

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**OPC unified architecture –
Part 10: Programs**

**Architecture unifiée OPC –
Partie 10: Programmes**

Withdrawn



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2012 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de la CEI ou du Comité national de la CEI du pays du demandeur.

Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,...).

It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente. un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,...).

Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**OPC unified architecture –
Part 10: Programs**

**Architecture unifiée OPC –
Partie 10: Programmes**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX



ICS 25.040.40; 25.100.01

ISBN 978-2-83220-284-5

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

CONTENTS

FOREWORD.....	4
INTRODUCTION.....	6
1 Scope.....	7
2 Normative references	7
3 Terms, definitions and abbreviations	7
3.1 Terms and definitions	7
3.2 Abbreviations	8
4 Concepts.....	8
4.1 General.....	8
4.2 Programs	9
4.2.1 Overview	9
4.2.2 Program Finite State Machine.....	10
4.2.3 Program States	11
4.2.4 State Transitions	11
4.2.5 Program State Transition Stimuli	12
4.2.6 Program Control Methods.....	12
4.2.7 Program State Transition Effects	13
4.2.8 Program Result Data	13
4.2.9 Program Lifetime	13
5 Model.....	14
5.1 General.....	14
5.2 ProgramType.....	15
5.2.1 Overview.....	15
5.2.2 ProgramType Properties.....	17
5.2.3 ProgramType Components	17
5.2.4 ProgramType Causes (Methods)	23
5.2.5 ProgramType Effects (Events).....	24
5.2.6 AuditProgramTransitionEventType.....	27
5.2.7 FinalResultData.....	28
5.2.8 ProgramDiagnosticType	28
Annex A (informative) Program Example.....	30
Figure 1 – Automation facility control	9
Figure 2 – Program illustration	10
Figure 3 – Program States and Transitions	11
Figure 4 – Program Type	15
Figure 5 – Program FSM References	18
Figure 6 – ProgramType Causes and Effects	23
Figure A.1 – Program Example	30
Figure A.2 – DomainDownload State Diagram.....	31
Figure A.3 – DomainDownloadType Partial State Model.....	38
Figure A.4 – ReadyToRunning Model.....	41
Figure A.5 – OpeningToSending ToClosing Model	43
Figure A.6 – RunningToSuspended Model	44

Figure A.7 – SuspendedToRunning Model	45
Figure A.8 – RunningToHalted – Aborted Model	46
Figure A.9 – SuspendedToAborted Model	47
Figure A.10 – RunningToCompleted Model	48
Figure A.11 – Sequence of Operations	49
Table 1 – Program Finite State Machine	10
Table 2 – Program States	11
Table 3 – Program State Transitions	12
Table 4 – Program Control Methods	12
Table 5 – ProgramType	16
Table 6 – Program States	19
Table 7 – Program Transitions	20
Table 8 – ProgramType Causes	23
Table 9 – ProgramTransitionEventType	25
Table 10 – ProgramTransitionEvents	26
Table 11 – AuditProgramTransitionEventType	27
Table 12 – ProgramDiagnosticType	28
Table A.1 – DomainDownload States	32
Table A.2 – DomainDownload Type	34
Table A.3 – Transfer State Machine Type	35
Table A.4 – Transfer State Machine – States	36
Table A.5 – Finish State Machine Type	36
Table A.6 – Finish State Machine – States	37
Table A.7 – DomainDownload Type Property Attributes Variable Values	37
Table A.8 – Additional DomainDownload Transition Types	39
Table A.9 – Start Method Additions	41
Table A.10 – StartArguments	42
Table A.11 – Intermediate Results Object	43
Table A.12 – Intermediate Result Data Variables	44
Table A.13 – Final Result Data	47
Table A.14 – Final Result Variables	48

INTERNATIONAL ELECTROTECHNICAL COMMISSION

OPC UNIFIED ARCHITECTURE –

Part 10: Programs

FOREWORD

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as “IEC Publication(s)”). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62541-10 has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation.

The text of this standard is based on the following documents:

FDIS	Report on voting
65E/244/FDIS	65E/269/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 62541 series, published under the general title *OPC unified architecture*, can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "<http://webstore.iec.ch>" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

Withdrawn

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.

Withdrawn

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.

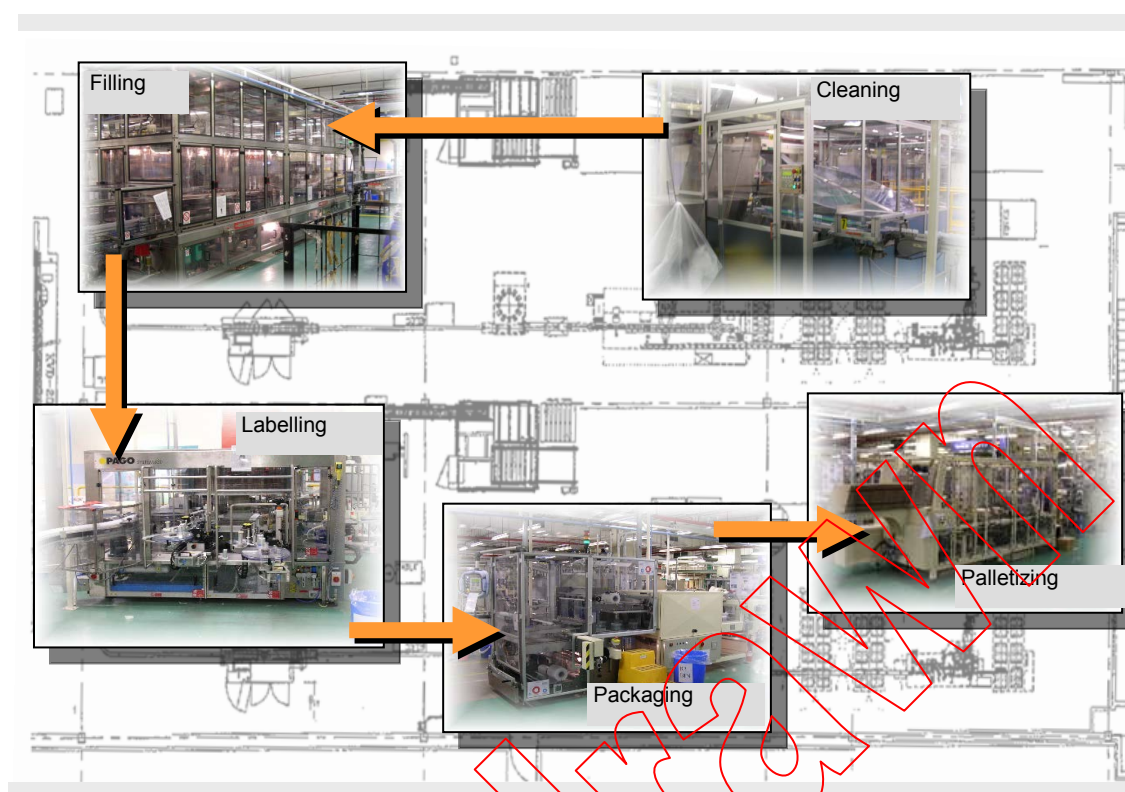


Figure 1 – Automation facility control

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.

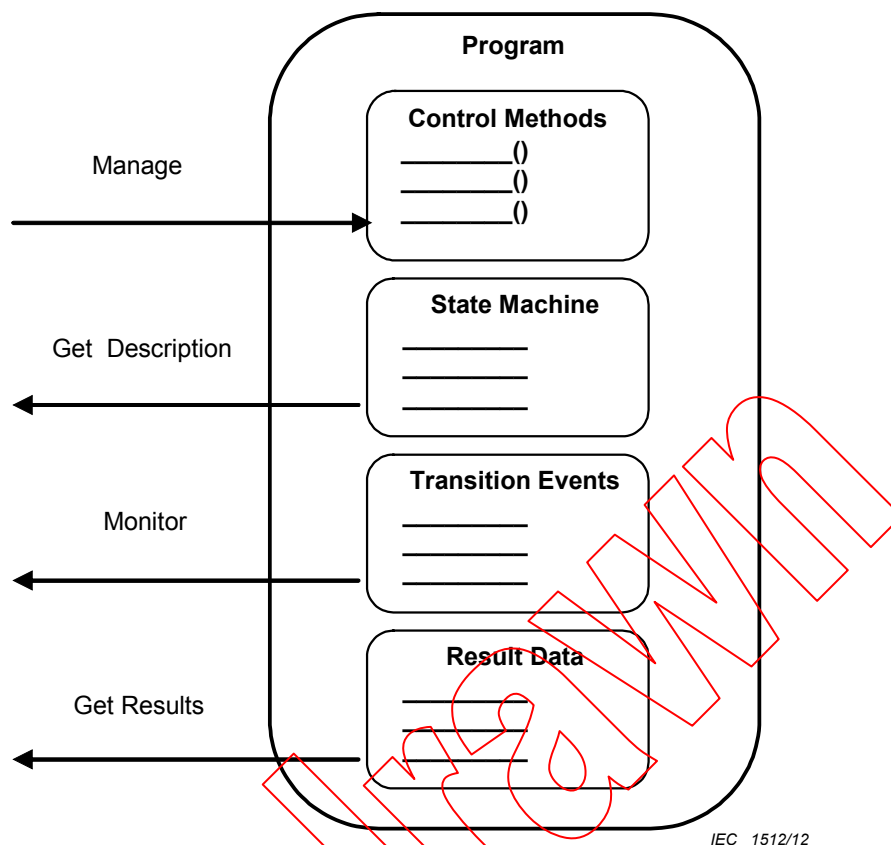


Figure 2 – Program illustration

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

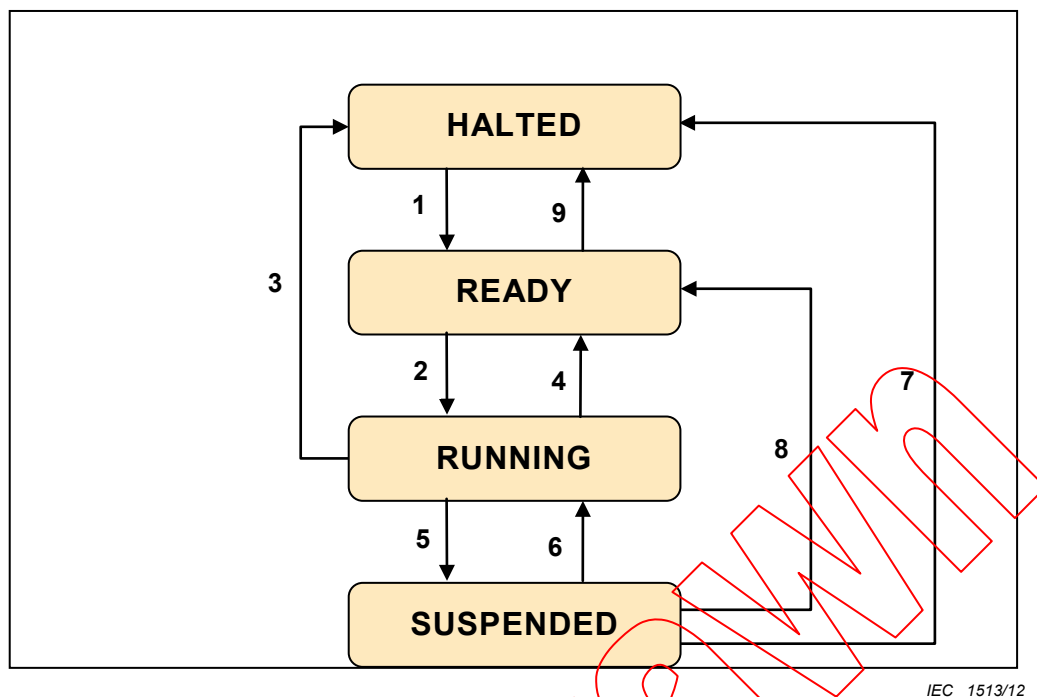


Figure 3 – Program States and Transitions

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 actions 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

used to create the *Program* Instance. The initial state of the *Program* can be *Halted* or *Ready*. Some Programs, for example, may require that a resource becomes available after its creation, before it is ready to run. In this case, it would be initialized in the *Halted* state and transition to *Ready* when the resource is delivered.

A Program can be Client removable. If the *Program* is client removable, then the *Client* can delete the *Program* Instance from the *Server*. The *DeleteNodes Service* defined in IEC 62541-4:2011, 5.7 is used to remove the *Program* Instance. The *Program* shall be in a *Halted* state to be removed. A *Program* may also be auto removable. An auto removable *Program* deletes itself when execution has terminated.

4.2.9.2 Program Instances

Programs can be multiple instanced or single instanced. A *Server* can support multiple instances of a *Program* if these *Program* instances can be run in parallel. For example, the *Program* may define a *Start Method* that has an input argument to specify which resource is acted upon by its functions. Each instance of the *Program* is then started designating use of different resources. The *Client* can discover all instances of a *Program* that are running on a *Server*. Each instance of a *Program* is uniquely identified on the *Server* and is managed independently by the *Client*.

4.2.9.3 Program Recycling

Programs can be run once or run multiple times (recycled). A program that is run once will remain in the *Halted state* indefinitely once it has run. The normal course of action would be to delete it following the inspection of its terminal results.

Recyclable *Programs* may have a limited or unlimited cycle count. These *Programs* may require a reset step to transition from the *Halted state* to the *Ready state*. This allows for replenishing resources or reinitializing parameters prior to restarting the *Program*. The *Program Control Method* “Reset” triggers this state transition and any associated actions or effects.

5 Model

5.1 General

The *Program* Model extends the *FiniteStateMachineType* and basic *ObjectType* Models presented in IEC 62541-5. Each *Program* has a type definition that is the subtype of the *FiniteStateMachineType*. The *ProgramType* describes the *Finite State Machine* model supported by any *Program Invocation* of that type. The *ProgramType* also defines the property set that characterize specific aspects of that *Program*’s behaviour such as lifetime and recycling, as well as specifying the result data that is produced by the *Program*.

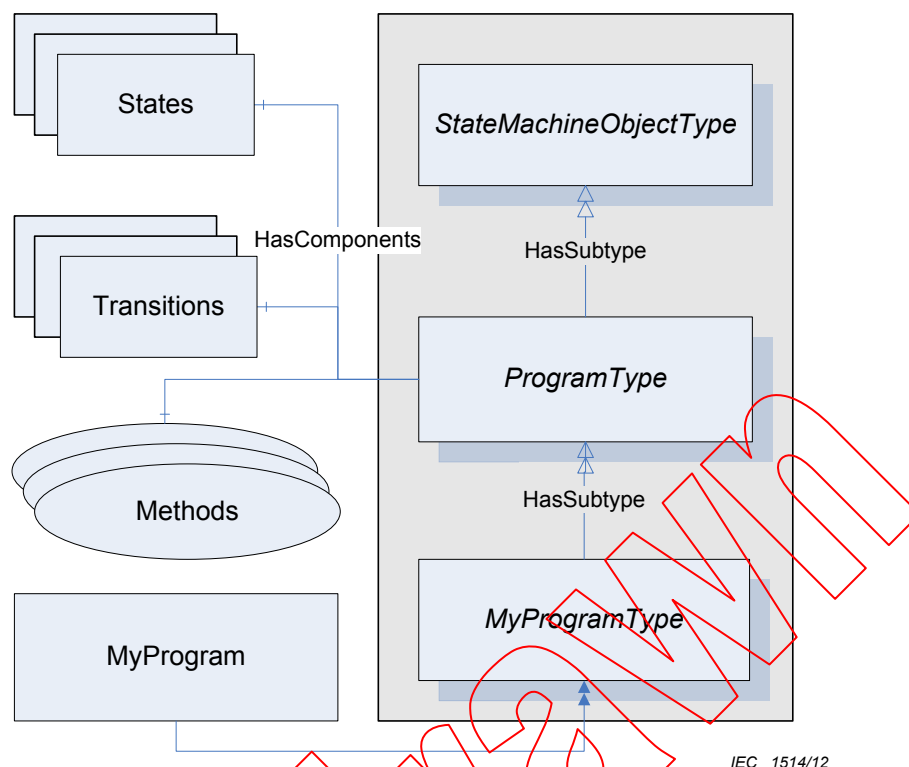


Figure 4 – Program Type

The base *ProgramType* defines the standard *Finite State Machine* specified for all *Programs*. This includes the states, transitions, transition causes (*Methods*) and effects (*Events*). Subtypes of the base *ProgramType* can be defined to extend or more specifically characterize the behaviour of an individual *Program* as illustrated with “*MyProgramType*” in Figure 4.

5.2 ProgramType

5.2.1 Overview

The additional properties and components that compose the *ProgramType* are listed in Table 5. No *ProgramType* specific semantics are assigned to the other base *ObjectType* or *FiniteStateMachineType* *Attributes* or *Properties*.

Table 5 – ProgramType

Attribute	Value				
	Includes all attributes specified for the FiniteStateMachineType				
BrowseName	ProgramType				
IsAbstract	False				
References	NodeClass	BrowseName	Data Type	TypeDefinition	Modelling Rule
HasProperty	Variable	Creatable	Boolean	PropertyType	--
HasProperty	Variable	Deletable	Boolean	PropertyType	Mandatory
HasProperty	Variable	AutoDelete	Boolean	PropertyType	Mandatory
HasProperty	Variable	RecycleCount	Int32	PropertyType	Mandatory
HasProperty	Variable	InstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxInstanceCount	UInt32	PropertyType	--
HasProperty	Variable	MaxRecycleCount	UInt32	PropertyType	--
HasComponent	Variable	ProgramDiagnostic		ProgramDiagnosticType	Optional
HasComponent	Object	Halted		StateType	Mandatory
HasComponent	Object	Ready		StateType	Mandatory
HasComponent	Object	Running		StateType	Mandatory
HasComponent	Object	Suspended		StateType	Mandatory
HasComponent	Object	HaltedToReady		TransitionType	Mandatory
HasComponent	Object	ReadyToRunning		TransitionType	Mandatory
HasComponent	Object	RunningToHalted		TransitionType	Mandatory
HasComponent	Object	RunningToReady		TransitionType	Mandatory
HasComponent	Object	RunningToSuspended		TransitionType	Mandatory
HasComponent	Object	SuspendedToRunning		TransitionType	Mandatory
HasComponent	Object	SuspendedToHalted		TransitionType	Mandatory
HasComponent	Object	SuspendedToReady		TransitionType	Mandatory
HasComponent	Object	ReadyToHalted		TransitionType	Mandatory
HasComponent	Method	Start			Optional
HasComponent	Method	Suspend			Optional
HasComponent	Method	Reset			Optional
HasComponent	Method	Halt			Optional
HasComponent	Method	Resume			Optional
HasComponent	Object	FinalResultData		BaseObjectType	Optional

5.2.2 ProgramType Properties

The *Creatable Property* is a Boolean that specifies if *Program Invocations* of this *ProgramType* can be created by a *Client*. If False, these *Program Invocations* are persistent or may only be created by the server.

The *Deletable Property* is a Boolean that specifies if a *Program Invocation* of this *ProgramType* can be deleted by a *Client*. If False, these *Program Invocations* can only be deleted by the server.

The *AutoDelete Property* is a Boolean that specifies if *Program Invocations* of this *ProgramType* are removed by the *Server* when execution terminates. If False, these *Program Invocations* persist on the server until they are deleted by the *Client*. When the *Program Invocation* is deleted, any result data associated with the instance is also removed.

The *RecycleCount Property* is an unsigned integer that specifies the number of times a *Program Invocation* of this type has been recycled or restarted from its starting point (not resumed). Note: The *Reset Method* may be required to prepare a *Program* to be restarted.

The *MaxRecycleCount Property* is an integer that specifies the maximum number of times a *Program Invocation* of this type can be recycled or restarted from its starting point (not resumed). If the value is less than 0, there is no limit to the number of restarts. If the value is zero, the *Program* may not be recycled or restarted.

The *InstanceCount Property* is an unsigned integer that specifies the number of *Program Invocations* of this type that currently exist.

The *MaxInstanceCount Property* is an integer that specifies the maximum number of *Program Invocations* of this type that can exist simultaneously on this *Server*. If the value is less than 0, there is no limit.

5.2.3 ProgramType Components

5.2.3.1 Overview

The *ProgramType Components* consists of a set of references to the object instances of *StateTypes*, *TransitionTypes*, *EventTypes* and the *Methods* that collectively define the *Program FiniteStateMachine*.

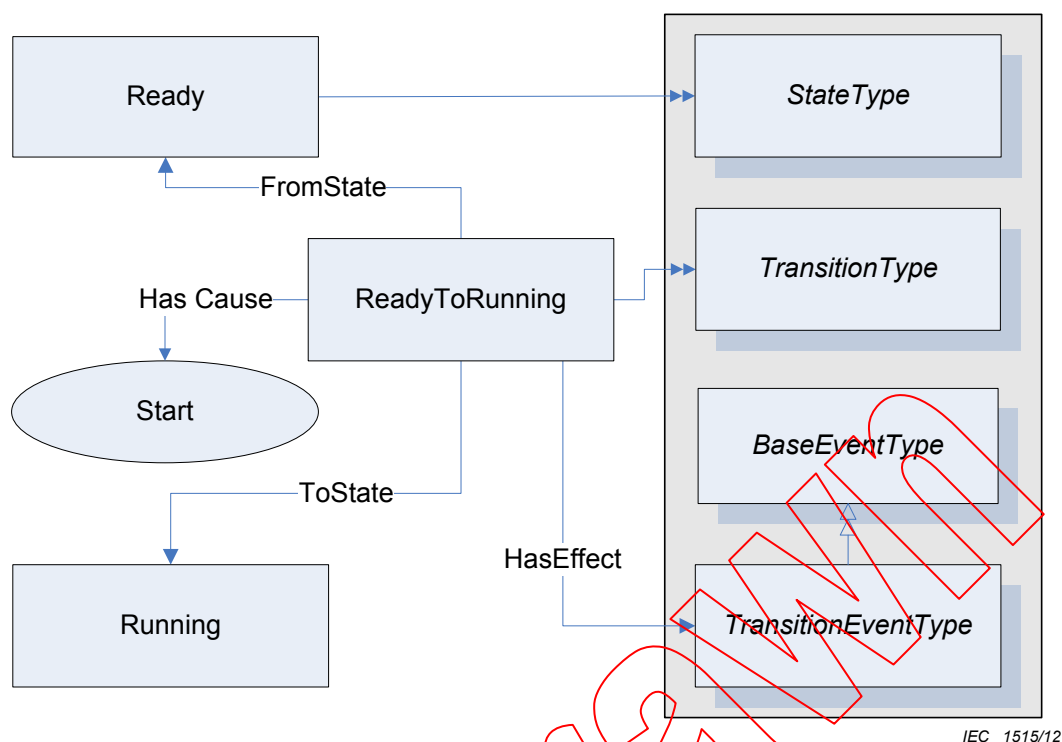


Figure 5 – Program FSM References

Figure 5 illustrates the *Component References* that define the associations between two of the *ProgramType*'s states, *Ready* and *Running*. The complementary *ReferenceTypes* have been omitted to simplify the illustration.

5.2.3.2 ProgramType States

Table 6 specifies the *ProgramType*'s *State Objects*. These *State Objects* are instances of the *StateType* defined in IEC 62541-5:2011, Annex B. Each *State* is assigned a unique *StateNumber* value. Subtypes of the *ProgramType* can add references from any state to a subordinate or nested *StateMachine Object* to extend the *FinitStateMachine*.

Table 6 – Program States

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
States					
Halted	HasProperty	StateNumber	1	PropertyType	
	ToTransition	HaltedToReady		TransitionType	
	FromTransition	RunningToHalted		TransitionType	
	FromTransition	SuspendedToHalted		TransitionType	
	FromTransition	ReadyToHalted		TransitionType	
Ready	HasProperty	StateNumber	2	PropertyType	
	FromTransition	HaltedToReady		TransitionType	
	ToTransition	ReadyToRunning		TransitionType	
	FromTransition	RunningToReady		TransitionType	
	ToTransition	ReadyToHalted		TransitionType	
Running	HasProperty	StateNumber	3	PropertyType	
	ToTransition	RunningToHalted		TransitionType	
	ToTransition	RunningToReady		TransitionType	
	ToTransition	RunningToSuspended		TransitionType	
	FromTransition	ReadyToRunning		TransitionType	
	FromTransition	SuspendedToRunning		TransitionType	
Suspended	HasProperty	StateNumber	4	PropertyType	
	ToTransition	SuspendedToRunning		TransitionType	
	ToTransition	SuspendedToHalted		TransitionType	
	ToTransition	SuspendedToReady		TransitionType	
	FromTransition	RunningToSuspended		TransitionType	

The *Halted* state is the idle state for a *Program*. It can be an initial state or a terminal state. As an initial state, the Program Invocation can not yet begin execution due to conditions at the server. As a terminal state, *Halted* can indicate either a failed or completed *Program*. A subordinate state or result can be used to distinguish the nature of the termination. The *Halted* state references four *Transition Objects*, which identify the allowed state transitions to the *Ready* state and from the *Ready*, *Running*, and *Suspended* States.

The *Ready* state indicates that the *Program* is prepared begin execution. *Programs* that are ready to begin upon their creation may transition immediately to the *Ready* State. The *Ready* state references four *Transition Objects*, which identify the allowed state transitions to the *Running* and *Halted* states and from the *Halted* and *Ready* states.

The *Running* state indicates that the *Program* is actively performing its function. The *Running* state references five *Transition Objects*, which identify the allowed state transitions to the *Halted*, *Ready*, and *Suspended* states and from the *Ready* and *Suspended* States.

The *Suspended* state indicates that the *Program* has stopped performing its function, but retains the ability to resume the function at the point at which it was executing when suspended.

The *Suspended* state references four *Transition Objects*, which identify the allowed state transitions to the *Ready*, *Running*, and *Halted* state and from the *Ready* state.

5.2.3.3 ProgramType Transitions

ProgramType Transitions are instances of the *TransitionType* defined in IEC 62541-5:2011, B.4 which also includes the definitions of the *ToState*, *FromState*, *HasCause*, and *HasEffect* references used. Table 7 specifies the Transitions defined for the *ProgramType*. Each Transition is assigned a unique *TransitionNumber*. The Notes column indicates when a cause is referencing a method and when effects are optional.

Table 7 – Program Transitions

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Transitions					
HaltedToReady	HasProperty	TransitionNumber	1	PropertyType	
	ToState	Ready		StateType	
	FromState	Halted		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
ReadyToRunning	HasProperty	TransitionNumber	2	PropertyType	
	ToState	Running		StateType	
	FromState	Ready		StateType	
	HasCause	Start			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToHalted	HasProperty	TransitionNumber	3	PropertyType	
	ToState	Halted		StateType	
	FromState	Running		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToReady	HasProperty	TransitionNumber	4	PropertyType	
	ToState	Ready		StateType	
	FromState	Running		StateType	
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition-EventType			
RunningToSuspended	HasProperty	TransitionNumber	5	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
	HasCause	Suspend			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition- EventType			
SuspendedToRunning	HasProperty	TransitionNumber	6	PropertyType	
	ToState	Running		StateType	
	FromState	Suspended		StateType	
	HasCause	Resume			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition- EventType			
SuspendedToHalted	HasProperty	TransitionNumber	7	PropertyType	
	ToState	Halted		StateType	
	FromState	Suspended		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition- EventType			
SuspendedToReady	HasProperty	TransitionNumber	8	PropertyType	
	ToState	Ready		StateType	
	FromState	Suspended		StateType	
	HasCause	Reset			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition- EventType			
ReadyToHalted	HasProperty	TransitionNumber	9	PropertyType	
	ToState	Halted		StateType	
	FromState	Ready		StateType	
	HasCause	Halt			Method
	HasEffect	ProgramTransitionEventType			
	HasEffect	AuditProgramTransition- EventType			

The *HaltedToReady* transition specifies the Transition from the *Halted* to *Ready* States. It may be caused by the *Reset Method*.

The *ReadyToRunning* transition specifies the Transition from the *Ready* to *Running* States. It is caused by the *Start Method*.

The *RunningToHalted* transition specifies the Transition from the *Running* to *Halted* States. It is caused by the *Halt Method*.

The *RunningToReady* transition specifies the Transition from the *Running* to *Ready* States. The *RunningToSuspended* transition specifies the Transition from the *Running* to *Suspended* States. It is caused by the *Suspend Method*.

The *SuspendedToRunning* transition specifies the Transition from the *Suspended* to *Running* States. It is caused by the *Resume Method*.

The *SuspendedToHalted* transition specifies the Transition from the *Suspended* to *Halted* States. It is caused by the *Halt Method*.

The *SuspendedToReady* transition specifies the Transition from the *Suspended* to *Ready* States. It is caused internally.

The *ReadyToHalted* transition specifies the Transition from the *Ready* to *Halted* States. It is caused by the *Halt Method*.

Two *HasEffect* references are specified for each Program Transition. These effects are events of type *ProgramTransitionEventType* and *AuditProgramTransitionEventType* defined in 5.2.5. The *ProgramTransitionEventType* notifies *Clients* of the Program Transition and conveys result data. The *AuditProgramTransitionEventType* is used to audit transitions that result from *Program Control Methods*.

The “Audit Server Facet” Profile defined in IEC 62541-7 requires support of the *AuditProgramTransitionEventType*.

Withd

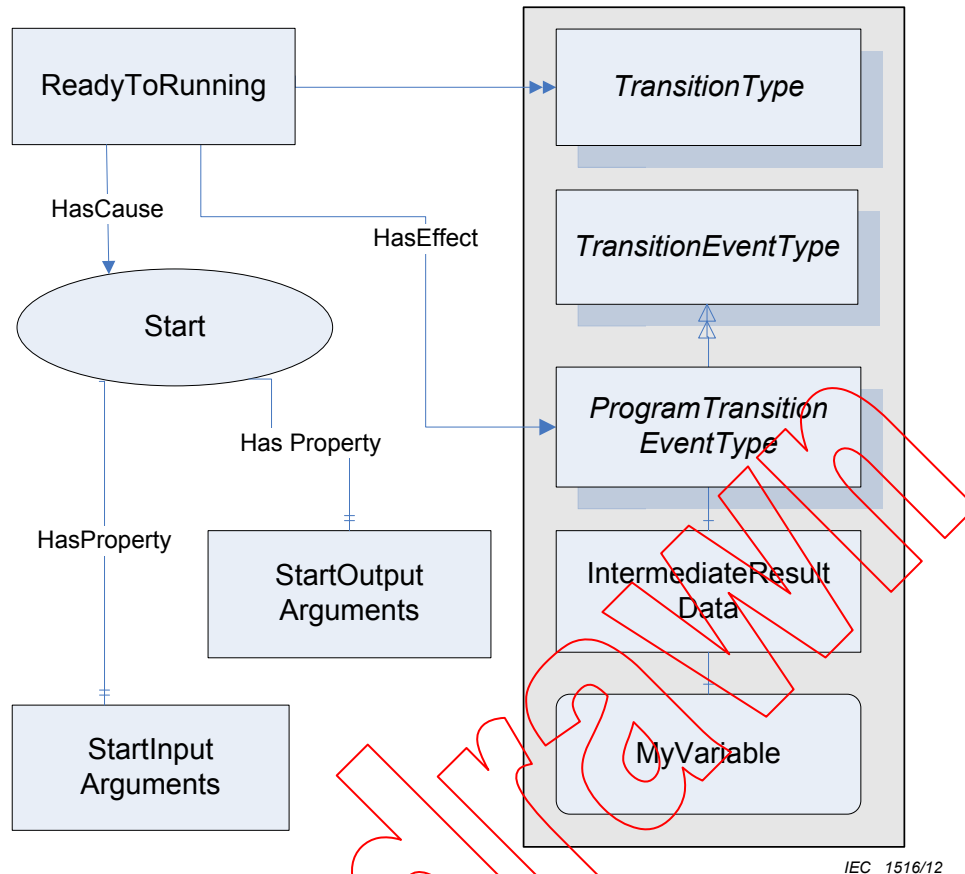


Figure 6 – ProgramType Causes and Effects

5.2.4 ProgramType Causes (Methods)

5.2.4.1 Overview

The *ProgramType* includes references to the *Causes* of specific *Program* state transitions. These causes refer to *Method* instances. *Programs* that do not support a *Program Control Method*, omit the *Causes* reference to that *Method* from the *ProgramType* references. If a *Method's Causes* reference is omitted from the *ProgramType*, a *Client* cannot cause the associated state transition. The *Method* instances referenced by the *ProgramType* identify the *InputArguments* and *OutputArguments* required for the *Method* calls to *Program* *Invocations* of that *ProgramType*. Table 8 specifies the *Methods defined as Causes for ProgramTypes*. Figure 6 illustrates the references associating the components and properties of *Methods* and *Events* with *ProgramTransitions*.

Table 8 – ProgramType Causes

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Causes					
Start	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Suspend	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Resume	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	NOTES
Halt	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional
Reset	HasProperty	InputArguments		PropertyType	Optional
	HasProperty	OutputArguments		PropertyType	Optional

The *Start Method* causes the *ReadyToRunning Program* transition.

The *Suspend Method* causes the *RunningToSuspended Program* transition.

The *Resume Method* causes the *SuspendedToRunning Program* transition.

The *Halt Method* causes the *RunningToHalted*, *SuspendedToHalted*, or *ReadyToHalted Program* transition depending on the *CurrentState* of the *Program*.

The *Reset Method* causes the *HaltedToReady Program* transition.

5.2.4.2 Standard Attributes

The *Executable Method* attribute indicates if a method can currently be executed. For *Program Control Methods*, this means that the owning *Program* has a *CurrentState* that supports the transition caused by the *Method*.

5.2.4.3 Standard Properties

5.2.4.3.1 Overview

Methods can reference a set of *InputArguments*. For each *ProgramType*, a set of *InputArguments* may be defined for the supported *Program Control Methods*. The data passed in the arguments supplements the information required by the *Program* to perform its function. All calls to a *Program Control Method* for each *Program Invocation* of that *ProgramType* shall pass the specified arguments.

Methods can reference a set of *OutputArguments*. For each *ProgramType*, a set of *OutputArguments* is defined for the supported *Program Control Methods*. All calls to a *Program Control Method* for each *Program Invocation* of that *ProgramType* shall pass the specified arguments.

5.2.5 ProgramType Effects (Events)

5.2.5.1 Overview

The *ProgramType* includes component references to the *Effects* of each of the *Program's* state transitions. These *Effects* are *Events*. Each *Transition* shall have a *HasEffect* reference to a *ProgramTransitionEventType* and can have an *AuditProgramTransitionEventType*. When the transition occurs, event notifications of the referenced type are generated for subscribed *Clients*. The *Program Invocation* may serve as the *EventNotifier* for these events or an owning object or the *Server Object* may provide the notifications.

ProgramTransitionEventTypes provide the means for delivering result data and confirming state transitions for subscribed *Clients* on each defined *Program State Transition*. The *AuditProgramTransitionEventType* allows the auditing of changes to the *Program's State* in conjunction with client *Method Calls*.

5.2.5.2 ProgramTransitionEventType

The *ProgramTransitionEventType* is a subtype of the *TransitionEventType*. It is used with Programs to acquire intermediate or final results or other data associated with a state transition. A Program can have a unique *ProgramTransitionEventType* definition for any transition. Each *ProgramTransitionEventType* specifies the *IntermediateResult* data specific to the designated state transition on that program type. Each transition can yield different *Intermediate* result data. Table 9 specifies the *ProgramTransitionEventType*.

Table 10 identifies the *ProgramTransitionEventTypes* that are specified for *ProgramTypes*.

Table 9 – ProgramTransitionEventType

Attribute	Value				
BrowseName	ProgramTransitionEventType				
IsAbstract	True				
References	NodeClass	BrowseName	Data Type	TypeDefinition	ModelingRule
Inherits the <i>Properties</i> of the base <i>TransitionEventType</i> defined in IEC 62541-5:2011, B.4.16.					
HasComponent	Object	IntermediateResult		BaseObjectType	Optional

TransitionNumber identifies the *Program Transition* that triggered the *Event*.

FromStateNumber identifies the *State* before the *Program Transition*.

ToStateNumber identifies the *State* after the *Program Transition*.

The *IntermediateResult* is an *Object* that aggregates a set of *Variables* whose values are relevant for the *Program* at the instant of the associated transition. The *ObjectType* for the *IntermediateResult* specifies the collection of *Variables* using a set of *HasComponent References*.

Table 10 – ProgramTransitionEvents

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
Effects					
HaltedToReadyEvent					
	HasProperty	TransitionNumber	1	PropertyType	
	HasProperty	FromStateNumber	1	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
ReadyToRunningEvent					
	HasProperty	TransitionNumber	2	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToHaltedEvent					
	HasProperty	TransitionNumber	3	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToReadyEvent					
	HasProperty	TransitionNumber	4	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
RunningToSuspendedEvent					
	HasProperty	TransitionNumber	5	PropertyType	
	HasProperty	FromStateNumber	3	PropertyType	
	HasProperty	ToStateNumber	4	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToRunningEvent					
	HasProperty	TransitionNumber	6	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	3	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToHaltedEvent					
	HasProperty	TransitionNumber	7	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	

BrowseName	References	Target BrowseName	Value	Target TypeDefinition	Notes
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
SuspendedToReadyEvent					
	HasProperty	TransitionNumber	8	PropertyType	
	HasProperty	FromStateNumber	4	PropertyType	
	HasProperty	ToStateNumber	2	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional
ReadyToHaltedEvent					
	HasProperty	TransitionNumber	9	PropertyType	
	HasProperty	FromStateNumber	2	PropertyType	
	HasProperty	ToStateNumber	1	PropertyType	
	HasComponent	IntermediateResults		ObjectType	Optional

5.2.6 AuditProgramTransitionEventType

The *AuditProgramTransitionEventType* is a subtype of the *AuditUpdateStateEventType*. It is used with *Programs* to provide a means to audit the *Program State Transitions* associated with any *Client* invoked *Program Control Method*. Servers shall generate *AuditProgramTransitionEvents* if they support the “Audit Server Facet” Profile defined in IEC 62541-7.

Table 11 specifies the definition of the *AuditProgramTransitionEventType*

Table 11 – AuditProgramTransitionEventType

Attribute	Value				
BrowseName	AuditProgramTransitionEventType				
IsAbstract	True				
References	NodeClass	BrowseName	Data Type	TypeDefinition	ModellingRule
Inherits the <i>Properties</i> of the <i>AuditUpdateStateEventType</i> defined in IEC 62541-5:2011, Annex B.4.17.					
HasProperty	Variable	TransitionNumber	UInt32	PropertyType	Mandatory

This *EventType* inherits all *Properties* of the *AuditUpdateStateEventType* defined in IEC 62541-5:2011, B.4.17 except as noted below IEC 62541-5.

The *Status Property*, specified in IEC 62541-5:2011, 6.4.3 “AuditEventType”, identifies whether the state transition resulted from a Program Control Method call (set *Status* to TRUE) or not (set *Status* to FALSE).

The *SourceName* specified in IEC 62541-5:2011, 6.4.2 “BaseEventType” identifies the *Method* causing the *ProgramTransition* when it is the result of a client invoked *ProgramControlMethod*. The *SourceName* is prefixed with “Method/” and the name of the *ProgramControlMethod*, “Method/Start” for example.

The *ClientUserId Property*, specified in IEC 62541-5:2011, 6.4.3 “AuditEventType”, identifies the user of the *Client* that issued the *Program Control Method* if it is associated with this *Program State Transition*.

The *ActionTimeStamp Property*, specified in IEC 62541-5:2011, 6.4.3 “AuditEventType”, identifies when the time the *Program State Transition* that resulted in the event being generated occurred.

The *TransitionNumber* property is a *Variable* that identifies the Transition that triggered the event.

5.2.7 FinalResultData

The *FinalResultData ObjectType* specifies the *VariableTypes* that are preserved when the *Program* has completed its function. The *ObjectType* includes a *HasComponent* for a *VariableType* of each variable that comprises the *FinalResultData*.

5.2.8 ProgramDiagnosticType

The *ProgramDiagnosticType* provides information that can be used to aid in the diagnosis of *Program* problems. This object contains a collection of *Variables* that chronicle the *ProgramInvocation*’s activity. Table 12 specifies the *Variables* that compose the *ProgramDiagnosticType*.

Table 12 – ProgramDiagnosticType

Attribute	Value			
BrowseName	ProgramDiagnosticsType			
IsAbstract	False			
References	NodeClass	BrowseName	Data Type / TypeDefinition	Modelling Rule
Subtype of the BaseObjectType defined in IEC 62541-5.				
HasComponent	Variable	CreateSessionId	NodeId	Mandatory
HasComponent	Variable	CreateClientName	String	Mandatory
HasComponent	Variable	InvocationCreationTime	UTCtime	Mandatory
HasComponent	Variable	LastTransitionTime	UTCtime	Mandatory
HasComponent	Variable	LastMethodCall	String	Mandatory
HasComponent	Variable	LastMethodSessionId	Int32	Mandatory
HasComponent	Variable	LastMethodInputArguments	InputArguments	Mandatory
HasComponent	Variable	LastMethodOutputArguments	OutputArguments	Mandatory
HasComponent	Variable	LastMethodCallTime	UTCtime	Mandatory
HasComponent	Variable	LastMethodReturnStatus	returnStatus	Mandatory

The *CreateSessionId* contains the *SessionId* of the session on which the call to the *Create Method* was issued to create the *Program Invocation*.

The *CreateClientName* is the name of the client of the session that created the *Program Invocation*.

The *InvocationCreationTime* identifies the time the *Program Invocation* was created.

The *LastTransitionTime* identifies the time of the last program state transition that occurred.

The *LastMethodCall* identifies the last *Program Method* called on the *Program Invocation*.

The *LastMethodSessionId* contains the *SessionId* of the session on which the last Program Control Method call to the *Program Invocation* was issued.

The *LastMethodClientName* is the name of the client of the session that made the last Method call to the *Program Invocation*.

The *LastMethodInputArguments* preserves the values of the input arguments on the last *Program Method* call.

The *LastMethodOutputArguments* preserves the values of the output arguments on the last *Program Method* call.

The *LastMethodCallTime* identifies the time of the last Method call to the *Program Invocation*.

The *LastMethodReturnStatus* preserves the value of the *returnStatus* for the last *Program Control Method* requested for this *Program Invocation*.

Withdrawn