

DIN EN 62541-10

ICS 35.200; 35.240.50

Ersatz für
DIN EN 62541-10:2013-06
Siehe Anwendungsbeginn

**OPC Unified Architecture –
Teil 10: Programme
(IEC 62541-10:2015);
Englische Fassung EN 62541-10:2015**

OPC unified architecture –
Part 10: Programs
(IEC 62541-10:2015);
English version EN 62541-10:2015

Architecture unifiée OPC –
Partie 10: Programmes
(IEC 62541-10:2015);
Version anglaise EN 62541-10:2015

Gesamtumfang 51 Seiten

DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in DIN und VDE

Anwendungsbeginn

Anwendungsbeginn für die von CENELEC am 2015-04-14 angenommene Europäische Norm als DIN-Norm ist 2015-11-01.

Für DIN EN 62541-10:2013-06 besteht eine Übergangsfrist bis 2018-04-14.

Nationales Vorwort

Vorausgegangener Norm-Entwurf: E DIN EN 62541-10:2014-11.

Für dieses Dokument ist das nationale Arbeitsgremium K 931 „Systemaspekte“ der DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik in 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 ein Dokument ohne Angabe des Ausgabedatums und ohne Hinweis auf eine Abschnittsnummer, eine Tabelle, ein Bild usw.) bezieht sich die Verweisung auf die jeweils aktuellste Ausgabe des in Bezug genommenen Dokuments.

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

Der Zusammenhang der zitierten Dokumente mit den entsprechenden Deutschen Dokumenten 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.

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 1, der den Anwendungsbereich festlegt, und Abschnitt 3, der die Begriffe festlegt, wurde eine Übersetzung angefertigt und als 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.

Das Original-Dokument enthält Bilder in Farbe, die in der Papierversion in einer Graustufen-Darstellung wiedergegeben werden. Elektronische Versionen dieses Dokuments enthalten die Bilder in der originalen Farbdarstellung.

Änderungen

Gegenüber DIN EN 62541-10:2013-06 wurden folgende Änderungen vorgenommen:

- a) Ergänzung von Sicherheitsbetrachtungen als 4.2.2 auf der Grundlage der NIST-Überarbeitung;
- b) Festlegung der Definition des ProgrammDiagnostikTyps (en: ProgramDiagnsoticsType) als ein Datentyp (5.2.8) und Ergänzung fehlender Datentypen für die ProgramDiagnostikVaribale (en: ProgramDiagnosticVariable) im ProgrammTyp (en: ProgramType) in Tabelle 5;
- c) Korrektur des BrowserNamens (en: BrowserName) des Auditereignisses für Programmübergänge in Tabelle 7.

Frühere Ausgaben

DIN EN 62541-10: 2013-06

Nationaler Anhang NA (informativ)

1 Anwendungsbereich

Dieser Teil der Normenreihe IEC 62541 ist Teil der gesamten Normenreihe zur OPC Unified Architecture (vereinheitlichte Architektur der plattformunabhängigen Kommunikation) (OPC-UA)) und legt das Informationsmodell fest, welches im Zusammenhang mit *Programmen* steht. Dies schließt die Festlegung der *KnotenKlassen* (en: *NodeClasses*), genormter *Eigenschaften* (en: *Properties*), *Methoden* (en: *Methods*) und *Ereignisse* (en: *Events*) und zugehöriges Verhalten und Informationen für *Programme* ein.

Das vollständige Adressraummodell, einschließlich aller *KnotenKlassen* und *Attribute* ist in IEC 62541-3 festgelegt. Die Dienste, wie beispielsweise jene zum Aufrufen *der Methoden*, welche zur Verwaltung von *Programmen* verwendet werden, sind in IEC 62541-4 festgelegt.

3 Begriffe und Konventionen

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 durch einen Server oder ein Gerät, üblicherweise durch Ausführung eines Maschinencodes

3.1.2

Endlicher Zustands Automat

(en: Finite State Machine)

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

3.1.3

ProgrammTyp

(en: ProgramType)

Typdefinition eines *Programms* und ist einen Subtyp des *EndlicherZustandsAutomatTyp*

3.1.4

Programmsteuerungsmethode

(en: Program Control Method)

Methode mit spezifischer Semantik und angelegt zur Steuerung eines *Programms* durch Auslösen eines Zustandsübergangs

3.1.5

Programmaufruf

(en: Program Invocation)

eindeutige *Objekt*-Instanz eines *Programms*, die auf einem *Server* existiert

ANMERKUNG 1 zum Begriff: Ein *Programmaufruf* unterscheidet sich von anderen *Objekt*-Instanzen desselben *ProgramType* durch den eindeutigen Suchpfad des *ObjektKnotens*.

English Version

OPC unified architecture - Part 10: Programs
(IEC 62541-10:2015)

Architecture unifiée OPC - Partie 10: Programmes
(IEC 62541-10:2015)

OPC Unified Architecture - Teil 10: Programme
(IEC 62541-10:2015)

This European Standard was approved by CENELEC on 2015-04-14. 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.



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

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Foreword

The text of document 65E/383/FDIS, future edition 2 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 62541-10:2015.

The following dates are fixed:

- latest date by which the document has to be (dop) 2016-01-14
implemented at national level by
publication of an identical national
standard or by endorsement
- latest date by which the national (dow) 2018-04-14
standards conflicting with the
document have to be withdrawn

This document supersedes EN 62541-10:2012.

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.

This document has been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association.

Endorsement notice

The text of the International Standard IEC 62541-10:2015 was approved by CENELEC as a European Standard without any modification.

CONTENTS

1	Scope.....	6
2	Normative references	6
3	Terms, definitions and conventions	6
3.1	Terms and definitions	6
3.2	Abbreviations	7
4	Concepts.....	7
4.1	General.....	7
4.2	Programs	8
4.2.1	Overview.....	8
4.2.2	Security considerations	9
4.2.3	Program Finite State Machine.....	9
4.2.4	Program states.....	10
4.2.5	State transitions	11
4.2.6	Program state transition stimuli	11
4.2.7	Program Control Methods.....	11
4.2.8	Program state transition effects	12
4.2.9	Program result data.....	12
4.2.10	Program lifetime.....	13
5	Model.....	14
5.1	General.....	14
5.2	ProgramType	14
5.2.1	Overview.....	14
5.2.2	ProgramType Properties.....	16
5.2.3	ProgramType components	16
5.2.4	ProgramType causes (Methods)	21
5.2.5	ProgramType effects (Events)	23
5.2.6	AuditProgramTransitionEventType	25
5.2.7	FinalResultData	26
5.2.8	ProgramDiagnostic DataType	26
5.2.9	ProgramDiagnosticType VariableType	27
Annex A (informative)	Program example.....	28
A.1	Overview	28
A.2	DomainDownload Program	28
A.2.1	General.....	28
A.2.2	DomainDownload states	29
A.2.3	DomainDownload transitions	30
A.2.4	DomainDownload Methods	30
A.2.5	DomainDownload Events.....	31
A.2.6	DomainDownload model.....	31
Annex ZA (normative)	Normative references to international publications with their corresponding European publications.....	47
Figure 1 – Automation facility control.....		8

Figure 2 – Program illustration	9
Figure 3 – Program states and transitions	10
Figure 4 – Program Type	14
Figure 5 – Program FSM References	17
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.....	35
Figure A.4 – Ready To Running model.....	38
Figure A.5 – Opening To Sending To Closing model.....	40
Figure A.6 – Running To Suspended model	41
Figure A.7 – Suspended To Running model	42
Figure A.8 – Running To Halted – Aborted model.....	43
Figure A.9 – Suspended To Aborted model	44
Figure A.10 – Running To Completed model	45
Figure A.11 – Sequence of operations	46
Table 1 – Program Finite State Machine	9
Table 2 – Program states.....	10
Table 3 – Program state transitions.....	11
Table 4 – Program Control Methods	12
Table 5 – ProgramType	15
Table 6 – Program states.....	17
Table 7 – Program transitions	19
Table 8 – ProgramType causes.....	22
Table 9 – ProgramTransitionEventType.....	23
Table 10 – ProgramTransitionEvents.....	24
Table 11 – AuditProgramTransitionEventType	25
Table 12 – ProgramDiagnosticDataType structure	26
Table 13 – ProgramDiagnosticDataType definition.....	26
Table 14 – ProgramDiagnosticType VariableType	27
Table A.1 – DomainDownload states	30
Table A.2 – DomainDownload Type	32
Table A.3 – Transfer State Machine Type.....	32
Table A.4 – Transfer State Machine – states	33
Table A.5 – Finish State Machine Type	33
Table A.6 – Finish State Machine – states.....	34
Table A.7 – DomainDownload Type Property Attributes variable values	34
Table A.8 – Additional DomainDownload transition types.....	36
Table A.9 – Start Method additions	38
Table A.10 – StartArguments	39
Table A.11 – IntermediateResults Object	40

Table A.12 – Intermediate result data Variables.....41

Table A.13 – FinalResultData.....44

Table A.14 – Final result Variables 45

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

1 Scope

This part of IEC 62541 is part of the overall OPC Unified Architecture (OPC UA) standard series and defines the information model associated with *Programs*. 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 62541-3:2015, *OPC Unified Architecture – Part 3: Address Space Model*

IEC 62541-4:2015, *OPC Unified Architecture – Part 4: Services*

IEC 62541-5:2015, *OPC Unified Architecture – Part 5: Information Model*

IEC 62541-7, *OPC Unified Architecture – Part 7: Profiles*

3 Terms, definitions and conventions

3.1 Terms and definitions

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

3.1.1

function

programmatic task performed by 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

type definition of a *Program* and is a subtype of the *FiniteStateMachineType*

3.1.4

Program Control Method

Method 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: A *Program Invocation* is distinguished from other *Object* instances of the same *ProgramType* by the object node's unique browse path.

3.2 Abbreviations

DA	Data Access
FSM	Finite State Machine
HMI	Human Machine Interfaces
PCM	Program Control Method
PGM	Program
PI	Program Invocation
UA	Unified Architecture

4 Concepts

4.1 General

Integrated automation facilities manage their operations through the exchange of data and the coordinated invocation of system functions as 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, 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.

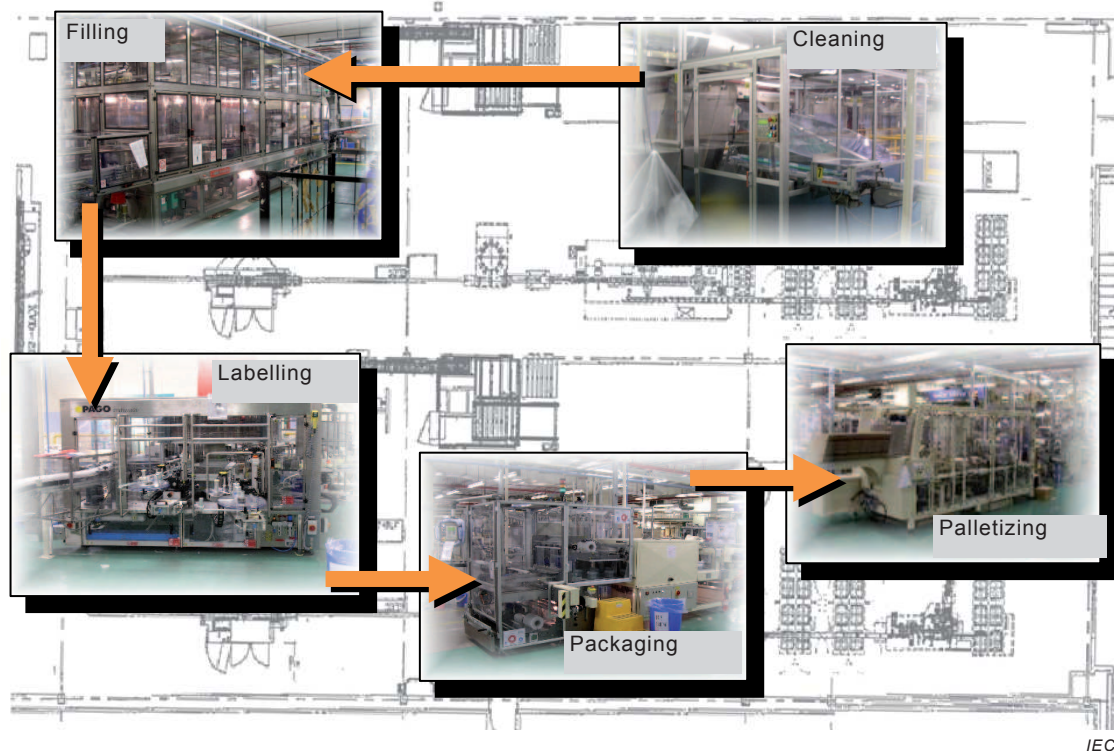


Figure 1 – Automation facility control

Methods and *Programs* model functions typically have different scopes, behaviours, lifetimes, 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 and 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 standard describes the extensions to, or specific use of, the core capabilities defined in IEC 62541-5 as required for *Programs*.

4.2 Programs

4.2.1 Overview

Programs are complex functions in a server or underlying system that can be invoked and managed by a *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.

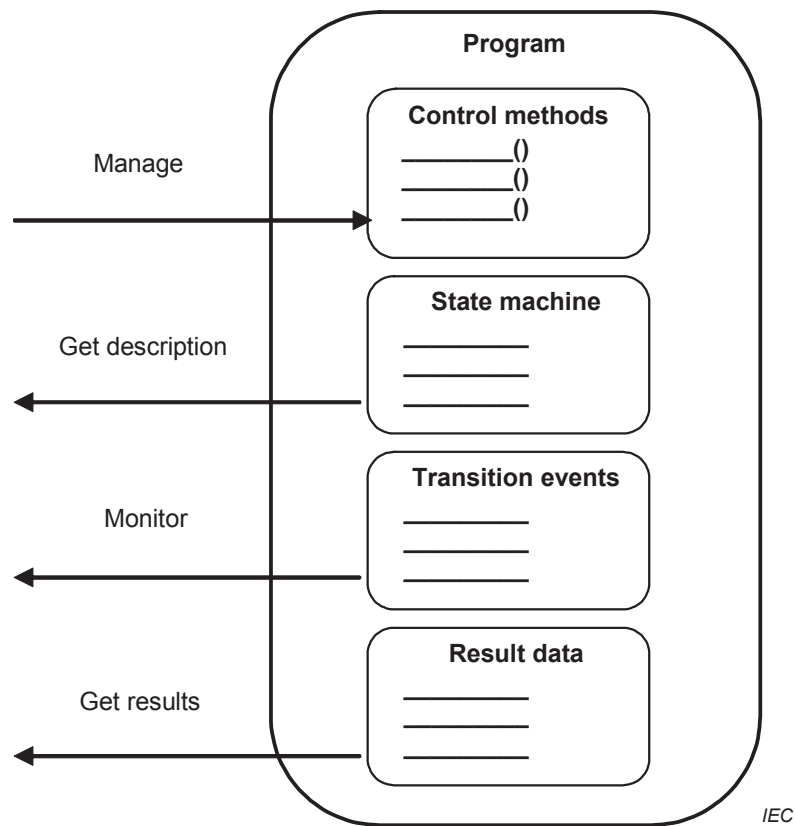


Figure 2 – Program illustration

Programs are state full and transition 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 Security considerations

Since *Programs* can be used to perform advanced control algorithms or other actions, their use should be restricted to personnel with appropriate access rights. It is recommended that *AuditUpdateMethodEvents* are generated to allow monitoring the number of running *Programs* in addition to their execution frequency.

4.2.3 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

No.	Transition name	Cause	From state	To state	Effect
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

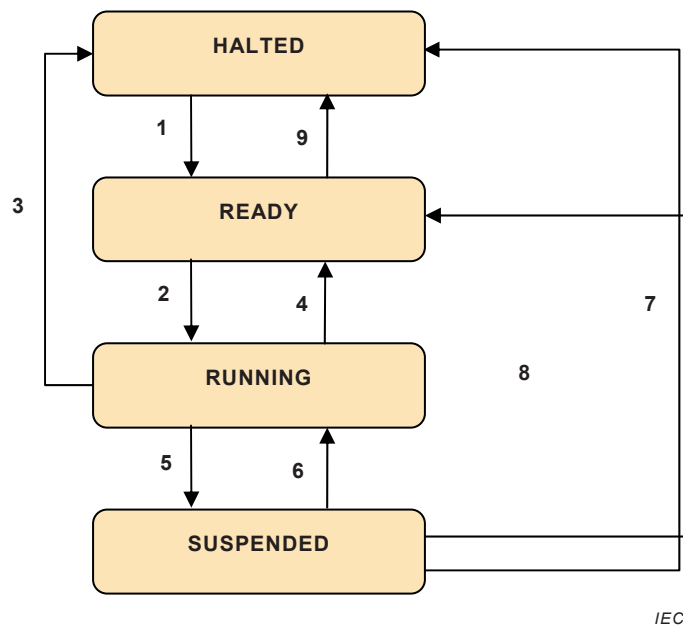


Figure 3 – Program states and transitions

4.2.4 Program states

A standard set of base states is 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* sub states can be defined for the base states to provide more resolution of a process and to describe the cause and effect(s) of additional stimuli and transitions. Standards bodies and industry groups may extend the

base *Program Finite State Model* to conform to various 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.

4.2.5 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

4.2.6 Program state transition stimuli

The stimuli or causes for a *Program*’s state transitions can be internal to the *Server* or external. The 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*.

4.2.7 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 actions performed by the *Program*. This standard 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 <i>Program</i> to transition from the Ready state to the Running state.
Suspend	Causes the <i>Program</i> to transition from the Running state to the Suspended state.
Resume	Causes the <i>Program</i> to transition from the Suspended state to the Running state.
Halt	Causes the <i>Program</i> to transition from the Ready, Running or Suspended state to the Halted state.
Reset	Causes the <i>Program</i> 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 62541-4:2015, 5.11 defines a return status. This return status indicates the success of the *Program Control Method* or a reason for its failure.

4.2.8 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.

4.2.9 Program result data

4.2.9.1 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.

4.2.9.2 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 level.

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.

4.2.9.3 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