The Workflow Management Coalition Specification

Workflow Management Coalition Workflow Client Application (Interface 2) Application Programming Interface (WAPI) Specification

Document Number WFMC-TC-1009

15-May-96
Version 1.1

Workflow Management Coalition
Avenue Marcel Thiry 204
1200 Brussels
Belgium
Tel: +32 2 774 96 33
Fax: +32 2 774 96 90
Email: 100113.1555@compuserve.com
or WfMC@eyam.be
www: http://www.aiai.ed.ac.uk/wfmc
or http://www.arms.ohio-state.edu/wfmc

First printing, November 1995
Second printing, version 1.1, May 1996

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 2 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 may 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 Process Control Functions**
- **WAPI Activity Control Functions**
- **WAPI Process Status Functions**
- **WAPI Activity Status Functions**
- **WAPI Worklist Functions**
- **WAPI Administration Functions**

3.1 Design Philosophy

There are a number of design assumptions and constraints that provide a framework or philosophy for the definition of this specification.

3.2 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.

3.3 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.

3.4 Defined Terms and Abbreviations

The terms used in this document are defined in the WFM Coalition Glossary.

3.5 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

3.6 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.

3.7 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.

```
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;

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:

```
#define NAME_STRING_SIZE      64
```

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

```
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:

```
#define UNIQUE_ID_SIZE        64
```

All WAPI function calls have a uniform error return datatype:

```
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 a 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.

```
typedef struct
{
       WMTText 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.

       WMTText password[NAME_STRING_SIZE];
       WMTText 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.
       WMTText scope[NAME_STRING_SIZE];
                                             // Identification of scope for the
                                                application.  If scope is not relevant,
                                                then this field would be empty and
                                                ignored.

}WMTConnectInfo;

typedef  WMTConnectInfo *WMTPConnectInfo;

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

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: <, >, =, !=, <=, <=
       WMTPText       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];
       WMTProcDefID          proc_def_id;
       WMTProcDefState       state;
} WMTProcDef;

typedef WMTProcDef *WMTPProcDef;
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;
       WMTInt32               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;
       WMTInt32               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;
       WMTWflParticipant     participant;
} WMTWorkItem;

typedef WMTWorkItem *WMTPWorkItem;
```

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 WMTAttrName[NAME_STRING_SIZE];
typedef WMTAttrName *WMTPAttrName;
```

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.

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**

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.

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

Argument	Description
`pconnect_info`	Pointer to structure containing the information required to create a connection.
`psession_handle`	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.

ERROR RETURN VALUE

```
        WM_SUCCESS
        WM_CONNECT_FAILED
```

6.1.2 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.

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

ERROR RETURN VALUE

```
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  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_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
```

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).

```
WMTErrRetType WMChangeProcessDefinitionState (
                  in   WMTPSessionHandle psession_handle,
                  in   WMTPProcDefID pproc_def_id,
                  in   WMTPProcDefState pproc_def_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_def_state`	Pointer to a structure that contains the name of the state to change the process definition to.

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.

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.

```
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) `proc_inst_id` through WMCreateProcessInstance, assign attributes to it and then call WMStartProcess.

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.

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.

```
WMTErrRetType 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.

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.

```
    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

```
WMTErrRetType WMCloseProcessInstanceStatesList (
                    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 **WMOpenProcessInstanceStatesList** query command.

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 *in-progress*.

```
    WMTErrRetType WMChangeProcessInstanceState (
                        in  WMTPSessionHandle psession_handle,
                        in  WMTPProcInstID pproc_inst_id,
                        in  WMTPProcInstState pproc_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.
pproc_inst_state	Pointer to a structure that contains the name of the process state that you want to change the instance to.

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.

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  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_attr_filter	Filter associated with the process instance attributes.
count_flag	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

```
WMTErrRetType 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.

```
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.
`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
```

6.2.19 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.

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 WMOpenActivityInstanceStatesList

NAME

WMOpenActivityInstanceStatesList - 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.

```
WMTErrRetType WMOpenActivityInstanceStatesList (
                     in  WMTPSessionHandle psession_handle,
                     in  WMTPProcInstID pproc_inst_id,
                     in  WMTPActivityInstID pactivity_inst_id,
                     in  WMTPFilter pact_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 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
`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.

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 *in-progress.* 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.

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.

```
WMTErrRetType WMFetchActivityInstanceAttribute (
                   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 **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
`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.

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  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.
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

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:

1	All the process instances associated with a process definition.	**WM(Open+Fetch+Close)ProcessInstancesList**
2	A view of a single process instance.	**WMGetProcessInstance**

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.

```
WMTErrRetType WMCloseProcessInstancesList (
                    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 **WMOpenProcessInstancesList** query command.

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.

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

Argument Name	Description
`psession_handle`	Pointer to a structure containing information about the context for this action.
`pproc_inst_id`	Pointer to the process instance identification.
`pproc_inst`	Pointer to a structure containing the requested process instance information. Includes the state and other attributes of the process instance.

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:

1	All the activity instances associated to a process definition or instance	**WM(Open+Fetch+Close)ActivityInstancesList**
2	A view of a single activity within a process instance.	**WMGetActivityInstance**

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.

```
WMTErrRetType WMOpenActivityInstancesList (
                    in  WMTPSessionHandle psession_handle,
                    in  WMTPFilter pactivity_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.
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,
                        in  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
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.

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.

```
WMTErrRetType 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.

```
WMTErrRetType 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.

```
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.

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)
```

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.

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.

6.6.6 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.)

```
WMTErrRetType WMReassignWorkItem (
                       in  WMTPSessionHandle psession_handle,
                       in  WMTPWflParticipant psource_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.

6.6.7 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  WMTPFilter pwork_item_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.
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
```

6.6.8 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
```

6.6.9 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
```

6.6.10 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.

```
WMTErrRetType 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
WM_INVALID_SESSION_HANDLE
WM_INVALID_ATTRIBUTE
WM_INSUFFICIENT_BUFFER_SIZE
```

6.6.11 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.

```
WMTErrRetType WMAssignWorkItemAttribute (
                  in  WMTPSessionHandle psession_handle,
                  in  WMTPProcInstID pproc_inst_id,
                  in  WMTPWorkItemID pwork_item_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.
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_WORKITEM
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_ASSIGNMENT_FAILED
```

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.

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.

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
```

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.

```
WMTErrRetType WMAssignProcessInstancesAttribute (
                   in  WMTPSessionHandle psession_handle,
                   in  WMTPProcDefID pproc_def_id,
                   in  WMTPFilter pproc_inst_filter,
                   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_def_id	Pointer to a structure containing the process definition ID for which the attribute of all process instances will be changed.
pproc_inst_filter	Pointer to a structure containing the filter information for this request.
ppattribute_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_DEFINITION
WM_INVALID_FILTER
WM_INVALID_ATTRIBUTE
```

6.7.5 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.

```
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	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 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.
pactivity_def_id	Pointer to a structure containing the activity definition identification for which the attribute will be assigned.
pact_inst_filter	Pointer to a structure containing the filter information for this request.
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_DEFINITION
WM_INVALID_ACTIVITY_NAME
WM_INVALID_FILTER
WM_INVALID_ATTRIBUTE
```

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.

```
WMTErrRetType WMAbortProcessInstances (
                    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 who all processes instances is being aborted.
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
```

REQUIREMENTS

None

RATIONALE FOR API

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

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.

```
WMTErrRetType 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.

7. Appendix A

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.