OPC Command Execution Specification

Version 1.0

Status: Draft 29

March 26, 2004

Specification Type	Industry Standard Specification		
Title:	**OPC Command Execution Specification**	Date:	**March 26, 2004**
Version:	1.0	Soft Source:	MS-Word OPC_CMD_Ver_1_Draft29NoWS.doc
Author:	OPC Foundation	Status:	**Draft 29**

Synopsis:

This specification is the specification of the interface for developers of OPC Command Execution clients and OPC servers.. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of servers and clients by multiple vendors that shall inter-operate seamlessly together.

Trademarks:

Required Runtime Environment:

For the COM version a Windows operating system (NT4, 2000, XP or later) is required.

For the WebService version, any operating system that is capable of parsing XML messages and can support 64-bit integers and 64-bit floating point types. Practically, the runtime environment should be a 32-bit operating system with a Web server, an XML parser and a SOAP API of some sort.

Table of Contents

1 Introduction

A general introduction to OPC is contained in a separate OPC Overview Document (OPCOVW.DOC). This particular document deals specifically with the OPC Command Services.

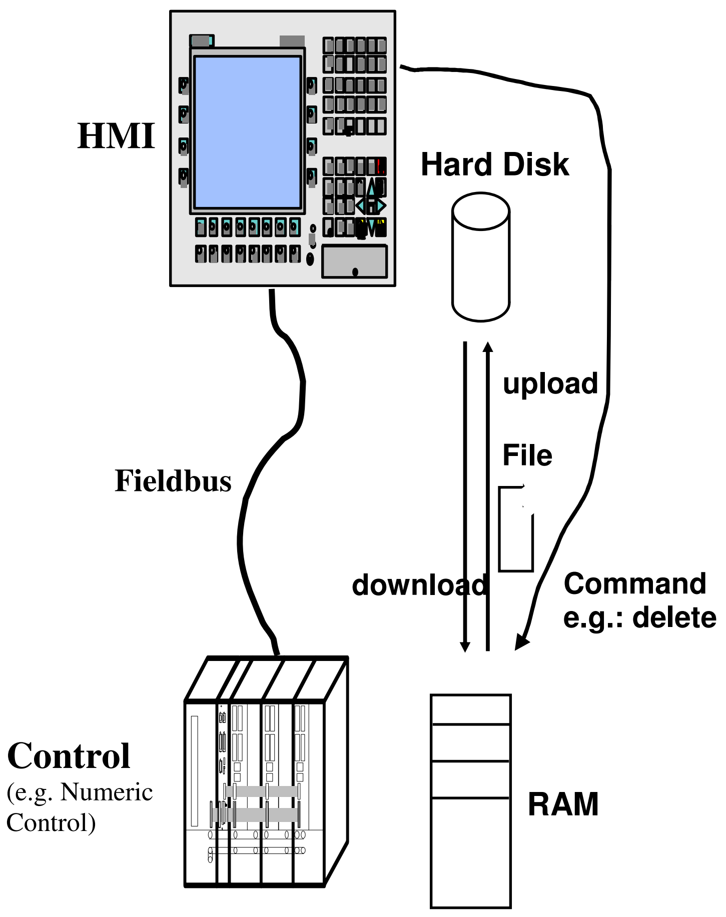
1.1 Background

Automation devices (like numerical controllers) often define certain features via commands. In comparison with variables (items), commands have a much more complex behavior:

- Each command has a set of parameters, which can be variables;
- Each command can be executed several times in parallel;
- A command has an underlying state machine, state changes can be notified;
- Commands can return complex information after their execution.

OPC Commands Servers can be implemented stand-alone, or in process with different types of OPC Servers like DataAccess, Alarms & Events or Batch.

The following figure illustrates an automation device (a Numeric Controller) which defines commands to download or upload programs (files) or to perform administration of the program namespace:



1.2 Purpose

The purpose of this specification is to provide a common base for definition of commands.

Based on this specification it will be possible to use generic clients for execution and control of commands. That is why this specification has a strong emphasis on how to discover commands and the associated execution state machine.

In this context a command is seen as an “atomic” method that is triggered on the server by one invocation call. If the application requires a more complex set/sequence of methods, it is expected that a number of commands will be defined to control the sequence.

1.3 Relationship to Other OPC Specifications

OPC Commands is a standalone specification in that Commands Servers and Clients can be implemented without using any other OPC-defined interfaces (like Data Access or Alarms &Events).

Note: OPC Commands Servers are also expected to implement the interfaces described in *OPC Common Definitions and Interfaces 1.0*. This document contains common rules, design criteria and the specification of interfaces which are common for most OPC Servers, including Commands.

1.4 Target Audience

This specification is intended for developers of OPC Commands Clients and Servers. It is assumed that the reader is familiar with Microsoft COM. It is also recommended that the reader review the reference documents described in Section 1.5.

1.5 Prerequisites

Readers are assumed to be familiar with the following OPC Specification.

- OPC Common Definitions and Interfaces 1.0

This specification is available from the OPC Foundation Web site: http://www.opcfoundation.org/

1.6 Deliverables

The deliverables from the OPC Foundation with respect to the OPC Commands Project include the OPC Commands 1.00 Specification and OPC Commands 1.00 sample code, available from the OPC Foundation Web Site (members only).

1.7 Errata

Any clarifications or corrections to this specification found after publication will be posted to the following web page:

http://www.opcfoundation.org/forum/viewtopic.php?t=583

2 Fundamental Concepts

2.1 Overview

The OPC Command services define the means to discover, invoke and manage commands (methods) that a server provides.

The discovery is provided by the Command Information Services (e.g., QueryCMDs). A client can discover the sets of commands (domains) that the server provides, the entities that commands can be applied to (command targets) and the detailed command descriptions

The Command Execution Services support command invocation and control. A command can be invoked either synchronously or asynchronously. A synchronously invoked command is similar to a blocking function call. The client is blocked during the execution of the command. When the command execution finishes, the call returns and the client receives a report and the results of the execution.

Any command execution that was started asynchronously can be monitored and managed via the Command Execution services. A client may also issue a command and disconnect from the server. Even when no client is connected, the command execution will proceed and when the client re-connects at a later time, it will receive a report of the execution history and the results of the execution.

2.2 CommandTarget

CommandTarget is an abstraction of the entities that commands can be applied to. e.g.,

- a control device
- a robot
- a vessel
- an area of an A&E server
- the ID of an order that is placed in a MES system
- the ID on a recipe

A service is provided to browse for available command targets.

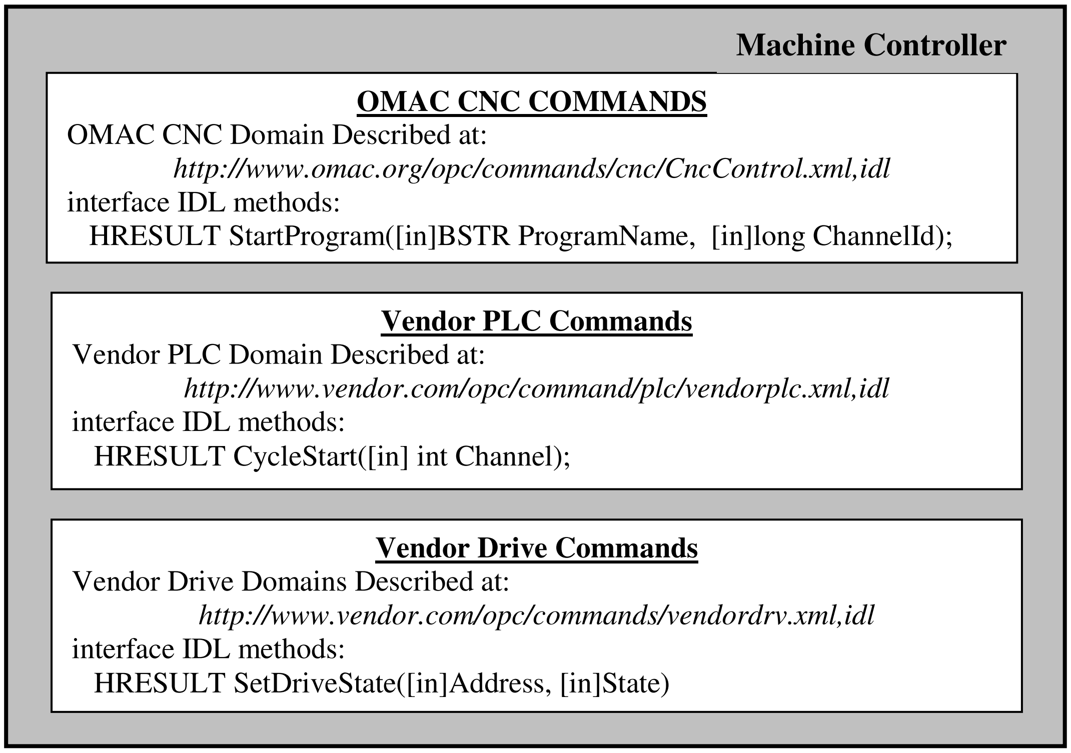
One command can apply to more than one target.

2.3 Command Domains

OPC Command Servers may support commands for different types of applications, e.g., for numeric control as well as for batch execution. OPC Commands therefore provides for the grouping of commands into so-called domains. Domains provide a name space or scope for a set of commands. They prevent ambiguity resulting from duplicate command names and facilitate directing the command to the correct system component for execution. As an example, the domains for numeric control and batch could both define a “Start” command.

The command set of a domain can be established by any product vendor.
In addition, however, domains with command sets can also be established by an industry group or a standards body. This is strongly recommended, as it provides a basis for conformance classes and better facilitates interoperability. For example, the open architecture industry group OMAC could publish domains containing sets of commands with well defined behaviors in manufacturing processes. Vendors could then state conformance to the domain by supporting the domain’s set of commands. Domains can be used to discover system capabilities prior to dispatching jobs.

The following Illustration shows a machine control device that supports multiple sets of commands: CNC, PLC, and Drive, including a “virtual” OMAC standard conformant set. Each set is defined within a domain namespace (specified by its URI). The behavior of each command is well defined. The device advertises its support for the commands by listing the domains. The different domain commands supported by the device may have identical names performing different functions.



OMAC CNC Domain Described at:
http://www.omac.org/opc/commands/cnc/CncControl.xml,idl
interface IDL methods:
```
HRESULT StartProgram([in]BSTR ProgramName,  [in]long ChannelId);
```

Vendor PLC Domain Described at:
http://www.vendor.com/opc/command/plc/vendorplc.xml,idl
interface IDL methods:
```
HRESULT CycleStart([in] int Channel);
```

Vendor Drive Domains Described at:
http://www.vendor.com/opc/commands/vendordrv.xml,idl
interface IDL methods:
```
HRESULT SetDriveState([in]Address, [in]State)
```

2.4 Command execution state machine

The logical model of the command execution state machine can be described as following:

The state machine is a list of transitions.

Each transition (a change from one state to another state) has associated with it:

- (optional) transition condition
- start state
- end state
- (optional) action (with input and ouput data)
- message event (with transitional data)

By default a transition during the “normal” execution of a command does not need a trigger. The command is invoked by one trigger, then is executed and finishes without any further trigger.

Exeptions for this need to be explicitely defined. Examples for such exeptions are: the transitions to and from the “Halted” state.

Transition Condition: A Boolean expression on which a transition may depend. The transition progresses when the expression evaluates to *True*.

State: A uniquely identified condition of the command system or subsystem. A state can contain defining state data such as an error code and can also contain a subordinate FSM within its namespace.

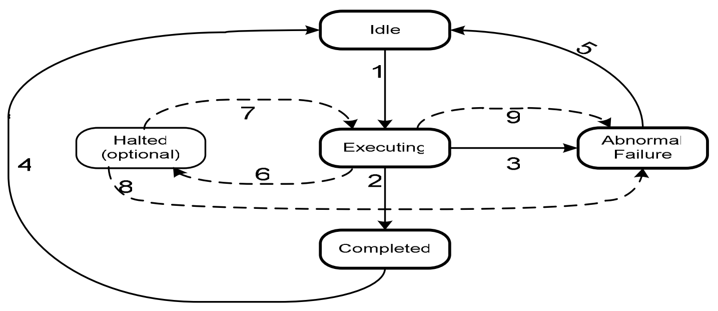
Action: Operation performed during a state transition. The action's attributes include input arguments and output arguments. The completion of the Action can result in triggering an associated event to initiate the next state transition. The input arguments can be provided from internal or external sources. Externally supplied arguments are defined as input parameters. The output arguments can be made available to external destinations in which case they are defined as output parameters.

Example for an action: sending of a message.

Messaging Event: A defined, detectable change in the system that can serve as the stimulus to initiate a state transition. An event is identified by a name and may have associated synchronous data. Detection of event conditions can be explicit or implicit, and results in the notification of its enrollees. The creation of a transition object results in an enrollment with an event condition.

2.5 Basic Command State Table

All Commands are derived from a base FSM state table that describes their most primitive behavior.



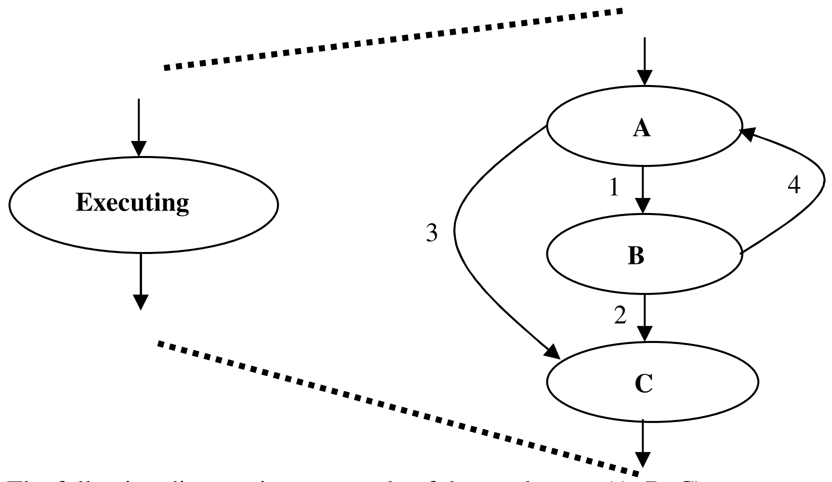
Basic Command Finite State Machine (FSM)

Transition Collection	Initial State	Resultant State	Event	Trigger	Action
1	Idle	Executing	Invoke	(Async)Invoke	Perform task
2	Executing	Complete	Finished	Automatic internal	Save result
3	Executing	Failure	Aborted	Automatic internal	Save result
4	Complete	Idle	Reset	Automatic internal	Return result
5	Failure	Idle	Reset	Automatic internal	Return result
6	Executing	Halted	Halted	ControlComma nd(Suspend)	Stop execution
7	Halted	Executing	Resumed	ControlComma nd(Resume)	Resume execution
8	Halted	Abnormal Failure	Cancelled	ControlComma nd(Cancel)	Cancel execution
9	Executing	Abnormal Failure	Cancelled	ControlComma nd(Cancel)	Cancel execution

The Command Invocation is the event that initiates the transition from idle to executing. The associated action is the initiation of the specified command task. As the task terminates, an internal mechanism triggers the finished or aborted event, causing a transition to complete or failure. The state data saved reflects the result of the task performed, such as a result code or an error code. The final transition event could be triggered by either internal or external means, causing the state data to be returned as the command reenters the idle state. The support for the "Halted" state for a command is optional. The transition into and from this state is caused by the method **ControlCommand**().

Substates:

Because each state can also contain a subordinate finite state machine within its namespace, more complex and interruptible commands can be defined. State transitions can describe varying sequences depending on internal or external event conditions.



Example: The following diagram is an example of three substates (A, B, C).

Subordinate State Machine

Transition Collection	Initial State	Resultant State	Event	Transition Condition	Action
1	Substate.A	Substate.B	Time to check	Can interrupt	Check progress
2	Substate.B	Substate.C	Work Complete	True	Summarize
3	Substate.A	Substate.C	Abort work	Can interrupt	Summarize
4	Executing.B	Executing.A	More work	True	Continue

3 OPC Command Execution Services

3.1 Overview

This section specifies a set of abstract OPC Command services. Refer to the Appendix for the mapping of these abstract services to specific interface technologies including DCOM. Version 1.0 of this specification will include only the DCOM mapping.

3.2 Parameter Definitions

This section contains definitions of the parameters used in the OPC Command services. They are presented alphabetically.

Constructed parameters are composed of two or more component parameters. Their descriptions define their "use". The following codes are defined for the "Use" column. See the Appendix for the definition of how optional and conditional parameters are implemented in each mapped interface technology.

Use	Description
M	Mandatory
O	Optional. The parameter may be provided by the client. Or returned by the server
C	Conditional. The parameter is present only under the conditions specified in the parameter description
-	The parameter is not used.

3.2.1 Argument

This constructed parameter contains the name and value of an argument. Arguments can be IN or OUT arguments either for the command or for actions that are executed as part of a state transition.

Parameter Name	Use	Description
Name	M	The name of the argument.
ArgumentValue	M	The value including type information.

3.2.2 StateChangeEvent

This constructed parameter contains the information that describes a single state change. It is used in response to a synchronous invoke, an AsyncQuery or in a callback.

Parameter Name	Use	Description
EventName	M	A string parameter with the name of the event that triggered the transition. The following are OPC-defined Events. If a command executes according to the Finite State Machine (FSM) these names should be used rather than defining vendor-specific events. These names should also be used in custom-state machines if the same events are provided..

Event	Description
Invoke	Triggered by an (Async)Invoke, this event causes a transition from *Idle* into *Executing*.
Finished	After successful execution, this event signals the transition from *Executing* into *Completed* state.
Aborted	Errors that cause abortion of the execution are signalled using this event. The new state will be *Aborted*.
Reset	This event is a notification that the state has changed to *Idle* state.
Halted	Triggered by ControlCommand(Suspend), this event causes a transition from *Executing* into *Halted*.
Resumed	Triggered by ControlCommand(Resume), this event causes a transition from *Halted* into *Executing*.
Cancelled	Triggered by ControlCommand(Cancel), this event causes a transition from *Executing* or *Halted* into *AbnormalFaulure*..

Parameter Name	Use	Description
EventTime	M	The time when the event occurred.
EventData	O	Vendor-specific data – a string that typically includes an XML document specified by **EventDataTypeDef** (see **GetCMDDescription**). If **EventDataTypeDef** is empty, the value is expected to be a human readable string. **EventData** is data whose value(s) are captured at the moment the event occurred. It provides the client with synchronous information associated with the event or execution of the command. Some examples of event data would be a sensor reading, a calibration value, or a performance statistic. Note: This data is distinct from state data in that it is not integral to the command or state transition.
OldState	M	Name of the previous state (includes full hierarchy if substate) The following state names of the Finite State Machine are pre-defined by OPC:

FSM States
Idle
Executing
Complete
AbnormalFailure
Executing
Halted
Executing

Parameter Name	Use	Description
NewState	M	Name of the current state (includes full hierarchy if substate)
StateData	O	Vendor-specific data – a string that typically includes an XML document specified by **StateDataTypeDef** (see **GetCMDDescription**). If **StateDataTypeDef** is empty, the value is expected to be a human readable string. StateData is intended to be information directly relevant to a specific state transition. An example of state data is error information relative to a subordinate state transition.

InArgument [1..N]	C	Input arguments to the action that triggered the transition into the new state. These arguments can be provided from internal or external sources. Externally supplied arguments are defined as input parameters.
OutArgument [1..N]	C	The output arguments generated by the action. The output arguments can be made available to external destinations in which case they are defined as output parameters. These are the expected products of the command, such as a query result, a completion status, or a calculated value. These results may serve as the input arguments to the next state's activity. An example would be PartCount - Number of Parts produced in this process.

3.3 OPC Command Information Services

OPC Command information services allow a client to retrieve information about commands from a server.

3.3.1 QueryCapabilities

With this service the client can query for specific features of the command server.

3.3.1.1 QueryCapabilities Request

The request has no data.

3.3.1.2 QueryCapabilities Response

The parameters of a **QueryCapabilities** response are described in the following table.

Parameter Name	Use	Description
MaxStorageTime	M	(double) Specifies the maximum time in Milliseconds that the command server will store state transition events.
SupportsEventFilter	M	(bool) If FALSE, the command server does not support filtering of state transition events.

The following table specifies the **QueryCapabilities** Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

3.3.2 QueryCMDs

With this service the client can query for the commands supported by the command server.

Commands are grouped into domains because the server might support commands for different areas of an application, e.g., for numeric control as well as for batch execution. The domains provide a type of namespace for a set of commands. This prevents ambiguity resulting from duplicate command names and facilitates directing the command to the correct system component for execution.

See section 2.3 for more information about command domains.

3.3.2.1 QueryCMDs Request

The request has no data.

3.3.2.2 QueryCMDs Response

The parameters of a **QueryCMDs** response are described in the following table.

Parameter Name	Use	Description		
DomainURIDefinition[1..N]	M	List of domains and their commands.		
		DomainURI	M	(string) A unique identifier for the set of commands.
		Description	O	(string) A description of the domain and its meaning
		CmdName[1..N]	M	(string) The command names that belong to this domain.

The following table specifies the **QueryCMDs** Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory

3.3.3 BrowseCMDTargets

A CommandTarget is an abstraction of an entity that commands can be applied to (e.g., ID of a Numeric Controller, a robot, an application, a plant, etc.). The following service is provided to browse for available command targets.

Note that commands may be defined as "global". I.e., they do not apply to a specific target. This is defined as part of the command description (**IsGlobal**).

Servers can organize the targets either flat or hierarchically.

3.3.3.1 BrowseCMDTargets Request

The parameters of a **BrowseCMDTargets** request are described in the following table.

Parameter Name	Use	Description
TargetID	O	(string) If specified, browsing happens at this branch. if missing or empty (Null), browsing happens at the root level. This parameter is only needed for hierarchical name spaces.
DomainURI	O	(string) This parameter is used as a filter. If missing, all targets are returned.
BrowseFilter	O	An enumeration {**all**, **branch**, **cmdTarget**} specifying which subset of browse elements to return. See the response section below to determine which combination of bits in **BrowseElement** are returned for each value of **BrowseFilter**.

3.3.3.2 BrowseCMDTargets Response

The parameters of a **BrowseCMDTargets** response are described in the following table.

Parameter Name	Use	Description		
Element [1..N]	M	An arbitrary number of elements which match the request.		
		Label	M	(string) A short, user-friendly portion of the namespace pointing to the element. This is the string to be used for display purposes in a tree control.
		TargetID	O	(string) Uniquely identifies this element in the server's browse space. It can be used in subsequent calls to **BrowseCMDTargets**, **SyncInvoke,** and **AsyncInvoke**.
		IsTarget	C	If **IsTarget** is set then the element can be used in **SyncInvoke** or **AsyncInvoke**. If **TargetID** is missing and **IsTarget** is TRUE then this element is a "hint" versus being a valid command target.
		HasChildren	C	If **HasChidren** is set, then this indicates that the returned element has children and can be used for a subsequent browse service calls. If it is too time consuming for a server to determine if an element has children, then this value should be set TRUE so that the the client is given the opportunity to attempt to browse for potential children.
		DomainURI [1..N]	C	One or more DomainURIs which implicitely define the set(s) of commands that can be applied to this **commandTarget**.

The following table describes the relationship between the possible values of **HasChildren** and **IsTarget** supplied in the request.

Element in Response		**Description**	**BrowseFilter in Request**		
IsTarget	**HasChildren**		**All**	**Branch**	**Item**
FALSE	FALSE	An empty branch	●	●	
FALSE	TRUE	A branch that has children, or possibly has children.	●	●	
TRUE	FALSE	A **commandTarget** that is not a branch	●		●
TRUE	TRUE	An element that has children, or possibly has children, that is also a **commandTarget**	●	●	●

The following table specifies the **BrowseCMDTargets** Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_INVALIDBRANCH	The **TargetID** specified in the request is not a valid bramch.
OPC_E_INVALIDDOMAIN	The specified domain is not a valid domain for this server.

3.3.4 GetCMDDescription

Return the command description items in XML Format for the specified command.

Example "CMDDescription"'s can be found in 0.

3.3.4.1 GetCMDDescription Request

The parameters of a **GetCMDDescription** request are described in the following table.

Parameter Name	Use	Description
CommandName	M	Identifies the command the description of which is requested..
DomainURI	M	URI of the domain the command belongs to.

3.3.4.2 GetCMDDescription Response

The parameters of a **GetCMDDescription** response are described in the following table.

Parameter Name	Use	Description		
Description	M	Textual description of the command.		
IsGlobal	M	(bool) If TRUE this command is a "global" command and cannot be applied to a specific command target.		
ExecutionTime	M	(double) Estimated executionTime in seconds. 0.0 means no estimate is possible.		
EventDefinition[1..N]	C	The table of event definitions for the command. This table is mandatory only if the server supports event filtering.		
		EventName	M	A string parameter with the name of the event.
		EventDescription	O	Describes the meaning of the event.
		EventDataTypeDef	O	The XML Schema for the data of this event definition. If **EventDataTypeDef** is missing, the data is expected to be a human readable message. **EventData** is data whose value(s) are captured at the moment the event occurred. It provides the client with synchronous information associated with the event or execution of the command. Some examples of event data would be a sensor reading, a calibration value, or a performance statistic. Note: This data is distinct from state data in that it is not integral to the command or state transition.
StateDefinition[1..N]	O	The state table for the command.		
		StateName	M	A string parameter with the name of the state.
		StateDescription	O	Describes the meaning of the state.
		StateDataTypeDef	O	The XML Schema for the data of this state definition. If **StateDataTypeDef** is missing, the data is expected to be a human readable message. **StateData** is intended to be information directly relevant to a specific state transition. An example of state data is error information relative to a subordinate state transition.
ActionDefinition[1..N]	O	The action table for the command.		
		ActionName	M	A string parameter with the name of the action.
		ActionDescription	O	Describes the meaning of the action.
		EventName	M	The event which is triggered by this action

		INArguments	O	Arguments that are used as input for the action.
		OUTArguments	O	Arguments which are the result of this action.
StateTransition[1..N]	O	The state transition table for the command.		
		TransitionID	M	(int) Identifier of the transition.
		StartState	M	Name of state before the transition occurs.
		EndState	M	Name of state after successful transition.
		TriggerEvent	M	Event which is signalled as part of the transition.
		Action	O	Operation performed as the result of the state transition..
INParameter[1..N]	C	Description of the externally supplied input parameters.		
		Name	M	The name (string) for this parameter.
		ValueType	M	The data type for this parameter. For interoperability it is recommended that only simple datatypes are used.
		Optional	M	If TRUE, this paramater can be absent in the invocation.
		Description	O	Description of this parameter
		Value	O	Default value if parameter not provided in invocation.
		UnitType	O	(string) Specifies the engineering units e.g. “miles”, “DEGC” or “GALLONS”
		LowLimit	O	Represents the lowest permitted value.
		HighLimit	O	Represents the highest permitted value.
OUTParameter[1..N]	C	Description of the output parameters available upon completion (Details – see **INParameters**).		
SupportedControls	M	Indicates supported command controls. The most common controls are pre-defined by OPC. Additional vendor-specific controls are permitted. The following command controls are pre-defined by OPC:		
		Command Controls		
		Cancel		
		Suspend		
		Resume		
The following lists of dependencies are ANDed to form the complete list of dependencies.				
ANDDependency [1..N]	O	List of commands all of which have to be executed prior to this command		
ORDependenc y [1..N]	O	List of commands one of which has to be executed prior to this command..		
NOTDependenc y [1..N]	O	List of commands that must not have been executed prior to this command..		

The following table specifies the GetCMDDescription Fault Codes

Code	**Description**
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_INVALIDDOMAIN	The DomainURI is invalid.
OPC_E_INVALIDCMDNAME	The command name is invalid.

3.4 Command Execution Services

A command is characterized by a sequence of transitions between execution states and associated actions. The actions may require input data and generate output data. Invoking a command initiates the first state transition and consumption of any input parameter.

Command Execution provides synchronous and asynchronous services. It is expected that synchronous command invocation is used rarely – mainly for commands that are expected to be executed fast (seconds or even milliseconds).

Synchronous Execution:

When a command is invoked **synchronously**, the service returns with the complete results of the command execution. This includes events related to the command's execution and any output arguments associated with the current state's action.

Asynchronous Execution:

Asynchronous execution means that the server immediately returns after the command has been started successfully. Command execution continues in "background". The client can either query from time to time to check for the state of execution or register for callbacks which are sent at specified update rates but only if state transitions occurred. If supported by the server, the client can control the command, causing a transition to a suspended or cancelled state.

Asynchronous commands continue execution even if the client disconnects from the server (or crashes). Clients can reconnect to the commands they invoked.

Completion:

Command completion occurs with a transition to a designated state and the performance of any actions associated with that state. At this point the active state and its associated data reflect the results of the command. If the terminal state is a success state, the state data contains a success result code. When the final state is an abnormal termination state, the state data is a failure result code. The failure result code could reflect an error sub-state transition that is promoted when the command aborts.

When a command does not complete successfully, the condition preventing completion could arise from several sources including transport mechanism, operating system platform, or application code errors. To provide a mechanism to uniquely identify and scope the source of errors it is proposed to adopt a name space convention for the "facility" reporting the malfunction.

Additional States to indicate execution errors:

If a server needs to pass information about failure conditions back to the client (like not being able to queue commands), it can do this by using additional states or the Abnormal Failure state. By going into such a state the server can provide additional information with the StateChange event (e.g., "can only execute one command at a time", "server is not in the proper state to execute this command").

3.4.1 SyncInvoke

Instructs the server to invoke the specified command and execute it synchronously.

The parameters of a **SyncInvoke** request are described in the following table.

Parameter Name	Use	Description
CommandName	M	Identifies the command to be invoked on the request or the executing command on the response.
DomainURI	M	Identifies the domain of the specified command.
CommandTarget	O	The entity for which the caller wants to issue the command invocation request
InArgumentList	O	The input argument list for the command instance. These are the input values required for the first state action.
EventFilter[1..N]	O	Identifies the state changes which will be returned in the response to this service. This functionality is optional in the server. Default: If no filter is specified, all state changes are returned.

3.4.1.1 SyncInvoke Response

The parameters of a SyncInvoke response are described in the following table.

Parameter Name	Use	Description
StateChangeEvent [1..N]	M	See 3.2.2.

The following table specifies the SyncInvoke Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_BUSY	The server is currently not able to process this command.
OPC_E_EVENTFILTER_NOTSUPPORTED	The server does not support filtering of events.

3.4.2 AsyncInvoke

Instructs the server to invoke the specified command and execute it asynchronously. The server will execute the command until it is finished, cancelled or otherwise terminated abnormally. Execution is independend of the client's connection. i.e., even if the client disconnects (on purpose or unintentionally), the execution continues. Clients can connect again at a later time using the *InvokeUID* to get results or control the execution.

Unique ID (InvokeUID):
Each asynchronous command is assigned a unique ID provided by the invoker. Clients can use GUIDs or network addresses to assign unique ids. The server has to verify that this ID is unique across all executing commands.

Getting Results during execution:
The state changes detected during execution are stored by the server as an array of *StateChangeEvent* structures (possibly filtered by the event filter). They are sent to the client either with a callback or in response to *QueryState* service request.

Controlling execution:
If the command is controllable (see *SupportedControls* in 3.3.4.2), the client can request suspension/resumption of cancellation of the command (using the *InvokeUID*).

Cacheing StateChangeEvents (MaxStorageTime):
Once sent or queried, the server deletes the *StateChangeEvent* structures from its cache. Servers should be able to keep the stored but not yet delivered *StateChangeEvents* for some time, depending on the application domain (see *MaxStorageTime* in 3.3.1.2).

3.4.2.1 AsyncInvoke Request

The parameters of an **AsyncInvoke** request are described in the following table.

Parameter Name	**Use**	**Description**
CommandName	M	Identifies the command to be invoked on the request or the executing command on the response.
DomainURI	M	Identifies the domain of the specified command.
CommandTarget	O	The entity for which the caller wants to issue the command invocation request
InvokeUID	M	(string) Unique ID for this command invocation provided by the caller. The client can save this UID after creation to be able to reconnect at a later time.
InArgumentList	O	List of (Name, Value) The input argument list for the command instance. These are the input values required for the first state action.
EventFilter[1..N]	O	List of Event Names Identifies the state changes which will be returned in callbacks or in response to an AsyncQuery request. This functionality is optional in the server. Default: If no filter is specified, all state changes are returned.
CallbackID	O	Identifies the callback for asynchronous notification. This value depends strongly on the mapping. In COM we require an Interface Pointer, for WebServices a URL can be specified. If missing, the client has to call Query to receive the **StateChangeEvents**. Clients can disconnect the **CallbackID** at any time during execution. The **CallbackID** is disconnected automatically after the command execution is finished and all **StateChangeEvents** have been notified.
UpdateFrequency	O	(unsigned integer) Number of Milliseconds This parameter requests the rate at which the server will notify the client about **StateChangeEvents**. Notification takes place only if a state change occurred. By default notifications are sent immediately when they occur.
KeepAliveTime	O	(unsigned integer) Number of Milliseconds Clients can set the keep-alive time for a command to cause callbacks even when there are no new StateChangeEvents to report. Clients can then be assured of the health of the connection. Using this facility, a client can expect a callback within the specified keep-alive time. Servers shall reset their keep-alive timers when real data is sent (i.e. it is not acceptable to constantly send the keep-alive callback at a fixed period equal to the keep-alive time irrespective of state changes). This parameter is used only if a **CallbackID** parameter is specified.

3.4.2.2 AsyncInvoke Response

The parameters of AsyncInvoke Response are described in the following table.

Parameter Name	**Use**	**Description**
RevisedUpdateFrequency	O	The server may not support the requested frequency and will then return the supported frequency (the closest possible value).

The following table specifies the AsyncInvoke Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_BUSY	The server is currently not able to process this command.
OPC_E_EVENTFILTER_NOTSUPPORTED	The server does not support filtering of events.
OPC_E_INVOKE_UUID_DUPLICATE	The command was already invoked.

3.4.3 Connect

Clients can connect a callbackID to an asynchronously invoked command. Any StateChangeEvents in the server's cache will be immediately sent to the callback. After that, state changes will be sent according to the requested or revised *UpdateFrequency*.

Only one connection is permitted at any time.

If the command is already finished at the time of connection, the server will send all stored *StateChangeEvents*.

3.4.3.1 Connect Request

The parameters of a **Connect** request are described in the following table.

Parameter Name	Use	Description
InvokeUID	M	See Invoke (3.4.2.1).
CallbackID	O	See Invoke (3.4.2.1). Clients can disconnect the **CallbackID** at any time during execution. The **CallbackID** is disconnected automatically after the command execution is finished and all StateChangeEvents have been notified.
UpdateFrequency	O	See Invoke (3.4.2.1).
KeepAliveTime	O	See Invoke (3.4.2.1).

3.4.3.2 Connect Response

The parameters of Connect Response are described in the following table.

Parameter Name	Use	Description
RevisedUpdateFrequency	O	See Invoke response (3.4.2.2).

The following table specifies the Connect Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_NO_SUCH_COMMAND	A command with the specified UID is neither executing nor completed (and still stored in the cache).
OPC_E_ALREADY_CONNECTED	A client is already connected for the specified **InvokeUID**.

3.4.4 Disconnect

Allows the disconnection of a *CallbackID* during execution of a command. This will stop asynchronous notifications.

3.4.4.1 Disconnect Request

The parameters of a **Disconnect** request are described in the following table.

Parameter Name	Use	Description
InvokeUID	M	See Invoke (3.4.2.1).

3.4.4.2 Disconnect Response

If successful this service has no response parameters.

The following table specifies the **Disconnect** Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_NO_SUCH_COMMAND	A command with the specified UID is neither executing nor completed (and still stored in the cache).
OPC_E_NOT_CONNECTED	No **CallbackID** is currently connected for the specified command.

3.4.5 QueryState

Read the current state and cached *StateChangeEvents* (if available) for the specified invocation of a command.

After the *QueryState* operation the buffer is cleared. i.e., the next call will return only new *StateChangeEvents*.

3.4.5.1 QueryState Request

The parameters of a **QueryState** request are described in the following table.

Parameter Name	Use	Description
InvokeUID	M	See Invoke (3.4.2.1).
WaitTime	O	(32 bit integer) Instructs the server to wait the specified number of milliseconds before returning if there are no changes to report. A state change during this wait period will result in the service returning immediately rather than completing the wait time. If no state change occurred during this time, the server will return with the latest **StateChangeEvent** information. If missing a WaitTime of 0 is assumed.

Comments:
Note, that both **QueryState** and **OnStateChange** callback reset the buffer of recent **StateChangeEvents**.

3.4.5.2 QueryState Response

The parameters of a **QueryState** response are described in the following table.

Parameter Name	Use	Description
StateChangeEvent [1..N]	M	See 3.2.2.
CurrentlyPermittedControl	M	Lists the control commands that may be used in the current state (i.e., none, suspend, resume, cancel).
NoStateChange	M	If TRUE, no state changes occurred since the last QueryState request. The **StateChangeEvent** parameter contains one element, representing the current state.

The following table specifies the QueryState Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_NO_SUCH_COMMAND	A command with the specified UID is neither executing nor completed (and still stored in the cache).

3.4.6 Control

Request to control the specified command execution. Controls include **Suspend**, **Resume**, and **Cancel**. Note: Certain commands may not support these control features. The support is indicated in the command description (see 3.3.4.2).

The permitted controls at the current state are returned with the response of the QueryState service.

Changing the execution state triggers an event in the command state table, causing a transition to a suspended or cancelled state.

3.4.6.1 Control Request

The parameters of an Control request are described in the following table.

Parameter Name	Use	Description
InvokeUID	M	Unique identifier of the command execution instance provided by the client in the async command invocation.
Action	M	Control action to be performed for the asynchronous command invocation. Valid actions are: • Suspend • Resume • Cancel

3.4.6.2 Control Response

No data are provided in response to this request.

The following table specifies the Control Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.
E_OUTOFMEMORY	Not enough memory
E_INVALIDARG	An argument to the function was invalid.
OPC_E_NO_SUCH_COMMAND	A command with the specified UID is neither executing nor completed (and still stored in the cache).
OPC_E_ACTION_NOTSUPPORTED	The server does not support the specified action for this command.
OPC_E_INVALID_ACTION	The specified action is not possible in the current state of execution.

3.5 Client-Side Interface

3.5.1 OnStateChange

If a client has specified a **CallbackID** (either in *Invoke* or in *Connect*) it has to implement the *OnStateChange* interface. The server will call *OnStateChange* either at every state change or as defined by the **UpdateFrequency**.

The implementation of callbacks is quite different depending on the transport it is mapped to.

In COM the connection between client and server is always established. The interface therefore has to be implemented as a COM Interface.

Client and server in a web environment are only loosely coupled. The connection is typically closed after some time and has to be reestablished for the next call. To perform callbacks in such an environment, the client has to be a web server with its own URL..

3.5.1.1 OnStateChange Request

The parameters of a *OnStateChange* request are described in the following table.

After the *OnStateChange* operation the buffer is cleared. e.e., the next callback will pass only new **StateChangeEvent**s.

Parameter Name	Use	Description
StateChangeEvent [1..N]	M	See 3.2.2.
CurrentlyPermittedControl	M	Lists the control commands that may be used in the current state (i.e., none, suspend, resume, cancel).
NoStateChange	M	If TRUE, no state changes occurred since the last callback (only possible in a KeepAlive callback). The **StateChangeEvent** parameter then contains one element, representing the current state.

3.5.1.2 OnStateChange Response

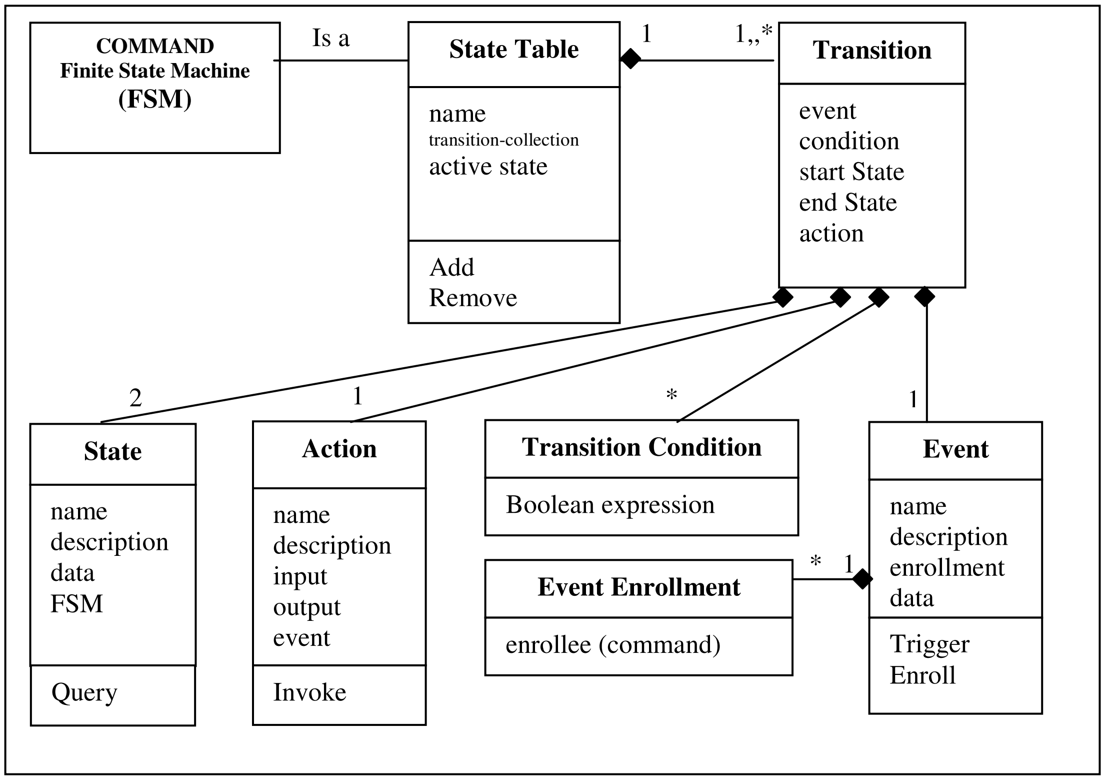
No data are provided in response to this request.

The following table specifies the *OnDataChange* Fault Codes

Code	Description
S_OK	The operation succeeded.
E_FAIL	The operation failed.

Appendix A. Abstract definition of state machine

The basis for the execution of an OPC command is the Finite State Machine (**FSM**). The FSM is an object that precisely describes the behavior of the command in terms of states, transitions, transition conditions, events, and actions. This provides the mechanism to convey information about the method and sequence of the command's execution. The FSM includes the triggering stimulus and resulting actions related to each state transition within the process as well as status and error data associated with resultant states.



FSM UML Description

State Table: A collection of state transitions associated with a Command FSM. The state table identifies all of the steps in the command sequence in terms of transitions between defined states. This provides for an explicit definition of the steps and order of execution.

Transition: A defined change from one state to another in a prescribed sequence, initiated by the occurrence of a defined event. The transition can depend on evaluation of a Boolean expression and can result in an associated action.

Transition Condition: A Boolean expression on which a transition may depend. The transition progresses when the expression evaluates to True.

State: A uniquely identified condition of the command system or subsystem. A state can contain defining state data such as an error code and can also contain a subordinate FSM within its namespace.

Event: A defined, detectable change in the system that can serve as the stimulus to initiate a state transition. An event is identified by a name and may have associated synchronous data. Detection of event conditions can be explicit or implicit, and results in the notification of its enrollees. The creation of a transition object results in an enrollment with an event condition.

Action: Operations performed as the result of a state transition. The action's attributes include input arguments and output arguments. The completion of the Action can result in triggering an associated event to initiate the next state transition. The input arguments can be provided from internal or external sources. Externally supplied arguments are defined as input parameters. The output arguments can be made available to external destinations in which case they are defined as output parameters.

Event Enrollment: An Event Enrollment is the association between an event and a command's state table that contains a transition that is sensitive to that event. Enrollments may be implicit and managed through the creation of the transition object.

Appendix B. Use Cases

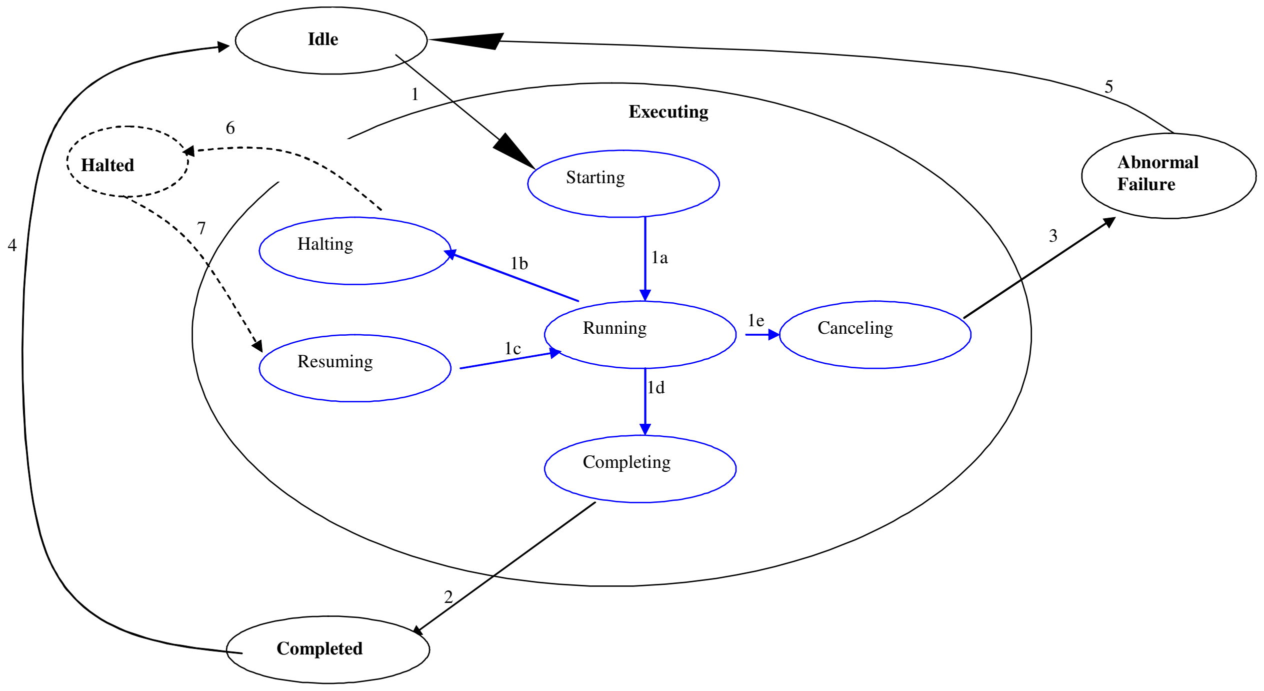
Editors Note: The examples that follow will be edited to remove the XML notation and show the command samples in a more generic format.

B.1. Substate example

Sometimes it is necessary to extend the basic state machine (BSM).

For instance if the transitions between the states of the BSM take a long time, it is useful to provide a state for representing each of the transitions.

An example for this is:



The new states (“Starting”, “Running” Halting”, “Resuming”, “Canceling” and “Completing”) can be seen as sub-states of the super-state “Executing”. The super state is only shown in the graphics above, but not in the description of the state machine.

Transition Collection	Initial State	Resultant State	Event	Trigger	Action
1	Idle	Starting	Invoke	(Async)Invoke	Perform task
1a	Starting	Running	Started	Automatic internal	Update Progress
1b	Running	Halting	Halting	Automatic internal	Update Progress
1c	Resuming	Running	Resumed	Automatic internal	Update Progress
1d	Running	Completing	Completin g	Automatic internal	Update Progress
1e	Running	Canceling	Canceling	ControlComma nd(Suspend)	Update Progress
2	Completing	Completed	Finished	Automatic internal	Update Progress
3	Canceling	Failure	Aborted	Automatic internal	Save error
4	Complete	Idle	Reset	Automatic internal	Return result
5	Failure	Idle	Reset	Automatic internal	Return result
6	Halting	Halted	Halted	Automatic internal	Stop execution
7	Halted	Resuming	Resuming	ControlComma nd(Resume)	Resume execution

The state machine above is based on the default state machine, but the transitions to/from the state of execution (transitions 1a-1e) are modeled as substates to indicate that these transitions may take a long time.

Command Description

```
<GetCMDDescriptionResponse xmlns=http://opcfoundation.org/webservices/OpcCmd/1.0
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation="http://opcfoundation.org/webservices/OpcCmd/1.0 OPCCommands.xsd">
<CMDDescription ExecutionTime="00.20" IsGlobal="false">
    <Description>Example for a state machine with substates</Description>
        <Events>
            <EventDefinition EventName="Invoked" />
            <EventDefinition EventName="Finished" />
            <EventDefinition EventName="Aborted" />
            <EventDefinition EventName="Reset" />
            <EventDefinition EventName="Started" />
            <EventDefinition EventName="Canceling" />
            <EventDefinition EventName="Cancelled" />
            <EventDefinition EventName="Halting" />
            <EventDefinition EventName="Halted" />
            <EventDefinition EventName="Resuming" />
            <EventDefinition EventName="Resumed" />
            <EventDefinition EventName="Completing" />
            <EventDefinition EventName="Running" />
        </Events>
        <States>
            <StateDefinition StateName="Idle" />
            <StateDefinition StateName="Starting" />
            <StateDefinition StateName="Running" />
            <StateDefinition StateName="Halting" />
            <StateDefinition StateName="Halted" />
            <StateDefinition StateName="Resuming" />
            <StateDefinition StateName="Completing" />
            <StateDefinition StateName="Canceling" />
            <StateDefinition StateName="Failure" />
            <StateDefinition StateName="Complete" />
        </States>
        <StateTransitions>
            <StateTransitionDefinition TransitionID="1" StartState="Idle" EndState="Starting"
                EventName="Invoked" />
            <StateTransitionDefinition TransitionID="2" StartState="Completing" EndState="Completed"
                EventName="Finished" />
            <StateTransitionDefinition TransitionID="3" StartState="Canceling" EndState="Failure"
                EventName="Aborted" />
            <StateTransitionDefinition TransitionID="4" StartState="Completed" EndState="Idle"
                EventName="Reset" />
            <StateTransitionDefinition TransitionID="5" StartState="Failure" EndState="Idle"
                EventName="Reset"/>
            <StateTransitionDefinition TransitionID="1a" StartState="Starting" EndState="Running"
                EventName="Started" />
            <StateTransitionDefinition TransitionID="1b" StartState="Running" EndState="Halting"
                EventName="Halting" />
            <StateTransitionDefinition TransitionID="1c" StartState="Resuming" EndState="Running"
                EventName="Resumed" />
            <StateTransitionDefinition TransitionID="1d" StartState="Running" EndState="Completing"
                EventName="Completing" />
            <StateTransitionDefinition TransitionID="1e" StartState="Running" EndState="Cancelling"
                EventName="Canceling" />
            <StateTransitionDefinition TransitionID="6" StartState="Halting" EndState="Halted"
                EventName="Halted" />
            <StateTransitionDefinition TransitionID="7" StartState="Halted" EndState="Resuming"
                EventName="Resuming" />
        </StateTransitions>
        <SupportedControlCommands>
            <ControlCommand>Suspend</ControlCommand>
            <ControlCommand>Resume</ControlCommand>
            <ControlCommand>Cancel</ControlCommand>
        </SupportedControlCommands>
    </CMDDescription>
</GetCMDDescriptionResponse>
```

B.2. Basic Command Sample (NC Cancel Alarm)

This example demonstrates a basic command, in this case an NC alarm reset/cancel command. The alarm reset command requires no input or output arguments. The command has no state extensions. The example command syntax is XML.

Command Description

```
<Command    CommandName= "NcResetAlarm"  Description= "Resets any active alarms"
    ExecutionTime="00.200">

    <EventDefinition EventName= "Invoke"/>
    <EventDefinition EventName= "Aborted"/>
    <EventDefinition EventName= "Finished"/>

    <StateDefinition StateName= "Idle"/>
    <StateDefinition StateName= "Executing"/>
    <StateDefinition StateName= "Failure"/>
    <StateDefinition StateName= "Completed"/>

    <SupportedControls>
    </SupportedControls>
</Command>
```

Synchronous Invocation

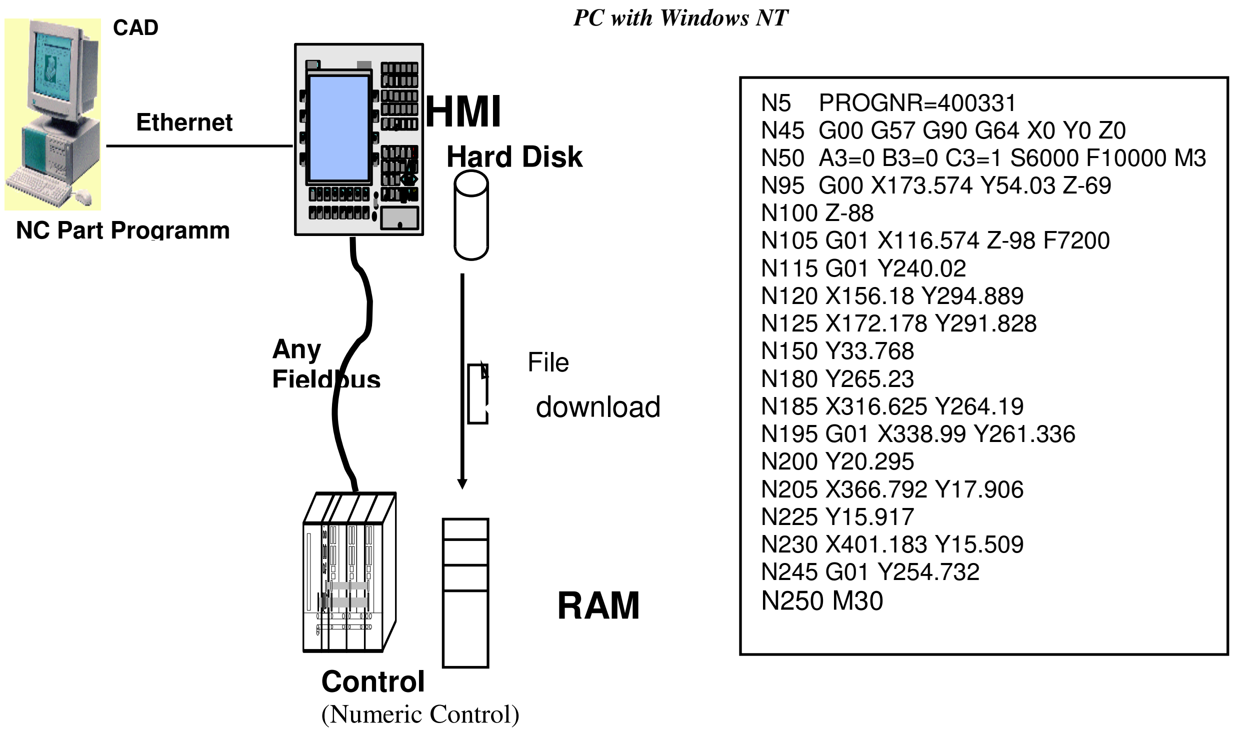
```
<SyncInvoke CommandName="NcResetAlarm" CommandUri="NCCommands" InvokeUid ="1234567">
</SyncInvoke>
```

Synchronous Response

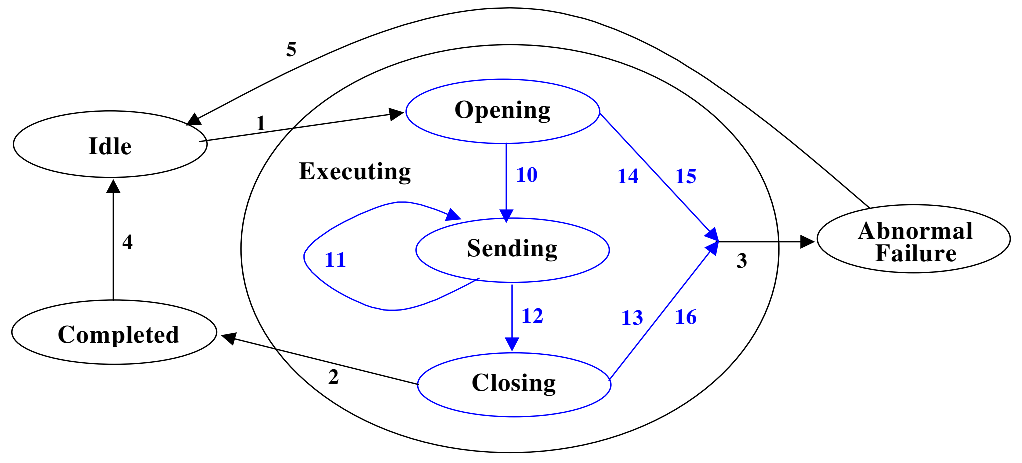
```
 <SyncResponse InvokeUid ="1234567" hres="00000000">
    <StateChangeEvent EventName="Invoke" EventTime="2003-05-27T15:18:22.284Z" OldState="Idle"
    NewState="Executing" />
    <StateChangeEvent EventName="Finished" EventTime="2003-05-27T15:18:22.484Z"
    OldState="Executing" NewState="Complete" />
</SyncResponse>
```

B.3. NC Program Download (Domain Services)

In this example, a new NC part program is generated out of a CAD system and needs to be transfered to the HMI. From there it must be downloaded to the NC Controller to be processed. The download may need several minutes or it may fail. Therefore we need feedback from the server about the transfer process and about the quality. In some cases (huge programs) it is necessary to process the part program from external storage. This means that the program remains on the Hard Disk and is downloaded part by part.The following example illustrates use of synchronous and asynchronious OPC commands to perform a program copy from a source location such as an HMI to a destination such as an NC controller. The example command syntax is XML.



```
N5    PROGNR=400331
N45  G00 G57 G90 G64 X0 Y0 Z0
N50  A3=0 B3=0 C3=1 S6000 F10000 M3
N95  G00 X173.574 Y54.03 Z-69
N100 Z-88
N105 G01 X116.574 Z-98 F7200
N115 G01 Y240.02
N120 X156.18 Y294.889
N125 X172.178 Y291.828
N150 Y33.768
N180 Y265.23
N185 X316.625 Y264.19
N195 G01 X338.99 Y261.336
N200 Y20.295
N205 X366.792 Y17.906
N225 Y15.917
N230 X401.183 Y15.509
N245 G01 Y254.732
N250 M30
```



NcCopyProgram State Model (FSM)

Transition Collection	Initial State	Resultant State	Event	Trigger	Action
1	Idle	Executing. Opening	Invoke	(Async)Invoke	Perform task
2	Executing. Closing	Complete	Finished	Automatic internal	Update Progress
3	Executing. Closing	Failure	Aborted	Automatic internal	Save error
4	Complete	Idle	Reset	Automatic internal	Return result
5	Failure	Idle	Reset	Automatic internal	Return result
10	Executing. Opening	Executing. Sending	Opened	Automatic internal	Start Sending
11	Executing. Sending	Executing. Sending	Data Sent	Automatic internal	Update Progress
12	Executing. Sending	Executing. Closing	Done Sending	(internal)/ Cancel Cmd	Update Progress
13	Executing. Closing	Failure	Cancelled	Cancel Command	Cancel Execution
14	Executing. Opening	Failure	Cancelled	Cancel Command	Cancel Execution
15	Executing. Opening	Failure	Aborted	Automatic internal	Save error
16	Executing. Sending	Closing	Aborted	Automatic internal	Abort Execution

Command Description

```
<Command CommandName="NcCopyProgram" Description="Copy an NC Program"
ExecutionTime="00.00">
    <EventDefinition EventName="Invoke"/>
    <EventDefinition EventName="Cancelled"/>
    <EventDefinition EventName="Aborted"/>
    <EventDefinition EventName="Finished"/>
    <EventDefinition EventName="Opened"/>
    <EventDefinition EventName="DataSent" EventData="double"/>
    <EventDefinition EventName="DoneSending"/>

    <StateDefinition StateName="Idle"/>
    <StateDefinition StateName="Executing.Opening"/>
    <StateDefinition StateName="Executing.Sending"/>
    <StateDefinition StateName="Executing.Closing"/>
    <StateDefinition StateName="Failure"/>
    <StateDefinition StateName="Complete"/>

    <StateTransition Id="1" StartState="Idle" EndState="Executing.Opening" EventName="Invoke"/>
    <StateTransition Id="2" StartState="Executing.Closing" EndState="Completed" EventName="Finished"/>
    <StateTransition Id="3" StartState="Executing.Closing" EndState="Failure" EventName="Aborted"/>
    <StateTransition Id="4" StartState="Complete" EndState="Idle" EventName="Reset"/>
    <StateTransition Id="5" StartState="Failure" EndState="Idle" EventName="Reset"/>

    <StateTransition Id="10" StartState="Executing.Opening" EndState="Executing.Sending"
    EventName="Opened"/>
    <StateTransition Id="11" StartState="Executing.Sending" EndState="Executing.Sending"
    EventName="DataSent" TriggerEvent="DataSent" Action="UpdateProgress"/>
    <StateTransition Id="12" StartState="Executing.Sending" EndState="Executing.Closing"
    EventName="DoneSending" TriggerEvent="DataSent" Action="UpdateProgress"/>
    <StateTransition Id="13" StartState="Executing.Closing" EndState="Failure" EventName="Cancelled"/>
    <StateTransition Id="14" StartState="Executing.Opening" EndState="Failure" EventName="Cancelled"/>
    <StateTransition Id="15" StartState="Executing.Opening" EndState="Failure" EventName="Aborted"/>
    <StateTransition Id="16" StartState="Executing.Sending" EndState="Executing.Closing"
    EventName="Aborted"/>

    <ActionDefinition ActionName="UpdateProgress" EventName="DataSent">
        <OutArgument Name="BytesCopied" Type="double"/>
        <OutArgument Name="Percentage" Type="integer"/>
    </ActionDefinition>

    <SupportedControls>
        <Control Name="Cancel"/>
    </SupportedControls>

    <InParameter Name="Source" ValueType="string"/>
    <InParameter Name="Destination" ValueType="string"/>
    <InParameter Name="Options" ValueType="enumeration"/>
</Command>
```

Command Synchronous Invoke

```
<SyncInvoke CommandName=" NcCopyProgram " CommandUri="NCCommands" >
    <InArgumentList>
        <Source>"c:\programs\abc.pgm"</Source>
        <Destination> "mpf\abc.pgm"</Destination>
        <Options>"Overwrite"</Options>
    </InArgumentList>

    <EventFilter>
        <StateTransition Id="2" StartState="Executing.Closing" EndState="Complete" EventName="Finished"/>
        <StateTransition Id="3" StartState=" Executing.Closing " EndState="Failure" EventName="Aborted"/>
        <StateTransition Id="10" StartState="Executing.Opening" EndState="Executing.Sending"
        EventName="Opened"/>
        <StateTransition Id="12" StartState="Executing.Sending" EndState="Executing.Closing"
        EventName="DoneSending" TriggerEvent="DataSent" Action="UpdateProgress"/>
        <StateTransition Id="13" StartState="Executing.Closing" EndState="Failure" EventName="Cancelled"/>
        <StateTransition Id="14" StartState="Executing.Opening" EndState="Failure" EventName="Cancelled"/>
        <StateTransition Id="15" StartState="Executing.Opening" EndState="Failure" EventName="Aborted"/>
        <StateTransition Id="16" StartState="Executing.Sending" EndState="Executing.Closing"
        EventName="Aborted"/>
    </EventFilter>
</SyncInvoke>
```

Command Synchronous Response

```
<SyncResponse >
    <StateChangeEvent EventName="Opened" EventTime="2003-05-27T15:18:22.123Z"
        OldState="Executing.Opening"  NewState="Executing.Sending"/>
    <StateChangeEvent EventName="DoneSending" EventTime="2003-05-27T15:19:22.456Z"
        OldState="Executing.Sending"  NewState="Executing.Closing">
        <EventData>
            <UpdateProgress>
                <BytesCopied>2345.0</BytesCopied>
                <Percentage>100</Percentage>
            </UpdateProgress>
        </EventData>
    </StateChangeEvent>
    <StateChangeEvent EventName="Finished" EventTime="2003-05-27T15:18:23.789Z"
        OldState="Executing.Closing" NewState="Complete"/>

</SyncResponse>
```

Command Asynchronous Invoke

```
<AsyncInvoke CommandName=" NcCopyProgram " CommandUri="NCCommands" InvokeUid="1234567">
    <InArgumentList>
        <Source>"c:\programs\abc.pgm"</Source>
        <Destination> "mpf\abc.pgm"</Destination>
        <Options>"Overwrite"</Options>
    </InArgumentList>
    <EventFilter>
        <StateTransition Id="2" StartState="Executing.Closing" EndState="Complete" EventName="Finished"/>
        <StateTransition Id="3" StartState=" Executing.Closing " EndState="Failure" EventName="Aborted"/>
        <StateTransition Id="10" StartState="Executing.Opening" EndState="Executing.Sending"
        EventName="Opened"/>
        <StateTransition Id="12" StartState="Executing.Sending" EndState="Executing.Closing"
        EventName="DoneSending" TriggerEvent="DataSent" Action="UpdateProgress"/>
        <StateTransition Id="13" StartState="Executing.Closing" EndState="Failure" EventName="Cancelled"/>
        <StateTransition Id="14" StartState="Executing.Opening" EndState="Failure" EventName="Cancelled"/>
        <StateTransition Id="15" StartState="Executing.Opening" EndState="Failure" EventName="Aborted"/>
        <StateTransition Id="16" StartState="Executing.Sending" EndState="Executing.Closing"
        EventName="Aborted"/>
    </EventFilter>
    <CallbackId>"IOnStateChange"</CallbackId>
</AsyncInvoke>
```

Command Asynchronous Response

```
<AsyncResponse InvokeUid="1234567" hres="00000000">
</AsyncResponse>
```

Command OnStateChange Request/Response Sequence

```
<OnStateChangeReq InvokeUid="1234567">
    <StateChangeEvent EventName="Opened" EventTime="2003-05-27T15:18:22.123Z"
    OldState="Executing.Opening"  NewState="Executing.Sending"/>

    <UpdateProgress>
        <BytesCopied>0.0</BytesCopied>
        <Percentage>0</Percentage>
    </UpdateProgress>

    <PermittedControls>
        <Control Name="Cancel"/>
        <Control Name="Suspend"/>
    </PermittedControls>
</OnStateChangeReq>

<OnStateChangeRsp InvokeUid="1234567" hres="00000000">
</OnStateChangeRsp>

<OnStateChangeReq InvokeUid="1234567">
    <StateChangeEvent EventName="DataSent" EventTime="2003-05-27T15:18:23.456Z"
    OldState="Executing.Sending" NewState="Executing.Sending"/>

    <UpdateProgress>
        <BytesCopied>750.0</BytesCopied>
        <Percentage>31</Percentage>
    </UpdateProgress>

    <PermittedControls>
        <Control Name="Cancel"/>
    </PermittedControls>
</OnStateChangeReq>

<OnStateChangeRsp InvokeUid="1234567" hres="00000000">
```

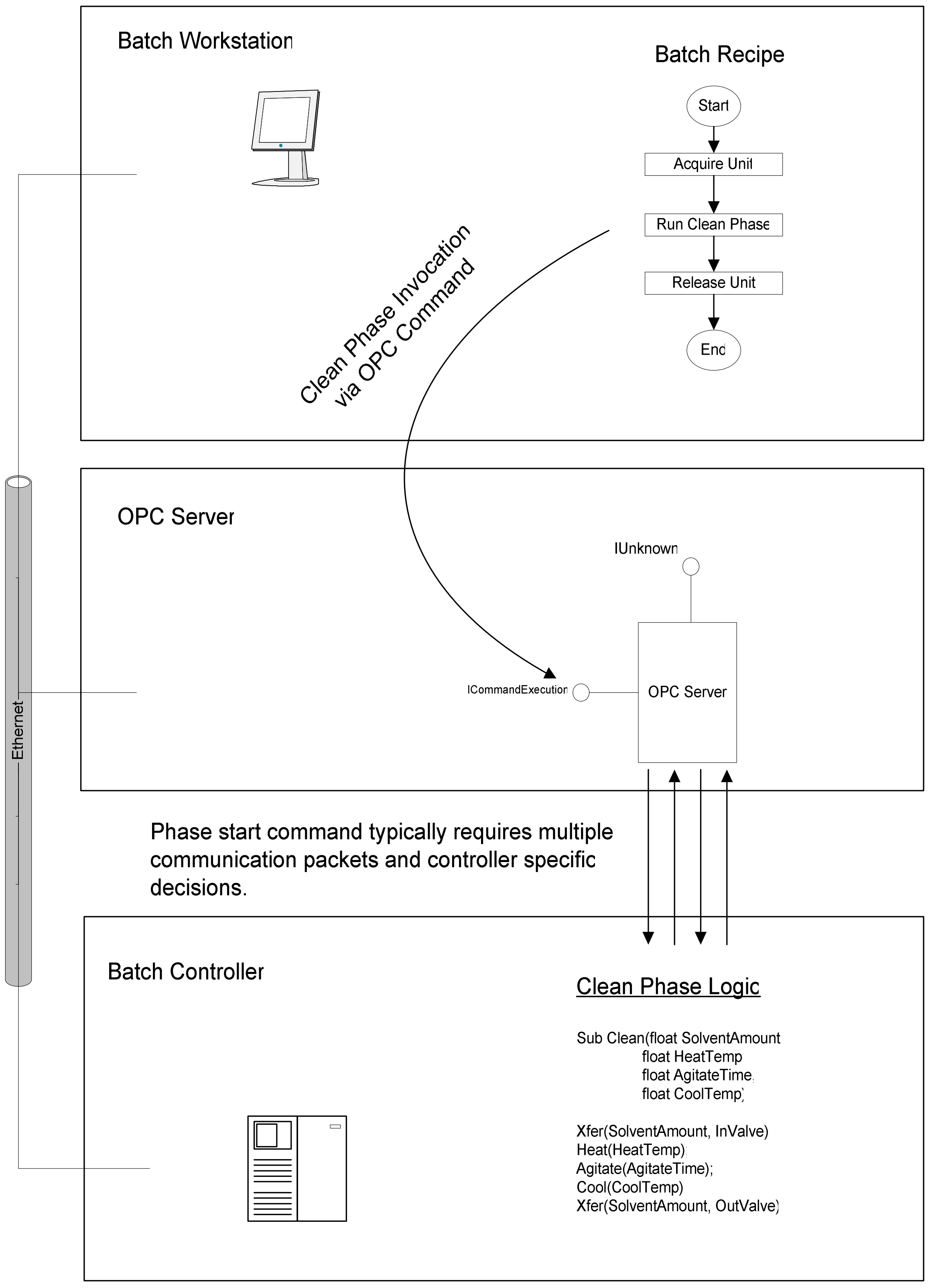
```
</OnStateChangeRsp>

<OnStateChangeReq InvokeUid="1234567">
    <StateChangeEvent EventName="DoneSending" EventTime="2003-05-27T15:18:24.456Z"
    OldState="Executing.Sending"  NewState="Executing.Closing"/>

    <UpdateProgress>
        <BytesCopied>2345.0</BytesCopied>
        <Percentage>100</Percentage>
    </UpdateProgress>

    <PermittedControls>
        <Control Name="Cancel"/>
    </PermittedControls>
</OnStateChangeReq>

<OnStateChangeRsp InvokeUid="1234567" hres="00000000">
</OnStateChangeRsp>
```

Command Cancel Invoke

```
<AsyncControlReq InvokeUid="1234567" Control="Cancel">
</AsyncControlReq>

<OnStateChangeReq InvokeUid="1234567">
    <StateChangeEvent EventName="Cancelled" EventTime="2003-05-27T15:19:22.456Z"
    OldState="Executing.Sending" NewState="Executing.Closing"/>
    <StateChangeEvent EventName="Cancelled" EventTime="2003-05-27T15:19:24.456Z"
    OldState="Executing.Closing" NewState="Failure"/>

    <PermittedControls>
    </PermittedControls>
</OnStateChangeReq>

<AsyncControlRsp InvokeUid="1234567" Control="Cancel" hres="00000000">
</AsyncControlRsp>
```

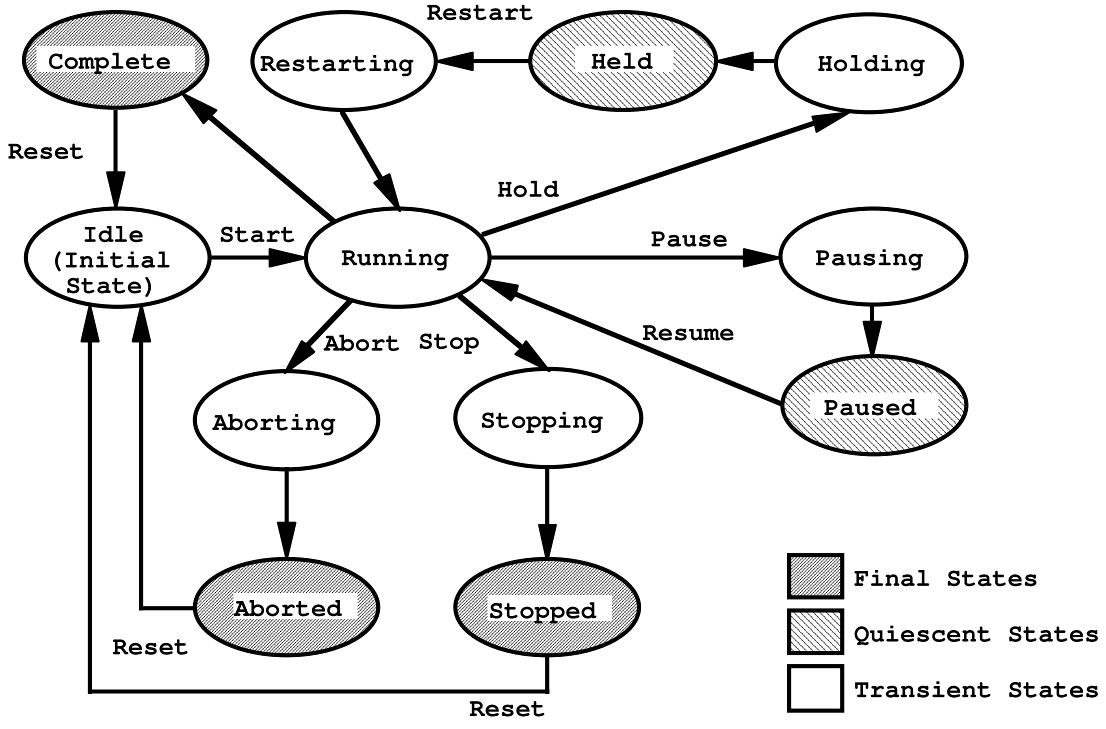
B.4. Batch Phase Start Example

In this example a Batch Phase will run in an embedded controller. The start command will come from Batch Manager Software executing on a workstation. The command is executed in the OPC Command server. The use of Industry Standard OPC Commands to control Batch processes allows vendors to develop Batch Manager Software independent of the underlying embedded Batch Controller. The example illustrates the use of asynchronous OPC commands.



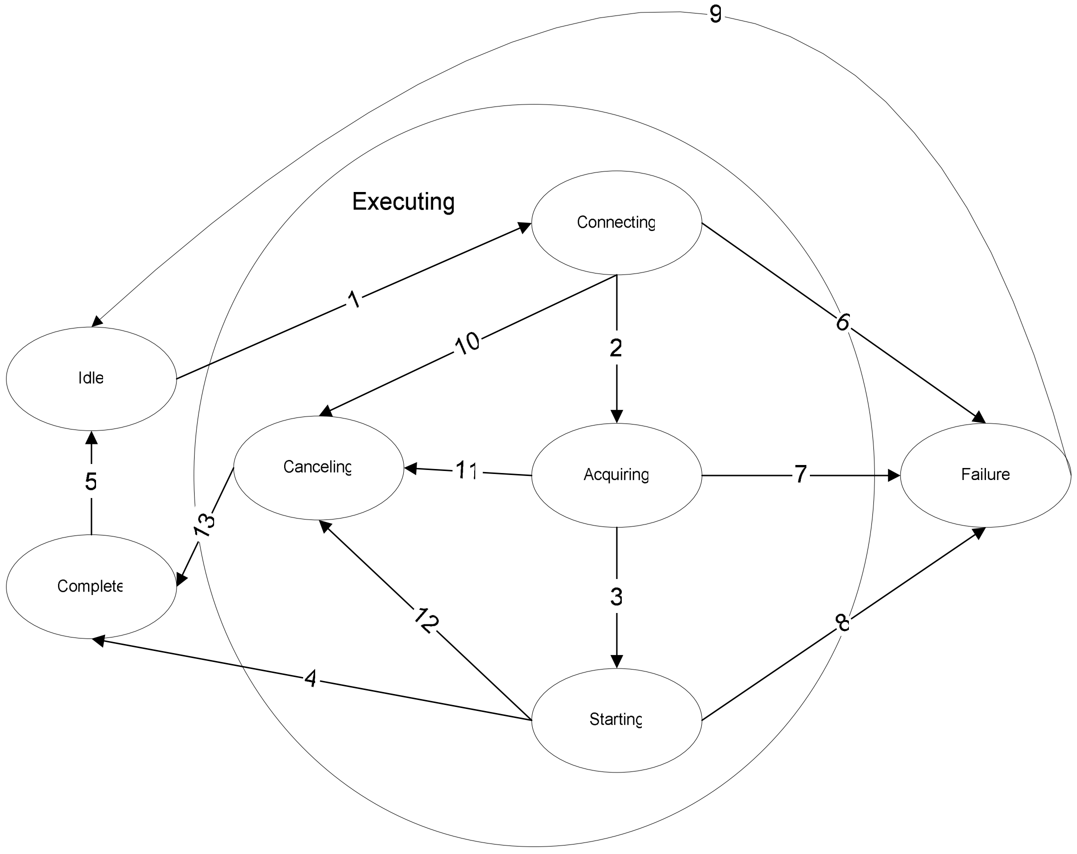
Start Phase Command Overview

To better understand the Phase Start command the S88 state transition diagram is presented first. The start phase command transitions the phase from the "Idle" state to the "Running" state within the S88 state transition diagram. The S88 state model is more theoretical than realistic. In a typical architecture the phase runs in an embedded controller after the Start Command is issued from workstation based Batch manager software. Communication failures, arbitration, user interaction, timing and other factors can result in a very long or failed transition. As a result, a realistic Start transition is modeled and implemented as a sub state transition diagram.



S88 Phase State Transition Diagram

The Start Phase Command Finite State Machine(FSM) presented below illustrates how the OPC Command Server implements the execution of the Start Phase command.



StartPhase State Model (FSM)

Transition Collection	Initial State	Resultant State	Event	Trigger	Action
1	Idle	Executing. Connecting	Invoke	(Async)Invoke	Perform task
2	Executing. Connecting	Executing. Acquiring	Connected	(internal)	Update Progress
3	Executing. Acquiring	Executing. Starting	Acquired	(internal)	Update Progress
4	Executing. Starting	Complete	Started	(internal)	Update Progress
5	Complete	Idle	Reset	(internal)	Return result
6	Executing. Connecting	Failure	Aborted	(internal)	Store error information
7	Executing. Acquiring	Failure	Aborted	(internal)	Store error information
8	Executing. Starting	Failure	Aborted	(internal)	Store error information
9	Failure	Idle	Reset	(internal)	Return result
10	Executing. Connecting	Canceling	Cancelled	Cancel Command	Cancel Execution
11	Executing. Acquiring	Canceling	Cancelled	Cancel Command	Cancel Execution
12	Executing. Starting	Canceling	Cancelled	Cancel Command	Cancel Execution
13	Canceling	Complete	CancelComplete	(internal)	Set result as command cancelled

Start Phase Command Description

```
<Command CommandName="StartPhase" Description="Start Batch Phase" ExecutionTime="00.00">

  <InParameter Name="Equipment" ValueType="string"/>
  <InParameter Name="EquipmentName" ValueType="string"/>
  <InParameter Name="CampaignID" ValueType="string "/>
  <InParameter Name="LotID" ValueType="string"/>
  <InParameter Name="BatchID" ValueType="string"/>
  <InParameter Name="PhaseName" ValueType=" string"/>
  <InParameter Name="ProcedurePath" ValueType="string"/>
  <InParameter Name="ModuleIndex" ValueType="string"/>
  <InParameter Name="PhaseParameters" ValueType="ArrayOfArgument"/>

  <EventDefinition EventName="Invoke"/>
  <EventDefinition EventName="Connected"/>
  <EventDefinition EventName="Acquired"/>
  <EventDefinition EventName="Started"/>
  <EventDefinition EventName="Reset"/>
  <EventDefinition EventName="Aborted/>
  <EventDefinition EventName="Cancelled"/>
  <EventDefinition EventName="CancelComplete"/>
  <StateDefinition StateName="Idle"/>
  <StateDefinition StateName="Executing.Connecting"/>
  <StateDefinition StateName="Executing.Acquiring"/>
  <StateDefinition StateName="Executing.Starting"/>
  <StateDefinition StateName="Executing.Canceling"/>
  <StateDefinition StateName="Failure"/>
  <StateDefinition StateName="Complete"/>

  <StateTransition Id="1" StartState="Idle" EndState="Executing.Connecting" EventName="Invoke"/>
  <StateTransition Id="2" StartState="Executing.Connecting " EndState=" Executing.Acquiring"
EventName="Connected" TriggerEvent="ConnectionMade" Action="UpdateProgress"/>
  <StateTransition Id="3" StartState="Executing.Acquiring" EndState="Executing.Starting"
EventName="Acquired" TriggerEvent="UnitAcquired" Action="UpdateProgress"/>
  <StateTransition Id="4" StartState="Executing.Starting" EndState="Complete" EventName="Started"
TriggerEvent="PhaseStarted" Action="UpdateProgress"/>
  <StateTransition Id="5" StartState="Complete" EndState="Idle" EventName="Reset"/>
  <StateTransition Id="6" StartState="Executing.Connecting" EndState="Failure" EventName="Aborted"/>
  <StateTransition Id="7" StartState="Executing.Acquiring" EndState="Failure" EventName="Aborted"/>
  <StateTransition Id="8" StartState="Executing.Starting" EndState="Failure" EventName="Aborted"/>
  <StateTransition Id="9" StartState="Failure" EndState="Idle" EventName="Reset"/>
  <StateTransition Id="10" StartState="Executing.Connecting" EndState="Executing.Canceling"
EventName="Cancelled"/>
  <StateTransition Id="11" StartState="Executing.Acquiring" EndState="Executing.Canceling"
EventName="Cancelled"/>
  <StateTransition Id="12" StartState="Executing.Starting" EndState="Executing.Canceling"
EventName="Cancelled"/>
  <StateTransition Id="13" StartState="Executing.Canceling" EndState="Complete"
EventName="CancelComplete"/>
  <ActionDefinition ActionName="UpdateProgress" EventName="Connected"/>
  <ActionDefinition ActionName="UpdateProgress" EventName="Acquired"/>
  <ActionDefinition ActionName="UpdateProgress" EventName="Started"/>

  <SupportedControls>
    <Control Name="Cancel"/>
  </SupportedControls>

</Command>
```

Asynchronous Invoke

```
<AsyncInvoke CommandName="StartPhase" CommandUri="BatchCommands" InvokeID="123456789">
  <InArgumentList>
    <Equipment>{9F3C4E6E-FF10-4ae4-B745-66A4D4056085}</Equipment>
    <EquipmentName>Reactor1</EquipmentName>
    <CampaignID>Campaign123</CampaignID>
    <LotID>Lot123</LotID>
    <BatchID>Batch123</BatchID>
    <PhaseName>Clean</PhaseName>
    <ProcedurePath>Procedure1.3.1</ProcedurePath>
    <ModuleIndex>1</ModuleIndex>
    <PhaseParameters>
      <Parameter Name="Param1" Value="1.0"/>
       <Parameter Name="Param2" Value="Unit2"/>
      <Parameter Name="Param3" Value="3"/>
    </PhaseParameters>
  </InArgumentList>
  <EventFilter>
    <StateTransition Id="5" StartState="Complete" EndState="Idle" EventName="Reset"/>
  </EventFilter>

  <CallbackID>IOPCStateChange</CallbackID>
</AsyncInvoke>
```

Command Asynchronous Response

```
<AsyncInvokeResponse InvokeID="123456789" RevisedUpdateFrequency="0"/>
```

Command OnStateChange Request

```
<OnStateChange InvokeUid="123456789">
  <StateData EventName="Opened" EventTime="2004-03-04T10:57:29.123Z" OldState="Complete"
  NewState="Idle"/>
</OnStateChange>
```

Command OnStateChange Response

```
<OnStateChangeResponse InvokeUid="123456789" Result ="0"/>
```

Appendix C. COM/DCOM Mapping

This appendix defines the mapping of the command services to COM.

C.1. Implementation Guidelines

C.1.1. Error Handling

When a method as a whole fails, errors will be returned in the HRESULT return value. If the method completely succeeds it will return S_OK.

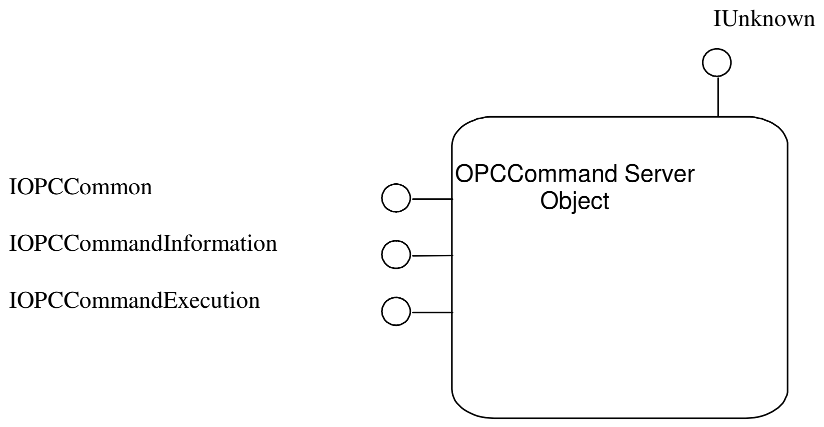
Error codes for each service are listed as part of the response messages in the sections that describe those services. As a convention, success codes begin with "S_" and error codes begin with "E_".

C.1.2. Locale Ids and Verbose Errors

COM clients will use the standard IOPCCommon interface to set the desired localeID and retrieve verbose error information for any of the HRESULTs.

C.2. Server-Side Interfaces

The interfaces described here are mandatory interfaces of the OPC CommandServer object. An OPCCommandServer Object can be restricted to the functionality as described in this document.



C.2.1. IOPCCommandInformation

C.2.2. IOPCCommandExecution

C.3. Client-Side Interfaces

C.3.1. IOPCStateChange

C.3.1.1 IOPCStateChange::OnStateChange

C.3.1.2 OPC Command IDL

C.3.1.3 OPCCommand IDL