

DIN EN 62541-10

ICS 35.200; 35.240.50

**OPC Unified Architecture –
Teil 10: Programme
(IEC 62541-10:2012);
Englische Fassung EN 62541-10:2012**

OPC unified architecture –
Part 10: Programs
(IEC 62541-10:2012);
English version EN 62541-10:2012

Architecture unifiée OPC –
Partie 10: Programmes
(CEI 62541-10:2012);
Version anglaise EN 62541-10:2012

Gesamtumfang 53 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2012-08-31 angenommene Europäische Norm als DIN-Norm ist 2013-06-01.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN IEC 62541-10:2008-12.

Für dieses Dokument ist das nationale Arbeitsgremium K 931 „Systemaspekte“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE (www.dke.de) zuständig.

Die enthaltene IEC-Publikation wurde vom SC 65E „Devices and integration in enterprise systems“ erarbeitet.

Das IEC-Komitee hat entschieden, dass der Inhalt dieser Publikation bis zu dem Datum (stability date) unverändert bleiben soll, das auf der IEC-Website unter „<http://webstore.iec.ch>“ zu dieser Publikation angegeben ist. Zu diesem Zeitpunkt wird entsprechend der Entscheidung des Komitees die Publikation

- bestätigt,
- zurückgezogen,
- durch eine Folgeausgabe ersetzt oder
- geändert.

Für den Fall einer undatierten Verweisung im normativen Text (Verweisung auf eine Norm ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils neueste gültige Ausgabe der in Bezug genommenen Norm.

Für den Fall einer datierten Verweisung im normativen Text bezieht sich die Verweisung immer auf die in Bezug genommene Ausgabe der Norm.

Der Zusammenhang der zitierten Normen mit den entsprechenden Deutschen Normen ergibt sich, soweit ein Zusammenhang besteht, grundsätzlich über die Nummer der entsprechenden IEC-Publikation. Beispiel: IEC 60068 ist als EN 60068 als Europäische Norm durch CENELEC übernommen und als DIN EN 60068 ins Deutsche Normenwerk aufgenommen.

Das Präsidium des DIN hat mit Präsidialbeschluss 1/2004 festgelegt, dass DIN-Normen, deren Inhalt sich auf internationale Arbeitsergebnisse der Informationsverarbeitung gründet, unter bestimmten Bedingungen allein in englischer Sprache veröffentlicht werden dürfen. Diese Bedingungen sind für die vorliegende Norm erfüllt.

Da sich die Benutzer der vorliegenden Norm der englischen Sprache als Fachsprache bedienen, wird die englische Fassung der EN 62541-10 veröffentlicht. Zu deren Abschnitt 3, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als informativer Nationaler Anhang NA der vorliegenden Norm hinzugefügt. Für viele der verwendeten Begriffe existieren keine gebräuchlichen deutschen Benennungen, da sich die deutschen Anwender in der Regel ebenfalls der englischen Benennungen bedienen. Diese Norm steht nicht in unmittelbarem Zusammenhang mit Rechtsvorschriften und ist nicht als Sicherheitsnorm anzusehen.

Nationaler Anhang NA (informativ)

3.1 Begriffe

Für die Anwendung dieses Dokuments gelten die Begriffe nach IEC/TR 62541-1 und IEC 62541-3 und die folgenden Begriffe.

3.1.1

Funktion

(en: **function**)

programmtechnische Aufgabenstellung, ausgeführt auf einem Server oder Gerät, gewöhnlich durch Maschinencode

3.1.2

endlicher Zustandsautomat

(en: **finite State Machine**)

Sequenz von Zuständen und gültigen Zustandsübergängen zusammen mit den Auslösern und Wirkungen dieser Zustandsübergänge, die die Aktionen eines Programms im Sinne von diskreten Phasen definieren

3.1.3

ProgramType

(en: **ProgramType**)

Objektypknoten, der die Typdefinition eines Programms repräsentiert und ein Subtyp des FinitStateMachineType ist

3.1.4

Programmsteuerungsmethode

(en: **Program Control Method**)

Methode, definiert in dieser Spezifikation mit spezifischer Semantik und angelegt zur Kontrolle eines Programms durch Auslösen eines Zustandsübergangs

3.1.5

Programmaufruf

(en: **Program Invocation**)

eindeutige Objektinstanz eines Programms, die auf einem Server existiert

ANMERKUNG 1 zum Begriff: Ein Programmaufruf unterscheidet sich von anderen Objektinstanzen desselben ProgramType durch den eindeutigen Suchpfad (en: browse path) des Knotens.

– Leerseite –

**OPC unified architecture -
Part 10: Programs
(IEC 62541-10:2012)**

Architecture unifiée OPC -
Partie 10: Programmes
(CEI 62541-10:2012)

OPC Unified Architecture -
Teil 10: Programme
(IEC 62541-10:2012)

This European Standard was approved by CENELEC on 2012-08-31. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Management Centre: Avenue Marnix 17, B - 1000 Brussels

Foreword

The text of document 65E/244/FDIS, future edition 1 of IEC 62541-10, prepared by SC 65E "Devices and integration in enterprise systems" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 62541-10:2012.

The following dates are fixed:

- latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement (dop) 2013-05-31
- latest date by which the national standards conflicting with the document have to be withdrawn (dow) 2015-08-31

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 62541-10:2012 was approved by CENELEC as a European Standard without any modification.

CONTENTS

INTRODUCTION.....	4
1 Scope	5
2 Normative references	5
3 Terms, definitions and abbreviations	5
3.1 Terms and definitions	5
3.2 Abbreviations	6
4 Concepts.....	6
4.1 General.....	6
4.2 Programs	7
4.2.1 Overview	7
4.2.2 Program Finite State Machine	8
4.2.3 Program States	9
4.2.4 State Transitions.....	9
4.2.5 Program State Transition Stimuli	10
4.2.6 Program Control Methods.....	10
4.2.7 Program State Transition Effects	11
4.2.8 Program Result Data.....	11
4.2.9 Program Lifetime.....	11
5 Model.....	12
5.1 General.....	12
5.2 ProgramType.....	13
5.2.1 Overview	13
5.2.2 ProgramType Properties	15
5.2.3 ProgramType Components.....	15
5.2.4 ProgramType Causes (Methods)	21
5.2.5 ProgramType Effects (Events).....	22
5.2.6 AuditProgramTransitionEventType.....	25
5.2.7 FinalResultData	26
5.2.8 ProgramDiagnosticType	26
Annex A (informative) Program Example	28
Figure 1 – Automation facility control.....	7
Figure 2 – Program illustration	8
Figure 3 – Program States and Transitions.....	9
Figure 4 – Program Type	13
Figure 5 – Program FSM References.....	16
Figure 6 – ProgramType Causes and Effects.....	21
Figure A.1 – Program Example.....	28
Figure A.2 – DomainDownload State Diagram.....	29
Figure A.3 – DomainDownloadType Partial State Model.....	36
Figure A.4 – ReadyToRunning Model	39
Figure A.5 – OpeningToSending ToClosing Model	41

Figure A.6 – RunningToSuspended Model	42
Figure A.7 – SuspendedToRunning Model	43
Figure A.8 – RunningToHalted – Aborted Model	44
Figure A.9 – SuspendedToAborted Model	45
Figure A.10 – RunningToCompleted Model	46
Figure A.11 – Sequence of Operations	47
Table 1 – Program Finite State Machine	8
Table 2 – Program States	9
Table 3 – Program State Transitions	10
Table 4 – Program Control Methods	10
Table 5 – ProgramType	14
Table 6 – Program States	17
Table 7 – Program Transitions	18
Table 8 – ProgramType Causes	21
Table 9 – ProgramTransitionEventType	23
Table 10 – ProgramTransitionEvents	24
Table 11 – AuditProgramTransitionEventType	25
Table 12 – ProgramDiagnosticType	26
Table A.1 – DomainDownload States	30
Table A.2 – DomainDownload Type	32
Table A.3 – Transfer State Machine Type	33
Table A.4 – Transfer State Machine – States	34
Table A.5 – Finish State Machine Type	34
Table A.6 – Finish State Machine – States	35
Table A.7 – DomainDownload Type Property Attributes Variable Values	35
Table A.8 – Additonal DomainDownload Transition Types	37
Table A.9 – Start Method Additions	39
Table A.10 – StartArguments	40
Table A.11 – Intermediate Results Object	41
Table A.12 – Intermediate Result Data Variables	42
Table A.13 – Final Result Data	45
Table A.14 – Final Result Variables	46

INTRODUCTION

This International Standard is a specification intended for developers of OPC UA applications. The specification is a result of an analysis and design process to develop a standard interface to facilitate the development of applications by multiple vendors that inter-operate seamlessly together.

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

1 Scope

This part of the IEC 62541 series specifies the standard representation of *Programs* as part of the OPC Unified Architecture and its defined information model. This includes the description of the *NodeClasses*, standard *Properties*, *Methods* and *Events* and associated behaviour and information for *Programs*.

The complete address space model including all *NodeClasses* and *Attributes* is specified in IEC 62541-3. The services such as those used to invoke the *Methods* used to manage *Programs* are specified in IEC 62541-4.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC/TR 62541-1, *OPC Unified Architecture – Part 1: Overview and Concepts*

IEC 62541-3:2010, *OPC unified architecture – Part 3: Address Space Model*

IEC 62541-4:2011, *OPC unified architecture – Part 4: Services*

IEC 62541-5, *OPC unified architecture – Part 5: Information Model*

IEC 62541-7, *OPC unified architecture – Part 7: Profiles*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC/TR 62541-1 and IEC 62541-3, as well as the following apply.

3.1.1

function

programmatic task performed at a server or device, usually accomplished by computer code execution

3.1.2

finite State Machine

sequence of states and valid state transitions along with the causes and effects of those state transitions that define the actions of a *Program* in terms of discrete stages

3.1.3

ProgramType

ObjectType Node that represents the type definition of a *Program* and is a subtype of the *FiniteStateMachineType*

3.1.4

Program Control Method

Method specified by this specification having specific semantics designed for the control of a *Program* by causing a state transition

3.1.5

Program Invocation

unique *Object* instance of a *Program* existing on a *Server*

Note 1 to entry: The Program Invocation is distinguished from other Object instances of the same ProgramType by the object node's unique browse path.

3.2 Abbreviations

API	Application Programming Interface
DA	Data Access
FSM	Finite State Machine
HMI	Human Machine Interfaces
PCM	Program Control Method
PGM	Program
PI	Program Invocation
PLC	Programmable Logic Controller
UA	Unified Architecture
UML	Unified Modelling Language

4 Concepts

4.1 General

Integrated automation facilities manage their operations through the exchange of data and coordinated invocation of system functions like illustrated in Figure 1. *Services* are required to perform the data exchanges and to invoke the functions that constitute system operation. These functions may be invoked through human machine interfaces, cell controllers, or other supervisory control and data acquisition type systems. OPC UA defines *Methods* and *Programs* as an interoperable way to advertise, discover, and request these functions. They provide a normalizing mechanism for the semantic description, invocation of, and result reporting of these functions. Together *Methods* and *Programs* complement the other OPC UA *Services* and *ObjectTypes* to facilitate the operation of an automation environment using a client server hierarchy.

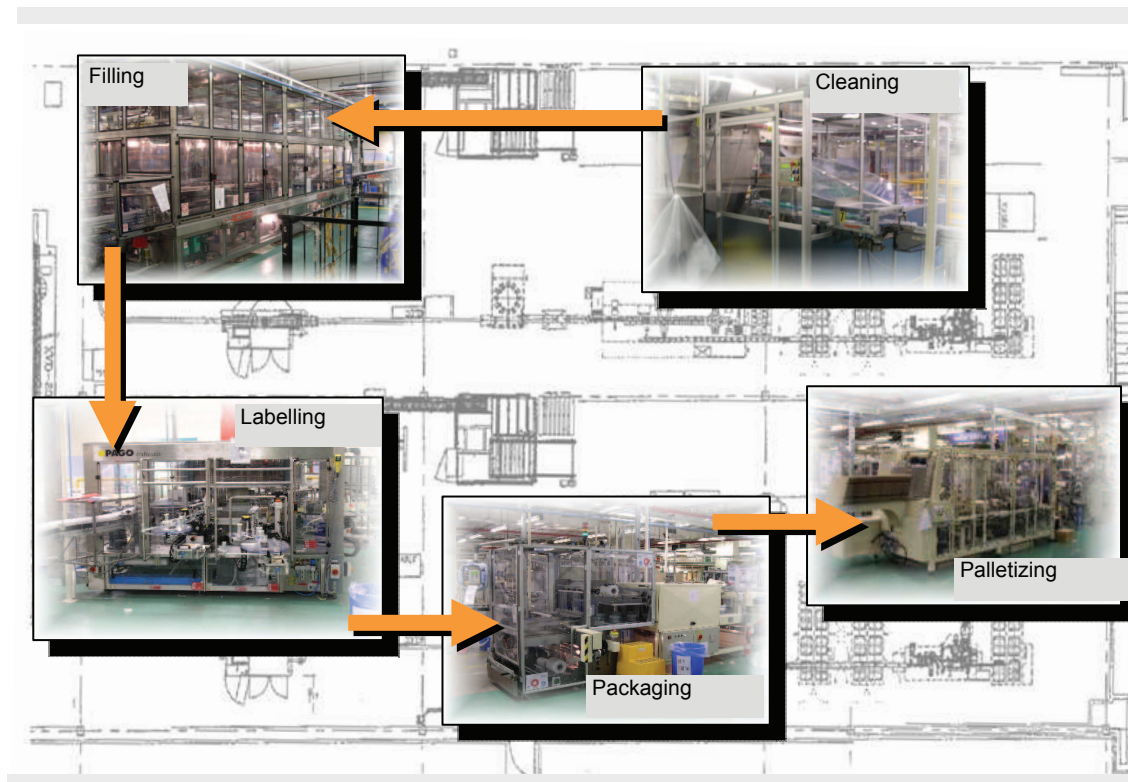


Figure 1 – Automation facility control

IEC 1511/12

Methods and *Programs* model functions typically having different scopes, behaviours, life-times, and complexities in *OPC Servers* and the underlying systems. These functions are **not** normally characterized by the reading or writing of data which is accomplished with the *OPC UA Attribute* service set.

Methods represent basic functions in the server that can be invoked by a client. *Programs* by contrast, model more complex, stateful functionality in the system. For example, a method call may be used to perform a calculation or reset a counter. A *Program* is used to run and control a batch process, execute a machine tool part program, or manage a domain download. *Methods* and their invocation mechanism are described in IEC 62541-3 and IEC 62541-4.

This specification describes the extensions to, or specific use of the core capabilities defined in IEC 62541-5. Support for the feature described in this specification is described by means of *Profiles* in IEC 62541-7.

4.2 Programs

4.2.1 Overview

Programs are complex functions in a server or underlying system that can be invoked and managed by an *OPC UA Client*. *Programs* can represent any level of functionality within a system or process in which client control or intervention is required and progress monitoring is desired. Figure 2 illustrates the model.

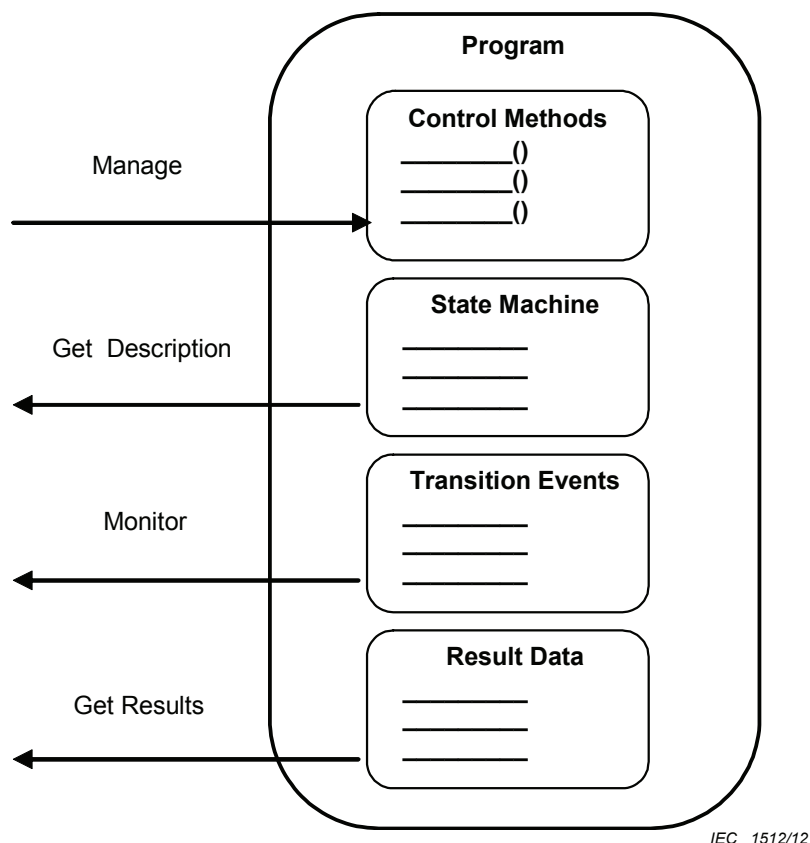


Figure 2 – Program illustration

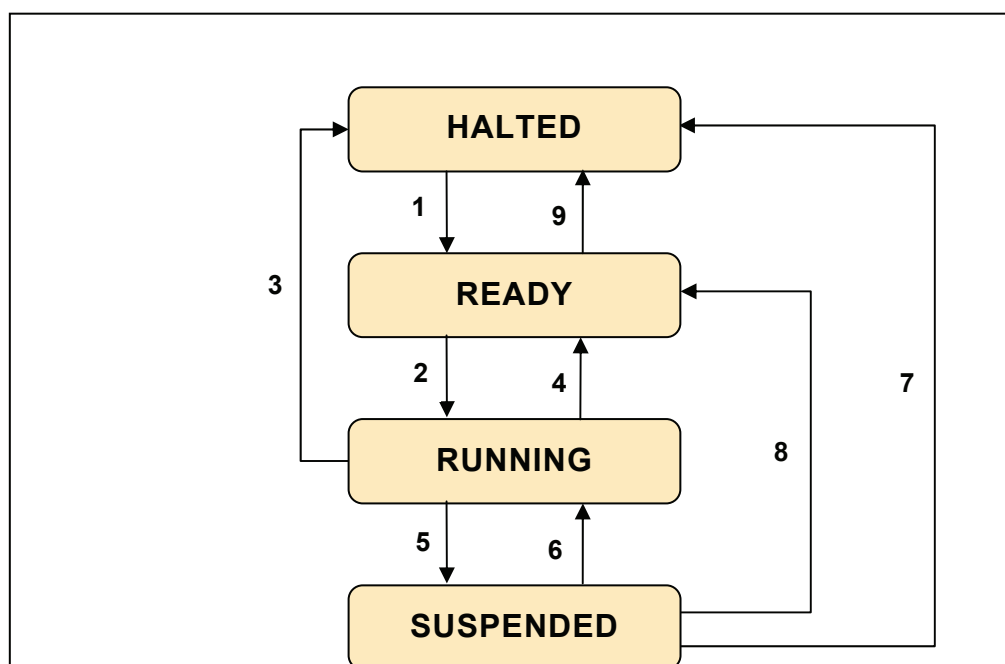
Programs are state full, transitioning through a prescribed sequence of states as they execute. Their behaviour is defined by a *Program Finite State Machine (PFSM)*. The elements of the PFSM describe the phases of a *Program's* execution in terms of valid transitions between a set of states, the stimuli or causes of those transitions, and the resultant effects of the transitions.

4.2.2 Program Finite State Machine

The states, transitions, causes and effects that compose the Program Finite State Machine are listed in Table 1 and illustrated in Figure 3.

Table 1 – Program Finite State Machine

No.	Transition Name	Cause	From State	To State	Effect
1	HaltedToReady	Reset Method	Halted	Ready	Report Transition 1 Event/Result
2	ReadyToRunning	Start Method	Ready	Running	Report Transition 2 Event/Result
3	RunningToHalted	Halt Method or Internal (Error)	Running	Halted	Report Transition 3 Event/Result
4	RunningToReady	Internal	Running	Ready	Report Transition 4 Event/Result
5	RunningToSuspended	Suspend Method	Running	Suspended	Report Transition 5 Event/Result
6	SuspendedToRunning	Resume Method	Suspended	Running	Report Transition 6 Event/Result
7	SuspendedToHalted	Halt Method	Suspended	Halted	Report Transition 7 Event/Result
8	SuspendedToReady	Internal	Suspended	Ready	Report Transition 8 Event/Result
9	ReadyToHalted	Halt Method	Ready	Halted	Report Transition 9 Event/Result



IEC 1513/12

Figure 3 – Program States and Transitions

4.2.3 Program States

A standard set of base states are defined for *Programs* as part of the *Program Finite State Machine*. These states represent the stages in which a *Program* can exist at an instance in time as viewed by a client. This state is the *Program's Current State*. All *Programs* shall support this base set. A *Program* may or may not require a client action to cause the state to change. The states are formally defined in Table 2.

Table 2 – Program States

State	Description
Ready	The <i>Program</i> is properly initialized and may be started.
Running	The <i>Program</i> is executing making progress towards completion.
Suspended	The <i>Program</i> has been stopped prior to reaching a terminal state but may be resumed.
Halted	The <i>Program</i> is in a terminal or failed state, and it cannot be started or resumed without being reset.

The set of states defined to describe a *Program* can be expanded. Program substates can be defined for the base states to provide more resolution to the process and to describe the cause and effects of additional stimuli and transitions. Standards bodies and industry groups may extend the base *Program Finite State Model* to conform to industry models. For example, the Halted state can include the sub states “Aborted” and “Completed” to indicate if the function achieved a successful conclusion prior to the transition to Halted. Transitional states such as “Starting” or “Suspending” might also be extensions of the running state, for example.

4.2.4 State Transitions

A standard set of state transitions is defined for the *Program Finite State Machine*. These transitions define the valid changes to the *Program's* current state in terms of an initial state and a resultant state. The transitions are formally defined in Table 3.

Table 3 – Program State Transitions

Transition No.	Transition Name	Initial State	Resultant State
1	HaltedToReady	Halted	Ready
2	ReadyToRunning	Ready	Running
3	RunningToHalted	Running	Halted
4	RunningToReady	Running	Ready
5	RunningToSuspended	Running	Suspended
6	SuspendedToRunning	Suspended	Running
7	SuspendedToHalted	Suspended	Halted
8	SuspendedToReady	Suspended	Ready
9	ReadyToHalted	Ready	Halted

4.2.5 Program State Transition Stimuli

The stimuli or causes for a *Program's* state transitions can be internal to the *Server* or external. Completion of machining steps, the detection of an alarm condition, or the transmission of a data packet are examples of internal stimuli. *Methods* are an example of external stimuli. Standard *Methods* are defined which act as stimuli for the control of a *Program*.

4.2.6 Program Control Methods

Clients manage a *Program* by calling *Methods*. The *Methods* impact a *Program's* behaviour by causing specified state transitions. The state transitions dictate the action's performed by the *Program*. This specification defines a set of standard *Program Control Methods*. These *Methods* provide sufficient means for a client to run a *Program*.

Table 4 lists the set of defined *Program Control Methods*. Each *Method* causes transitions from specified states and shall be called when the *Program* is in one of those states.

Individual *Programs* can optionally support any subset of the *Program Control Methods*. For example, some *Programs* may not be permitted to suspend and so would not provide the *Suspend* and *Resume Methods*.

Programs can support additional user defined *Methods*. User defined *Methods* shall not change the behaviour of the base *Program Finite State Machine*.

Table 4 – Program Control Methods

Method Name	Description
Start	Causes the Program to transition from the Ready state to the Running state.
Suspend	Causes the Program to transition from the Running state to the Suspended state.
Resume	Causes the Program to transition from the Suspended state to the Running state.
Halt	Causes the Program to transition from the Ready, Running or Suspended state to the Halted state.
Reset	Causes the Program to transition from the Halted state to the Ready state.

Program Control Methods can include arguments that are used by the *Program*. For example, a *Start* method may include an options argument that specifies dynamic options used to determine some program behaviour. The arguments can differ on each *ProgramType*. The *Method Call* service specified in IEC 62541-4:2011, 5.11 defines a return status. This return status indicates the success of the *Program Control Method* or a reason for its failure.

4.2.7 Program State Transition Effects

A *Program*'s state transition generally has a cause and also yields an effect. The effect is a by product of a *Program* state transition that can be used by a *Client* to monitor the progress of the *Program*. Effects can be internal or external. An external effect of a state transition is the generation of an event notification. Each *Program* state transition is associated with a unique event. These events reflect the progression and trajectory of the *Program* through its set of defined states. The internal effects of a state transition can be the performance of some programmatic action such as the generation of data.

4.2.8 Program Result Data

4.2.8.1 Overview

Result data is generated by a running *Program*. The result data can be intermediate or final. Result data may be associated with specific *Program* state transitions.

4.2.8.2 Intermediate Result Data

Intermediate result data is transient and is generated by the *Program* in conjunction with non terminal state transitions. The data items that compose the intermediate results are defined in association with specific *Program* state transitions. Their values are relevant only at the transition.

Each *Program* state transition can be associated with different result data items. Alternately, a set of transitions can share a result data item. Percentage complete is an example of intermediate result data. The value of percentage complete is produced when the state transition occurs and is available to the client.

Clients acquire intermediate result data by subscribing to *Program* state transition events. The events specify the data items for each transition. When the transition occurs, the generated event conveys the result data values captured to the subscribed clients. If no *Client* is monitoring the *Program*, intermediate result data may be discarded.

4.2.8.3 Terminal Result Data

Terminal result data is the final data generated by the *Program* as it ceases execution. Total execution time, number of widgets produced, and fault condition encountered are examples of terminal result data. When the *Program* enters the terminal state, this result data can be conveyed to the client by the transition event. Terminal result data is also available within the *Program* to be read by a client after the program stops. This data persists until the program instance is rerun or deleted.

4.2.8.4 Monitoring Programs

Clients can monitor the activities associated with a *Program*'s execution. These activities include the invocation of the management methods, the generation of result data, and the progression of the *Program* through its states. Audit Events are provided for *Method Calls* and state transitions. These events allow a record to be maintained of the clients that interacted with any *Program* and the *Program* state transitions that resulted from that interaction.

4.2.9 Program Lifetime

4.2.9.1 Overview

Programs can have different lifetimes. Some programs may always be present on a *Server* while others are created and removed. Creation and removal can be controlled by a *Client* or may be restricted to local means.

A *Program* can be *Client* creatable. If a *Program* is client creatable, then the *Client* can add the *Program* to the server. The *Object Create Method* defined in IEC 62541-13:2010, 5.5.4 is