
10. Appendix E: Conformance Profiles

This chapter deals with definition of criteria for a specific implementation of a Workflow Enactment Service to be conformant with the WAPI specification. Rather than requesting an implementation to support all of the functions specified above to conform with the WiMC standard, we define various levels of conformance. A set of Profiles is defined, each profile identifying a set of operations that address a specific usage scenario. An implementation of an Enactment Service might choose to comply with some, but not necessarily all of the Profiles.

10.1 Philosophy and Approach

The following conformance profiles are non-exclusive sets of functions from the WAPI2 specification. They strike a balance between the Vendor’s desire to simplify the conformance process and the Customer’s desire to have a straightforward and understandable conformance statement. The Conformance Profiles achieve this balance through the use of required WAPI Functions within optional profiles -- by definition, any profile is optional but its functionality is not. The philosophy behind their organization is as follows:

*Their basic structure is easy to understand.* The framework is understandable to customers and vendors who may not be intimately familiar with the specification and the history of its development.

*They provide flexibility for vendors by avoiding an “all-or-nothing” conformance framework.* The profiles mirror the general capabilities of today’s workflow products. Vendors may choose to support any number of the profiles, but do not have to support them all -- we will measure conformance on a profile-by-profile basis. For example, a vendor could choose to provide only WorkList Handler support, and could earn a conformance certification just for that Profile.

*Each Profile defines a set of functions that deliver business value to the customer in a predictable, meaningful way.* Customers can evaluate products using these conformance profiles. Each profile provides a meaningful service between the vendor’s product and the customer’s client applications that use the profile. Customers want behavioral consistency across different implementations of this interface; that consistency is the result of the simple nature of these profiles.

10.2 Practice and Policy

*A vendor can not claim conformance to this or any other WiMC specification unless specifically authorized to make that claim by the WiMC. WiMC grants this permission only upon the verification of the particular vendor’s implementation of the published specification, according to applicable test procedures defined by WiMC.*

When a vendor chooses to support a Conformance Profile, all WAPI Functions in that profile must actually “do something” in the vendor product representative of that WAPI Function’s purpose. It is not acceptable to return a “WM_Unsupported” error message for a WAPI Function that is part of a supported profile.

Each vendor must produce documentation showing attribute mappings; i.e., which of their product’s attributes are accessible using any of the attribute WAPI Functions in each supported profile.

Vendors may choose to support additional WAPI functions, along with vendor-specific API functions not prescribed in the Coalition specification. In such a case, WiMC encourages the vendor to document those function calls (and their associated attribute mappings) as an addendum to their documentation.

Each implementation must include program stubs for **all** unsupported WAPI functions. A call to any of these unsupported functions must return a “WM_Unsupported” error message.
10.3 The WAPI Conformance Profiles and Functions

For each Conformance Profile a function is defined that allows an application to check whether or not the specific Profile is supported by the implementation. Each implementation must include all Is<xxx>ProfileSupported() functions. These functions are in the following format:

WMIs<xxx>ProfileSupported() - where <xxx> is the name of a particular Conformance Profile - API commands are intended to allow a user application to inquire whether a vendor's implementation of WM functions supports a certain Conformance Profile.

10.3.1 WMIsWorkListHandlerProfileSupported

NAME

WMIsWorkListHandlerProfileSupported  - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsWorkListHandlerProfileSupported informs the user application that this WFMC implementation fully supports all the WorkList Handler functions that comprise the Work List Handler Conformance Profile.

INTENDED USE

Implementation of this conformance profile provides external worklist handler functionality to a client application.

WMTErrRetType WMIsWorkListHandlerProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

WORKLIST HANDLER CONFORMANCE PROFILE FUNCTIONS

The following functions comprise the Worklist Handler Conformance Profile:

WMConnect
WMDisconnect
WMOpenWorkList
WMFetchWorkItem
WMCloseWorkList
WMGetWorkItem
WMCompleteWorkItem
WMReassignWorkItem
WMOpenWorkItemAttributesList
WMFetchWorkItemAttribute
WMCloseWorkItemAttributesList
WMGetWorkItemAttributeValue
WMAssignWorkItemAttribute

RELATED AUDIT EVENTS

The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:
• All Audit Events related to state and attribute changes of Work Items, described by the Audit Data Types ‘Change WorkItem State’ and ‘Assign WorkItem Attributes’
10.3.2 WMIsProcessControlStatusProfileSupported

NAME

WMIsProcessControlStatusProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsProcessControlStatusProfileSupported informs the user application that this WFMC implementation fully supports all the Process Control Status functions that comprise the Process Control Status Conformance Profile.

INTENDED USE

Implementation of this conformance profile allows a client application to select and manage process instances.

WMTErrRetType WMIsProcessControlStatusProfileSupported()

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Arguments</td>
<td></td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

PROCESS CONTROL STATUS CONFORMANCE PROFILE FUNCTIONS

The following functions comprise the Process Control Status Conformance Profile:

WMConnect
WMDisconnect
WMOpenProcessDefinitionsList
WMFetchProcessDefinition
WMCloseProcessDefinitionsList
WMCreateProcessInstance
WMStartProcess
WMTerminateProcessInstance
WMOpenProcessInstanceStatesList
WMFetchProcessInstanceState
WMCloseProcessInstanceStatesList
WMChangeProcessInstanceState
WMOpenProcessInstancesList
WMFetchProcessInstance
WMCloseProcessInstancesList
WMGetProcessInstance
WMOpenProcessInstanceAttributesList
WMFetchProcessInstanceAttribute
WMCloseProcessInstanceAttributesList
WMGetProcessInstanceAttributeValue
WMAssignProcessInstanceAttribute

RELATED AUDIT EVENTS

The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:
• All Audit Events related to state and attribute changes of Process Instances, described by the Audit Data Types ‘Change Process / Subprocess Instance State’ and ‘Assign Process / Subprocess Attributes’
10.3.3 WMIsProcessDefinitionProfileSupported

NAME
WMIsProcessDefinitionProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION
The WMIsProcessDefinitionProfileSupported informs the user application that this WFMC implementation fully supports all the Process Definition functions that comprise the Process Definition Conformance Profile.

INTENDED USE
Implementation of this conformance profile enables a client application to display a list of available process definitions and their respective states.

WMTErrRetType WMIsProcessDefinitionProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

PROCESS DEFINITION CONFORMANCE PROFILE FUNCTIONS
The following functions comprise the Process Definition Conformance Profile:

WMConnect
WMDisconnect
WMOpenProcessDefinitionStatesList
WMFetchProcessDefinitionState
WMCloseProcessDefinitionStatesList
WMChangeProcessDefinitionState
WMOpenProcessDefinitionsList
WMFetchProcessDefinition
WMCloseProcessDefinitionsList

RELATED AUDIT EVENTS
The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:

• All Audit Events related to state changes of Process Definitions, described by the Audit Data Types ‘Change Process Definition State’
10.3.4 WMIsProcessAdminProfileSupported

NAME

WMIsProcessAdminProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsProcessAdminProfileSupported informs the user application that this WFMC implementation fully supports all the Process Admin functions that comprise the Process Admin Conformance Profile.

INTENDED USE

Implementation of this conformance profile allows a client application to support global manipulation of process instances by an administrator. Contrast this set with the Process Control Status functions which work only on individual process instances.

WMTErrRetType WMIsProcessAdminProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

PROCESS ADMIN CONFORMANCE PROFILE FUNCTIONS

The following functions comprise the Process Admin Conformance Profile:

WMConnect
WMDisconnect
WMChangeProcessInstancesState
WMTerminateProcessInstances
WMAbortProcessInstances
WMAbortProcessInstance
WMAssignProcessInstancesAttribute
WMOpenProcessInstanceStatesList
WMFetchProcessInstanceState
WMCloseProcessInstanceStatesList
WMOpenProcessDefinitionsList
WMFetchProcessDefinition
WMCloseProcessDefinitionsList
WMOpenProcessInstancesList
WMFetchProcessInstance
WMCloseProcessInstancesList
WMOpenProcessInstanceAttributesList
WMFetchProcessInstanceAttribute
WMCloseProcessInstanceAttributesList

RELATED AUDIT EVENTS

The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:
• All Audit Events related to state changes of Process Instances, described by the Audit Data Types
  ‘Change Process / Subprocess Instance State’
10.3.5 WMIsActivityControlStatusProfileSupported

NAME

WMIsActivityControlStatusProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsActivityControlStatusProfileSupported informs the user application that this WFMC implementation fully supports all the Activity Control Status functions that comprise the Activity Control Status Conformance Profile.

INTENDED USE

Implementation of this conformance profile allows a client application to select and manage activity instances.

WMErrRetType WMIsActivityControlStatusProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

ACTIVITY CONTROL STATUS CONFORMANCE PROFILE FUNCTIONS

The following functions comprise the Activity Control Status Conformance Profile:

WMConnect
WMDisconnect
WMOpenActivityInstanceStatesList
WMFetchActivityInstanceState
WMCloseActivityInstanceStatesList
WMChangeActivityInstanceState
WMOpenActivityInstancesList
WMFetchActivityInstance
WMCloseActivityInstancesList
WMGetActivityInstance
WMOpenActivityInstanceAttributesList
WMFetchActivityInstanceAttribute
WMCloseActivityInstanceAttributesList
WMGetActivityInstanceAttributeValue
WMAssignActivityInstanceAttribute

RELATED AUDIT EVENTS

The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:

- All Audit Events related to state and attribute changes of Activity Instances, described by the Audit Data Types ‘Change Activity Instance State’ and ‘Assign Activity Instance Attributes’
10.3.6 WMIsActivityAdminProfileSupported

NAME
WMIsActivityAdminProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION
The WMIsActivityAdminProfileSupported informs the user application that this WFMC implementation fully supports all the Activity Admin functions that comprise the Activity Admin Conformance Profile.

INTENDED USE
Implementation of this conformance profile allows a client application to support global manipulation of activity instances by an administrator. Contrast this set with the Activity Control Status functions which work only on individual activity instances.

WMTErrRetType WMIsActivityAdminProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE
WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

ACTIVITY ADMIN CONFORMANCE PROFILE FUNCTIONS
The following functions comprise the Activity Admin Conformance Profile:

WMConnect
WMDisconnect
WMChangeActivityInstancesState
WMAssignActivityInstancesAttribute
WMOpenProcessDefinitionsList
WMFetchProcessDefinition
WMCloseProcessDefinitionsList
WMOpenActivityInstanceStatesList
WMFetchActivityInstanceState
WMCloseActivityInstanceStatesList
WMOpenActivityInstanceAttributesList
WMFetchActivityInstanceAttribute
WMCloseActivityInstanceAttributesList

RELATED AUDIT EVENTS
The following Audit events are related to the operations included in this profile and would be audited by an implementation that is compliant with the Audit Data Profile:
• All Audit Events related to state and attribute changes of Activity Instances, described by the Audit Data Types ‘Change Activity Instance State’ and ‘Assign Activity Instance Attributes’
10.3.7 WMIsEntityHandlerProfileSupported

NAME

WMIsEntityHandlerProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsEntityHandlerProfileSupported informs the user application that this WFMC implementation fully supports all the Entity Handler functions that comprise the Entity Handler Conformance Profile.

INTENDED USE

Implementation of this conformance profile provides entity handler functionality to a client application.

WMTErrRetType WMIsEntityHandlerProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

ENTITY HANDLER CONFORMANCE PROFILE FUNCTIONS

The following functions comprise the Entity Handler Conformance Profile:

WMConnect
WMDisconnect
WMOpenProcessDefinitionsList
WMFetchProcessDefinition
WMCloseProcessDefinitionsList
WMCreateEntity
WMAddEntity
WMOpenEntitiesList
WMOpenOwnedEntitiesList
WMFetchEntity
WMCloseEntitiesList
WMRemoveEntity
WMDeleteEntity

10.3.8 WMIsAuditRecordProfileSupported

NAME

WMIsAuditRecordProfileSupported - Connect to the WFM Engine for this series of interactions

DESCRIPTION

The WMIsAuditRecordProfileSupported informs the user application that this WFMC implementation fully supports all the Audit Record capabilities for all other implemented Conformance Profiles.
INTENDED USE
Implementation of this conformance profile provides audit record support for the other conformance profiles.

WMErrRetType WMIsAuditRecordProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

AUDIT RECORD CONFORMANCE PROFILE FUNCTION
The following guidelines apply to the Audit Record Conformance Profile:

An implementation of any of the previous WAPI 2 Conformance Profiles may optionally include implementation of the Audit Record requirement for that Profile’s functions. In order to be conformant with the Audit Record Specification for this interface, the vendor must implement Audit Records for each implemented Profile. For example, if a vendor has a conforming implementation of both the WorkList Handler and the Process Control and Status profiles, they must implement Audit Records for both profiles in order to achieve Audit Record Specification Conformance.

10.3.9 WMToolAgentProfileSupported

NAME
WMToolAgentProfileSupported – Connects and supports different Tool Agents to enable application invocation

DESCRIPTION
The WMToolAgentProfileSupported informs the user application that this WFMC implementation fully supports application invocation via the Tool Agent architecture model.

INTENDED USE
Implementation of this conformance profile provides an interface to integrate application control mechanisms for workflow integration reasons.

WMErrRetType WMToolAgentProfileSupported()

Argument Description
No Arguments

ERROR RETURN VALUE

WM_TRUE  - If Conformance Profile is supported
WM_FALSE - If Conformance Profile is not supported.

TOOL AGENT CONFORMANCE PROFILE FUNCTION
The following guidelines apply to the Tool Agent Conformance Profile:
An implementation of any of the previous WAPI 2 Conformance Profiles may optionally include implementation of the Tool Agent requirement for that Profile’s functions. In order to be conformant with the Tool Agent Specification for this interface, the vendor must implement Tool Agent interfaces, which enable application invocation via the implemented Profile.

The following functions comprise the Tool Agent Conformance Profile:

- WMTAConnect
- WMTADisconnect
- WMTAInvokeApplication
- WMTARequestAppStatus
- WMTATerminateApp
11. **Appendix F: Workflow Definition Functions**

The following describes a new set of functions that deals with definition of workflow models. The first section describes an abstract machinery for handling of building blocks of workflow models - abstract entities. Entity handling functions include creating and deleting entities as well as functions to get and set their attributes.

An entity can be whatever a specific vendor supports as building block for a workflow definition; however, a basic set of entity types that should always be supported (i.e., those corresponding to the Instances that can be accessed via the Application Client Interface) is defined in the documentation of Process Definition Interchange documentation (Interface-1).

The middle piece of this chapter deals with connecting the abstract machinery of entities to the objects already introduced in this specification: entities are owned either by an Enactment Service or by a particular Process Definition; functions are described that enable editing of workflow objects in the context of an Enactment Service or a concrete Process Definition.

Due to the generic architecture of Workflow Definition Functions, implementations have to obey the semantical structure of the Process Definition Interchange Process Model as defined in the documentation of Interface-1, Process Definition Interchange Interface [Process Definition Interchange Process Model, WfMC TC-1016].

A list of attributes of Process Model Entities is provided with the documentation of the Process Definition Interchange documents [Process Definition Attribute List, WfMC TC-1019].

### 11.1 Entity Handling functions

The following defines a set of generic functions which treat all objects maintained by an Enactment Service as Entities, ignoring their specific semantics in a Workflow context. All entities have an identifier, a name and a type and other, type specific attributes. The ID is unique within a scope and remains constant from session to session, and from client to client. The ID is used to allow entities to refer to each other in a persistent way.

#### 11.1.1 Entity Data Types

```c
typedef struct
{
    WMTEntityID entity_id;
    WMTText entity_type[NAME_STRING_SIZE];
    WMTText entity_name[NAME_STRING_SIZE];
    void * entity_private_data;
} WMTEntity;
```
11.1.2 WMCreateEntity

NAME

WMCreateEntity - Creates a new entity.

DESCRIPTION

This is how new entities are created that compose the workflow definition. The entity created is a workflow persistent entity. The structure for the new entity will be returned. The entity is scoped either by the context of an enactment service or by another entity.

```c
WMTErrRetType WMCreateEntity (in WMTPSessionHandle psession_handle, in WMTPEntity scoping_entity, in WMTName entity_class, in WMTName entity_name, out WMTPEntity entity)
```

Argument Description

- **psession_handle**: Pointer to the structure with the session information created by a call to WMConnect.
- **scoping_entity**: The entity that owns the new entity
- **entity_class**: The vendor defined entity class that is to be created. Specifies what class of entity is to be created.
- **entity_name**: The user defined name provided for this entity.
- **entity**: Pointer to a buffer which will receive the entity structure.

ERROR RETURN VALUE

- WM_SUCCESS
- WM_INVALID_SCOPE
- WM_INVALID_CLASS
- WM_READONLY_CLASS

11.1.3 WMOpenEntitiesList

NAME

WMOpenEntitiesList - Specifies and opens the query to produce a list of all entities (owned by a specific entity) that meet the selection criterion of the filter.

DESCRIPTION

This command directs the WFM Engine to open the query to provide a list of entities which are available to a particular workflow participant, some of which may be modifiable by the participant. A typical usage for this operation is to get a list of all entities of a specific entity_type within a certain process model. This command will return a query handle for a list of entities that match the specified value for the attribute. The command will also return, optionally, the total count of entities available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1. If pentity_def_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL entities in a given scope.

```c
WMTErrRetType WMOpenEntitiesList ( in WMTPSessionHandle psession_handle, in WMTPEntity scoping_entity, in WMTPFilter pentity_def_filter, in WMBoolean count_flag, out WMTPQueryHandle pquery_handle, out WMTPInt32 pcount)
```
Argument Name                  Description
psession_handle              Pointer to a structure containing information about the context for this action.
scoping_entity               The entity that represents the scope of entities to be included in the query result.
entity_def_filter            Filter associated with the entities.
count_flag                   Boolean flag that indicates if the total count of entities should be returned.
pquery_handle               Pointer to a structure containing a unique query information.
Pcount                       Total number of entities that fulfill the filter condition.

ERROR RETURN VALUE
WM_SUCCESS                   WM_INVALID_SESSION_HANDLE
WM_INVALID_FILTER            WM_INVALID_SCOPE

REQUIREMENTS
No requirements are assumed to exist with regard to the type of process model. No requirements are assumed to exist with regard to how workflow participant’s are identified within the WFM Engine.

RATIONALE FOR API
This command and the corresponding fetch calls allows a workflow participant to retrieve the entities which a workflow participant is authorized to work on.
11.1.4 WMFetchEntity

NAME

WMFetchEntity - Returns the next entity from the set of entities that met the selection criterion stated in the WMOpenEntitiesList call.

DESCRIPTION

This command directs the WFM Engine to provide one entity from the list of entities which are available to a particular workflow participant, some of which may be modifiable by the participant. It is assumed that not all processes in an organization may be modified by all workflow participants. This fetch function, as well as all other fetch functions in this API, will return subsequent items after every call, one at a time. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The sort order in which the items are returned is specific of the workflow engine servicing the call, no specific order should be assumed.

WMTErrRetTypeDef WMFetchEntity {
    in WMTPSessionHandle psession_handle,
    in WMTPQueryHandle pquery_handle,
    out WMTPEntityID entity_id)

Argument Name | Description
--------------|--------------------------------------------------
psession_handle | Pointer to a structure containing information about the context for this action.
pquery_handle | Identification of the specific query handle returned by the WMOpenEntitiesList query command.
entity_id | Id of the next entity.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ENTITY
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
11.1.5 WMCloseEntitiesList

NAME
WMCloseEntitiesList - Closes the query of entities.

DESCRIPTION
WMTErrRetType WMCloseProcessModelEntitiesList(
    in WMTPSessionHandle psession_handle,
    in WMTPQueryHandle pquery_handle)

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to a structure containing information about the context for this action.</td>
</tr>
<tr>
<td>pquery_handle</td>
<td>Identification of the specific query handle returned by the WMOpenEntitiesList query command.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
11.1.6 WMDeleteEntity

NAME
WMRemoveEntity - Deletes an entity.

DESCRIPTION
WMTErrRetType WMDeleteEntity ( 
    WMTPSessionHandle psession_handle,
    WMTPEntity scoping_entity,
    WMTPEntityID entity_id)

Argument Description
psession_handle Pointer to the structure with the session information created by a call to WMConnect.
scoping_entity The entity that owns the entity to be deleted
entity_id Pointer to the unique id of the entity being deleted.

ERROR RETURN VALUE
WM_SUCCESS
WM_INVALID_SCOPE
11.2 Entity Attribute Manipulation

Every entity has attributes which contain specific information about the entity. These values are accessed via the WMGetEntityAttributeValue and WMSetEntityAttributeValue commands. Standard attributes will be defined for each standard entity type, and there will be other attributes that vendors will wish to implement specifically for their systems. In this way the entities are extensible by vendors. Some attributes contain scalar values, and others contain a collection of values. The multi valued attributes are called “attribute lists” in this document. The values in an attribute list are accessed through the following functions: WMOpenEntityAttributeValueList, WMFetchEntityAttributeValue, WMCloseEntityAttributeValueList. The open command returns a query handle which is used to fetch subsequent values. Multi-valued attributes are updated through the use of WMClearEntityAttributeValueList and WMAddEntityAttributeValue.
11.2.1 WMOpenEntityAttributesList

NAME

WMOpenEntityAttributesList - Specifies and opens the query to produce the list of attributes for a specific entity that match the filter criterion.

DESCRIPTION

This command will return a query handle for a list of attributes for an entity. The command will also return, optionally, the total count of attributes available. If the count is requested and the implementation does not support it, the command will return a pcount value of -1.

One of the uses of this API, together with the corresponding fetch and close calls is to allow a workflow application to query the Workflow Engine for the available attributes that are defined for an entity, in order to offer this list to the application user. If pentity_attr_filter is NULL, then the function, with the corresponding fetch calls will return the list of ALL attributes available for the entity.

WMTErrRetType WMOpenEntityAttributesList (  
in WMTPSessionHandle psession_handle,  
in WMTPEntity scoping_entity,  
in WMTEntity_Id entity_id,  
in WMTPFilter pentity_attr_filter,  
in WMTBoolean count_flag,  
out WMTPQueryHandle pquery_handle,  
out WMTPInt32 pcount)

Argument Name | Description
---|---
psession_handle | Pointer to a structure containing information about the context for this action.
scoping_entity | The entity that scopes the entity
entity_id | Filter associated with the entity attributes.
pentity_attr_filter | Boolean flag that indicates if the total count of entity attributes should be returned.
count_flag | Pointer to a structure containing a unique query information.
pquery_handle | Total number of attributes for this entity.
pcount | **ERROR RETURN VALUE**
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ENTIY
11.2.2 WMFetchEntityAttribute

NAME

WMFetchEntityAttribute - Returns the next entity attribute from the list of attributes that match the filter criterion.

DESCRIPTION

This command returns an entity attribute. This fetch function will return subsequent entity attributes after every call. The fetch process is complete when the function returns the error WM_NO_MORE_DATA. The function will return attribute name and its type and length; valid types are all WMT data types defined below in this document plus

- expressions of the form ListOf(Entity_Class) where Entity_Class is a string, identifying an entity class supported by the Enactment Service
- expressions of the form ListOf(Data_Type) where Data_Type is one of the basic WMT types Values of attributes of type List are handled using the WMT...EntityAttributeValuesList operations described below.

WMTErrRetType WMFetchEntityAttribute (  
in WMTPSessionHandle psession_handle,  
in WMTPQueryHandle pquery_handle,  
out WMTPAttrName pattribute_name,  
out WMTPInt32 pattribute_type,  
out WMTPInt32 pattribute_length,  
in WMTInt32 buffer_size)

Argument Name     Description

psession_handle   Pointer to a structure containing information about the context for this action.
pquery_handle     Identification of the specific query handle returned by the WMOpenEntityAttributesList query command.
pattribute_name   Pointer to the name of the attribute.
pattribute_type   Pointer to the type of the attribute.
pattribute_length Pointer to the length of the attribute value.
buffer_size       Size of the buffer.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_NO_MORE_DATA
11.2.3 WMCloseEntityAttributesList

NAME

WMCloseEntityAttributesList - Closes the query for entity attributes.

DESCRIPTION

WMTErrRetType WMCloseEntityAttributesList ( 
    in WMFSessionHandle psession_handle,
    in WMFQueryHandle pquery_handle)

Argument Name               Description
psession_handle             Pointer to a structure containing information about the context for this action.
pquery_handle               Identification of the specific query handle returned by the
                            WMOpenEntityAttributesList query command.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
11.2.4 WMGetEntityAttributeValue

**NAME**

WMGetEntityAttributeValue - Retrieves an attribute from an entity.

**DESCRIPTION**

Returns the value of the attribute named. See WMOpenEntityAttributeValueList to get all of the elements of a multi-valued attribute. The value of the attribute named is copied into the attribute_value buffer specified. If the buffer is not large enough for the entire value, then only the part that fits will be placed in the buffer, but no error will result. The attribute_length will return the correct length of the attribute value, not necessarily the amount of data returned.

**WMTErrRetType WMGetEntityAttributeValue (****

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to the structure with the session information created by a call to</td>
</tr>
<tr>
<td></td>
<td>WMConnect.</td>
</tr>
<tr>
<td>entity_handle</td>
<td>Pointer to the entity structure from which the attribute is being retrieved.</td>
</tr>
<tr>
<td>attribute_name</td>
<td>The name of the attribute from which to retrieve the value.</td>
</tr>
<tr>
<td>attribute_type</td>
<td>Returns the type of the value that has been returned.</td>
</tr>
<tr>
<td>attribute_length</td>
<td>Returns the length of the value in the attribute.</td>
</tr>
<tr>
<td>pattribute_value</td>
<td>A pointer to a buffer which will receive the value of the attribute.</td>
</tr>
<tr>
<td>buffer_size</td>
<td>The size of the buffer. This value used by the API to restrict writing of data to this length.</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

WM_SUCCESS
WM_NOT_SINGLE_VALUE
11.2.5 WMOpenEntityAttributeValueList

NAME

WMOpenEntityAttributeValueList - opens a multi-valued attribute on an entity for retrieving each of the values individually. The type of a value from the attribute is returned. A query handle is returned to fetch the individual values from. The count of items in the collection is optional. If the name of a single valued attribute is given, an error will result.

DESCRIPTION

WMErrMsgRetType WMOpenEntityAttributeValueList(
    in WMTPSessionHandle psession_handle,
    in WMTPEntity scoping_entity,
    in WMTPEntity entity_handle,
    in WMTPAttrName attribute_name,
    out WMTPInt32 attribute_type,
    out WMTPQueryHandle query_handle,
    out WMTPInt32 pcount)

Argument            Description
psession_handle     Pointer to the structure with the session information created by a call to WMConnect.
entity_handle       Pointer to the struct representing the entity.
attribute_name      The name of the multi-valued attribute to retrieve values from.
attribute_type      The collection of values as assumed to be of the same type since a collection is just a multi-valued attribute, so the collection_type is really the type of a single value in the collection.
query_handle        This query handle is used for WMFetchEntityCollectionValue and WMCloseEntityCollection
pcount               The number of values held in this attribute. This is optional. The value of negative one (-1) will indicate that the value is not supported.

ERROR RETURN VALUE

WM_SUCCESS
WM_NOT_MULTI_VALUED
11.2.6  WMFetchEntityAttributeValue

NAME

WMFetchEntityAttributeValue - Retrieves an attribute from an entity.

DESCRIPTION

WMTErrRetType WMFetchEntityAttributeValue(
    in WMTPSessionHandle psession_handle,
    in WMTPQueryHandle pquery_handle,
    out WM TInt32 attribute_length,
    out WMTPVoid pattribute_value,
    in WM TInt32 buffer_size)

Argument  Description
psession_handle  Pointer to the structure with the session information created by a call to
pquery_handle  WMConnect.
attribute_length  Pointer to the query structure created with WMOpenEntityCollection
pattribute_value  Returns the length of the value in the attribute
buffer_size  A pointer to a buffer which will receive the value of the attribute.
The size of the buffer. This value used by the API to restrict writing of data to
this length.

ERROR RETURN VALUE

WM_SUCCESS
### 11.2.7 WMCloseEntityAttributeValueList

**NAME**

WMCloseEntityAttributeValueList - Closes the query handle used to retrieve a collection (a multi-valued attribute).

**DESCRIPTION**

```c
WMErrRetType WMCloseEntityAttributeValueList(
  in WMTPSessionHandle psession_handle,
  in WMTPQueryHandle pquery_handle)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to the structure with the session information created by a call to WMConnect.</td>
</tr>
<tr>
<td>pquery_handle</td>
<td>Pointer to the query structure created with WMOpenEntityCollection</td>
</tr>
</tbody>
</table>

**ERROR RETURN VALUE**

WM_SUCCESS
11.2.8 WMAssignEntityAttributeValue

NAME

WMAssignEntityAttributeValue - Set an attribute of an entity.

DESCRIPTION

WMTErrRetType WMAssignEntityAttributeValue (  
in WMTPSessionHandle psession_handle,  
in WMTEntity entity_handle,  
in WMTPAttrName attribute_name,  
in WM TInt32 attribute_type,  
in WM TInt32 attribute_length,  
in WM TText pattribute_value)

Argument Description

Psession_handle Pointer to the structure with the session information created by a call to WMConnect.

entity_handle Pointer to the entity structure from which the attribute is being retrieved.

Attribute_name The name of the attribute to put the value into.

Attribute_type The type of the value.

Attribute_length The length of the value in the buffer.

Pattribute_value A pointer to a buffer which contains the value of the attribute.

ERROR RETURN VALUE

WM_SUCCESS

WM_NOT_SINGLE_VALUED
11.2.9 WMClearEntityAttributeList

NAME

WMClearEntityAttributeList - Deletes all of the values in a multi-valued attribute.

DESCRIPTION

WMTErrRetType WMClearEntityAttributeList(
    in WMTPSessionHandle psession_handle,
    in WMPEntity entity_handle,
    in WMTPAttrName attribute_name
);

Argument Description

Psession_handle Pointer to the structure with the session information created by a call to
WMConnect.
entity_handle Pointer to the entity structure from which the attribute is being erased.
Attribute_name The name of the attribute to be cleared out.

ERROR RETURN VALUE

WM_SUCCESS
WM_NOT_MULTI_VALUE
11.2.10 WMAddEntityAttributeValue

NAME

WMAddEntityAttributeValue - Add a value to a multi-valued attribute of an entity.

DESCRIPTION

WMTErrRetType WMAddEntityAttributeValue(
    in WMTPSessionHandle psession_handle,
    in WMTPEntity entity_handle,
    in WMTPAttrName attribute_name,
    in WM TInt32 attribute_type,
    in WM TInt32 attribute_length,
    in WMTPVoid pattribute_value
);

Argument Description
Psession_handle Pointer to the structure with the session information created by a call to WMConnect.
entity_handle Pointer to the entity structure from which the attribute is being retrieved.
Attribute_name The name of the collection (multi-valued attribute) to add the value into.
Attribute_type The type of the value.
Attribute_length The length of the value in the buffer.
Pattribute_value A pointer to a buffer which contains the value of the attribute.

ERROR RETURN VALUE

WM_SUCCESS
WM_NOT_MULTI_VALUEGED
11.3 Process Modelling Functions

The following set of functions supports creation and modification of a workflow process model. A process model is made up from building blocks called process definition entities in this specification. Examples for process definition entities are Activity Definitions (the nodes of a process model, which become Activity Instances when the process model is executed) and Transitions (the connections between Activity Definitions). The generic entity handling functions defined above can be applied to modify the contents of a process model. A standard set of such entities, which is obtained from the WfMC Process Definition Specification document is described in the next section.
11.3.1 WMOpenWorkflowDefinition

NAME

WMOpenWorkflowDefinition - Prepares for editing of workflow definition entities (i.e., on the Enactment Service scope level).

DESCRIPTION

This command tells the Enactment Service to prepare for editing of workflow definition entities it controls. This is the starting point for getting all of the entities that compose workflow definitions. This entity will form the scoping entity for most of the requests for further entities controled by the Enactment Service.

WMTErrRetType WMOpenWorkflowDefinition {
  in  WMTPSessionHandle psession_handle,
  in  WMTText name(NAME_STRING_SIZE)
  in  WMTText scope(NAME_STRING_SIZE)
  out WMTPEntity workflow_definition_handle
}

Argument Description

Psession_handle

Pointer to the structure with the session information created by a call to WMConnect.

Name

Identifier of the editing context

Scope

Scope of editing context

Workflow_definition_handle

Handle of the entity representing the workflow editing context. This entity will be used as scoping entity for subsequent editing on entities owned by the Enactment Service. The entity has type ‘workflow definition’, name taken from the second input parameter, and no additional attributes.

ERROR RETURN VALUE

WM_SUCCESS
11.3.2 WMCloseWorkflowDefinition

NAME

WMCloseWorkflowDefinition - Allows the system to free up any resources that are maintained to handle requests for entities within the Enactment Service.

DESCRIPTION

WMTErrRetType WMCloseWorkflowDefinition(
    in WMTPSessionHandle psession_handle,
    in WMTPEntity workflow_definition_handle
)

Argument Description
Psession_handle
    Pointer to the structure with the session information created by a call to WMConnect.
Workflow_definition_handle
    Pointer to an entity structure which represents the contents of the Enactment Service. It is assumed that all entities within the scope of this context become inaccessible once the workflow definition is closed.

ERROR RETURN VALUE

WM_SUCCESS
11.3.3 WMCreateProcessDefinition

NAME

WMCreateProcessDefinition - creates a new process definition

DESCRIPTION

Creates an entity for a new empty process definition within the system. The empty process definition can then have entities created within it.

```c
WMTErrRetType WMCreateProcessDefinition(
    in WMTPSessionHandle psession_handle,
    out WMTPProcDefID pproc_def_id
)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psession_handle</td>
<td>Pointer to the structure with the session information created by a call to WMConnect.</td>
</tr>
<tr>
<td>pproc_def_id</td>
<td>Pointer to the new process definition id for the process definition to create.</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_SUCCESS
11.3.4 WMDeleteProcessDefinition

NAME

WMDeleteProcessDefinition - deletes a process definition

DESCRIPTION

DELETES a process definition from the scope defined by the current session.

```c
WMTErrRetType WMDeleteProcessDefinition(
    in WMTPSessionHandle psession_handle,
    in WMTPProcDefID pproc_def_id
)
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psession_handle</td>
<td>Pointer to the structure with the session information created by a call to</td>
</tr>
<tr>
<td></td>
<td>WMConnect.</td>
</tr>
<tr>
<td>pproc_def_id</td>
<td>Pointer to the process definition to be deleted</td>
</tr>
</tbody>
</table>

ERROR RETURN VALUE

WM_SUCCESS
11.3.5 WMOpenProcessDefinition

NAME

WMOpenProcessDefinition - Prepares for editing of a process model.

DESCRIPTION

This command tell the Enactment Service to prepare for editing of the specified process model. This is the starting point for getting all of the entities that compose the process definition itself. This entity will form the scoping entity for most of the requests for further entities within the process definition.

WMTErrRetType WMOpenProcessDefinition (  
  in WMTPSessionHandle psession_handle,  
  in WMTPProcDefinition proc_definition  
  out WMTPEntity proc_model_handle  
 )

Argument Description
Psession_handle Pointer to the structure with the session information created by a call to WMConnect.
proc_definition Process Definition to be edited
proc_model_handle Handle of the entity representing the process model. This entity will be used as scoping entity for subsequent editing on the process definition

ERROR RETURN VALUE

WM_SUCCESS
11.3.6 WMCloseProcessDefinition

NAME

WMCloseProcessDefinition - Allows the system to free up any resources that are maintained to handle requests for entities within the process definition.

DESCRIPTION

WMTErrRetType WMCloseProcessDefinition(
    in WMTPSessionHandle psession_handle,
    in WMTPEntity proc_model_handle
)

Argument Description

psession_handle Pointer to the structure with the session information created by a call to WMConnect.

proc_model_handle Pointer to an entity structure which represents the contents of the process definition. It is assumed that all entities within the scope of this process definition become inaccessible once the process definition is closed.

ERROR RETURN VALUE

WM_SUCCESS

11.4 Standard Process Modelling Entity Types

The following describes the standard entity types that should be supported by every Workflow Engine and their respective attributes (mandatory and optional); specific implementations may have additional types and additional attributes for each type. The types are ProcessDefinition, ActivityDefinition, Transition, Participant, Application and ProcessData. The entity types and their attributes are taken from the WfMC specification of the Process Definition Interface, which describes the Workflow Process Definition Language (WPDL); please refer to this document for further details. Some changes have been made to adjust the attribute names used by WPDL to those used in the Workflow Client Application Interface specification.

11.4.1 Additional Data Types

typedef struct 
{ 
    WMTText name[NAME_STRING_SIZE];
} WMTName;

typedef struct 
{ 
    WMTText date[NAME_STRING_SIZE];
} WMTDate;

typedef struct 
{ 
    WMTInt32 duration;
} WMTDuration;

typedef struct 
{ 
    WMTInt32 cost;
} WMTCost;

typedef struct 
{ 
    WMTText documentation[1024];
} WMTDocumentation;
typedef struct {
    WMTText expression[256];
} WMTCondExpression; // Condition expression. To be refined using expression grammar

typedef struct {
    WMTText expression[256];
} WMTPartExpression; // Participant expression. To be refined using expression grammar

typedef struct {
    WMTText expression[256];
} WMTApplicationSpec; // Application identification. To be refined...

Attribute structure used by ProcessData entity type to define complex data structures; attribute value might hold ProcessDataID if type is COMPLEX, default value otherwise.

typedef struct {
    WMTText attribute_name[NAME_STRING_SIZE];
    WMTInt32 attribute_type; // type of the attribute
    WMTInt32 attribute_length; // length of the attribute value
    WMTPText pattribute_value; // pointer to the attribute value
} WMTAttribute;

11.4.1.1 WMAddTransition

NAME

WMAddTransition - Adds a transition definition to a process model.

DESCRIPTION

This command will return a transition definition entity owned by the process definition that is passed as second parameter, connecting the activity definition entities passed as third and fourth parameter.

WMTErrRetType WMAddTransition {
    in WMTPSessionHandle psession_handle,
    in WMTPProcModelID pproc_model_id,
    in WMTPActDefID psource_act_def_id,
    in WMTPActDefID ptarget_act_def_id,
    out WMTPEntity entity_handle
}

Argument          Description

psession_handle   Pointer to the structure with the session information created by a call to WMConnect.
pproc_model_id    Pointer to the process model owning the new transition
psource_act_def_id Pointer to the source activity definition of the transition
ptarget_act_def_id Pointer to the target activity definition of the transition
entity_handle     Pointer to a buffer which will receive the structure which represents a transition

ERROR RETURN VALUE

WM_SUCCESS

11.4.1.2 WMAddProcessDataAttribute

NAME

WMAddProcessDataAttribute - Adds an attribute to the list of attributes that define the data structure.
DESCRIPTION

WMErrRetType WMAddProcessDataAttribute (  
  in WMTPSessionHandle psession_handle, 
  in WMTPProcModID pproc_model_id, 
  in WMTPProcDataID pproc_data_id, 
  in WMTPAttrName pattribute_name, 
  in WM TInt32 attribute_type, 
  in WM TInt32 attribute_length, 
  in WM TPText pattribute_value)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pproc_model_id Pointer to a structure containing the process model entity ID.
pproc_data_id Pointer to a structure containing the process data definition identification for which the attribute will be assigned.
pattribute_name Pointer to the name of the attribute.
attribute_type Type of the attribute.
attribute_length Length of the attribute value.
pattribute_value Pointer to a buffer area provided by the client application where the attribute value will be placed. Can be identifier of another process data entity.

ERROR RETURN VALUE
WM_SUCCESS

11.4.1.3 WMRemoveProcessDataAttribute

NAME
WMRemoveProcessDataAttribute - Removes an attribute from the list of attributes that define the data structure.

DESCRIPTION

WMErrRetType WMRemoveProcessDataAttribute (  
  in WMTPSessionHandle psession_handle, 
  in WMTPProcModID pproc_model_id, 
  in WMTPProcDataID pproc_data_id, 
  in WMTPAttrName pattribute_name)

Argument Name Description
psession_handle Pointer to a structure containing information about the context for this action.
pproc_model_id Pointer to a structure containing the process model entity ID.
pproc_data_id Pointer to a structure containing the process data definition identification for which the attribute will be assigned.
pattribute_name Pointer to the name of the attribute. Must be unique within the data structure.

ERROR RETURN VALUE
WM_SUCCESS
12. Appendix G: States

The following describes above a set of standard valid states for each of the major workflow objects defined in this document. States are organized into several levels of granularity, lower level states refining higher-level ones. An implementation of the Enactment Service might choose to support states on any level of granularity, omit states and add additional states to the list defined below. A state for a particular workflow object can be identified by its name only or by specifying its full name including its super-state parents using dot notation; for examples see the section on Process Instance states below.

12.1 Process Instance States

The top level of states for a Process Instance distinguishes two states, open and closed. The open state has two sub-states, running and notRunning; notRunning in turn has two sub-states, notStarted and suspended.

The following list describes the states in detail:

- **open** - the Process Instance is being enacted
- **open.running** - the Process Instance is executing
- **open.notRunning** - the Process Instance is temporarily not executing
- **open.notRunning.notStarted** - the Process Instance has been created, but was not started yet
- **open.notRunning.suspended** - execution of the Process Instance was temporarily suspended
- **closed** - enactment of the Process Instance has been finished
- **closed.aborted** - enactment of the Process Instance has been aborted by a user (see the specification of WMAbortProcessInstance for a definition of abortion in contrast to termination)
- **closed.terminated** - enactment of the Process Instance has been terminated by a user (see the specification of WMTerminateProcessInstance for a definition of termination in contrast to abortion)
- **closed.completed** - enactment of the Process Instance has completed normally (i.e., was not forced by a user)

An implementation might decide to support refinement of states to a certain level only or omit certain states; valid sets of states include for example:

- open and closed
- notRunning, running and closed
- notStarted, running, completed and terminated
- ...

The following diagram shows the states and potential state-transitions; transitions are shown for the bottom-level states only, transitions between the higher-level states can be deduced from that easily; e.g., there is a transition from open to closed or from notRunning to running, but no transition backwards in both cases.
Here is a short discussion of the various state-transitions:

- When a Process Instance is created it will take its initial state, which is open.notRunning.notStarted (or just open, or open.notRunning depending on the level of granularity supported).
- Transitions can be made from notRunning states to the running state; transitions from the running to the notRunning super-state can be made to the suspended sub-state only.
- When enactment of a Process Instance is finished, its state will take one of the flavours of the closed state, depending on the way of ending enactment (normally completed, terminated or aborted). The completed state can only be reached from the running state since it represents normal completion of the Process Instance; the other closed sub-states are reached via the WMAbrtProcessInstance or WMTerminateProcessInstance operations.
- The closed state is a final state, i.e., there is no transition from a closed state to an open state.

### 12.2 Activity Instance States

The top level of states for an Activity Instance distinguishes two states, open and closed. The open state has three sub-states, running, notRunning; and suspended. The following list describes the states in detail:

- **open** - the Activity Instance is active
- **open.running** - the Activity Instance is executing
- **open.notRunning** - the Activity Instance is ready, but has not been started yet
- **open.suspended** - execution of the Activity Instance was temporarily suspended
- **closed** - enactment of the Activity Instance has been finished
- **closed.aborted** - enactment of the Activity Instance has been aborted, probably due to abortion of the owning Process Instance (see the specification of WMAbrtProcessInstance for a definition of abortion in contrast to termination)
- **closed.terminated** - enactment of the Activity Instance has been terminated, probably due to termination of the owning process instance (see the specification of WMTerminateProcessInstance for a definition of termination in contrast to abortion)
- **closed.completed** - enactment of the Activity Instance has completed normally (i.e., was not forced by a user or by a state change of its owning Process Instance)

The following diagram shows the states and potential state-transitions; transitions are shown for the bottom-level states only, transitions between the higher-level states can be deduced from that easily.
Here is a short discussion of the various state-transitions:

- When an Activity Instance is created it will take its initial state, which is `open.notRunning`.
- Transitions between the `notRunning` and the `suspended` states are in general initiated by the Enactment Service, triggered by a corresponding state change of the owning Process Instance; they could also be triggered via the WMChangeActivityInstanceState operation.
- Transitions between the `notRunning` and the `running` state might be initiated by the Application Client user via the WMGetWorkitem operation, but this is up to the specific Enactment Service; otherwise the transition is either initiated by the Enactment Service or by the Application Client user via the WMChangeWorkitemState or WMChangeActivityInstanceState operation.
- Transitions between the `running` and the `suspended` state are in general initiated by the Enactment Service as a result of a corresponding state change of the owning Process Instance; an Enactment service might allow this transition to be performed as a result of the WMChangeWorkitemState or via the WMChangeActivityInstanceState operation also.
- When enactment of an Activity Instance is finished it’s state will take one of the flavours of the `closed` state, depending on the way of ending enactment (normally `completed`, `terminated` or `aborted`). The `completed` state can only be reached from the `running` state since it represents normal completion of the Activity Instance.
- The `closed` state is a final state, i.e., there is no transition from a `closed` state to an `open` state.

### 12.3 Workitem States

The top level of states for a Workitem distinguishes two states, `open` and `closed`. The open state has three sub-states, `running`, `notRunning`; and `suspended`. The following list describes the states in detail:

- **open** - the Workitem is active
- **open.running** - the Workitem is executing
- **open.notRunning** - the Workitem is assigned to a participant, but has not been started yet
- **open.suspended** - execution of the Workitem was temporarily suspended
- **closed** - enactment of the Workitem has been finished
- **closed.aborted** - enactment of the Workitem has been aborted, probably due to abortion of the owning Process Instance (see the specification of WMAbortProcessInstance for a definition of abortion in contrast to termination)
• **closed.terminated** - enactment of the Workitem has been terminated, probably due to termination of the owning process instance (see the specification of WMTerminateProcessInstance for a definition of termination in contrast to abortion)

• **closed.completed** - enactment of the Workitem has completed normally (i.e., was not forced by a user or by a state change of its owning Process Instance)

The following diagram shows the states and potential state-transitions; transitions are shown for the bottom-level states only, transitions between the higher-level states can be deduced from that easily.

Here is a short discussion of the various state-transitions:
• When an Workitem is created it will take its initial state, which is open.notRunning.

• Transitions between the notRunning and the suspended state are in general initiated by the Enactment Service as a result of a corresponding state change of the owning Process Instance; an Enactment service might decide to allow this transition to be performed via the WMChangeWorkitemState operation also.

• Transitions between the notRunning and the running state might be initiated by the Application Client user via the WMGetWorkitem operation, but this is up to the specific Enactment Service; otherwise the transition is either initiated by the Enactment Service or by the Application Client user via the WMChangeWorkitemState operation or as a result of a WMChangeActivityInstanceState on the associated Activity Instance.

• Transitions between the running and the suspended state are in general initiated by the Enactment Service as a result of a corresponding state change of the owning Process Instance; an Enactment service might decide to allow this transition to be performed via the WMChangeWorkitemState operation also.

• When enactment of an Workitem is finished it’s state will take one of the flavours of the closed state, depending on the way of ending enactment (normally completed, terminated or aborted). The completed state can only be reached from the running state (via the WMCompleteWorkitem operation) since it represents normal completion of the Workitem.

• The closed state is a final state, i.e., there is no transition from a closed state to an open state.