NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
61970-1

Première édition
First edition
2005-12

Interface de programmation d'application pour système de gestion d'énergie (EMS-API) –

Partie 1: Lignes directrices et exigences générales

Energy management system application program interface (EMS-API) –

Part 1: Guidelines and general requirements

Numéro de référence
Reference number
CEI/IEC 61970-1:2005

Numérotation des publications

Depuis le 1er janvier 1997, les publications de la CEI sont numérotées à partir de 60000. Ainsi, la CEI 34-1 devient la CEI 60034-1.

Editions consolidées

Les versions consolidées de certaines publications de la CEI incorporant les amendements sont disponibles. Par exemple, les numéros d'édition 1.0, 1.1 et 1.2 indiquent respectivement la publication de base, la publication de base incorporant l'amendement 1, et la publication de base incorporant les amendements 1 et 2.

Informations supplémentaires sur les publications de la CEI

Le contenu technique des publications de la CEI est constamment revu par la CEI afin qu'il reflète l'état actuel de la technique. Des renseignements relatifs à cette publication, y compris sa validité, sont disponibles dans le Catalogue des publications de la CEI (voir ci-dessous) en plus des nouvelles éditions, amendements et corrigenda. Des informations sur les sujets à l'étude et l'avancement des travaux entrepris par le comité d'études qui a élaboré cette publication, ainsi que la liste des publications parues, sont également disponibles par l'intermédiaire de:

- **Site web de la CEI (www.iec.ch)**
- **Catalogue des publications de la CEI**

  Le catalogue en ligne sur le site web de la CEI (www.iec.ch/searchpub) vous permet de faire des recherches en utilisant de nombreux critères, comprenant des recherches textuelles, par comité d'études ou date de publication. Des informations en ligne sont également disponibles sur les nouvelles publications, les publications remplacées ou retirées, ainsi que sur les corrigenda.
- **IEC Just Published**

  Ce résumé des dernières publications parues (www.iec.ch/online_news/justpub) est aussi disponible par courrier électronique. Veuillez prendre contact avec le Service client (voir ci-dessous) pour plus d'informations.
- **Service clients**

  Si vous avez des questions au sujet de cette publication ou avez besoin de renseignements supplémentaires, prenez contact avec le Service clients:

  Email: custserv@iec.ch
  Tél: +41 22 919 02 11
  Fax: +41 22 919 03 00

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**
- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.
- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.
- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

NORME INTERNATIONALE

INTERNATIONAL STANDARD

CEI
IEC
61970-1

Première édition
First edition
2005-12

Interface de programmation d'application pour système de gestion d'énergie (EMS-API) –

Partie 1: Lignes directrices et exigences générales

Energy management system application program interface (EMS-API) –

Part 1: Guidelines and general requirements

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE X

Pour prix, voir catalogue en vigueur
For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

ENERGY MANAGEMENT SYSTEM APPLICATION PROGRAM INTERFACE (EMS-API) –

Part 1: Guidelines and general requirements

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 provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
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 61970-1 has been prepared by IEC technical committee 57: Power systems management and associated information exchange.

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

FDIS	Report on voting
57/777/FDIS	57/795/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.

IEC 61970 consists of the following parts, under the general title *Energy management system application program interface (EMS-API)*:

Part 1: Guidelines and general requirements

Part 2: Glossary

Part 301: Common Information Model (CIM) base

Part 302: Common information model (CIM) financial, energy scheduling and reservations[1]

Part 401: Component interface specification (CIS) framework

Part 402: Component interface specification (CIS) – Common services[1]

Part 403: Component Interface Specification (CIS) – Generic data access[1]

Part 404: Component Interface Specification (CIS) – High speed data access[1]

Part 405: Component Interface Specification (CIS) – Generic eventing and subscription[1]

Part 407: Component Interface Specification (CIS) – Time series data access[1]

Part 453: Exchange of Graphics Schematics Definitions (Common Graphics Exchange)[1]

Part 501: Common Information Model (CIM) XML Codification for Programmable Reference and Model Data Exchange

The committee has decided that the contents of this publication will remain unchanged until the maintenance result 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.

[1] Under consideration.

INTRODUCTION

This standard is part of the IEC 61970 series that defines application program interfaces (APIs) for an energy management system (EMS). This standard is based to a large extent upon the work of the EPRI Control Center API (CCAPI) research project (RP-3654-1). The principle objectives of the EPRI CCAPI project are to:

– reduce the cost and time needed to add new applications to an EMS or other system[2];

– protect the investment in existing applications that are working effectively;

– improve the capability to exchange information between disparate systems both within and external to the control center environment.

The technical approach is to provide an integration framework for interconnecting existing applications/systems that is

– based on a common architecture and information model;

– independent of the underlying technology.

The principal task of the IEC 61970 series of standards is to develop a set of guidelines and standards to facilitate 1) the integration of applications developed by different suppliers in the control center environment[3] and 2) the exchange of information to systems external to the control center environment. The scope of these specifications includes other transmission systems as well as distribution and generation systems external to the control center that need to exchange real-time operational data with the control center. Therefore, another related goal of these standards is to enable the integration of existing legacy systems as well as new systems built to conform to these standards in these application domains.

The complete set of standards includes the following parts:

Part 1: Guidelines and general requirements

Part 2: Glossary

Part 3XX: Common Information Model (CIM)

Part 4XX: Component Interface Specification (CIS)

Part 5XX: CIS Technology Mappings

IEC 61970-1 provides a set of guidelines and general infrastructure capabilities needed for the application of the EMS-API interface standards. It describes the reference model that provides the framework for the application of the other parts of the EMS-API standards. This reference model is based on a component architecture, which places the focus of the standards on component interfaces for information exchange between applications in a control center environment. The model is also applicable to similar information exchanges between control center applications and systems external to the control center environment, such as other control centers, Independent System Operators (ISOs), Regional Transmission Organizations (RTOs), and Distribution Management Systems (DMS).

IEC 61970-1 also includes general capabilities for the integration infrastructure, which while not part of this standard, is expected to provide certain essential services to support the EMS-API interface standards.

---

2 Ideally, an application should be installed on a system with minimal effort and no modification of source code; i.e., the way software packages are installed on a desktop computer. The EMS-API Project goal is to at least approach that ideal by reducing the often significant efforts currently required to install third-party applications in an EMS.

3 The control center environment includes traditional transmission control within a utility as well as the newer Independent System Operators (ISOs) and Regional Transmission Operators (RTOs), which are not affiliated with any one utility.

ENERGY MANAGEMENT SYSTEM APPLICATION PROGRAM INTERFACE (EMS-API) –

Part 1: Guidelines and general requirements

1 Scope

This part of the IEC 61970 series provides a set of guidelines and general infrastructure capabilities required for the application of the EMS-API interface standards. This part of the IEC 61970 series describes typical integration scenarios where these standards are to be applied and the types of applications to be integrated. A reference model is defined to provide a framework for the application of the other parts of these EMS-API standards. This reference model is based on a component architecture that places the focus of the standards on component interfaces for information exchange between applications in a control center environment. While the primary objective of the EMS-API is to support the integration of applications within the control center, the reference model is also applicable to information exchanges between control center applications and systems external to the control center environment, such as other control centers, ISOs, RTOs, and distribution systems. This standard describes the role of the other parts of the standard, including the Common Information Model (CIM) in the IEC 61970-3XX series, the Component Interface Specifications (CIS) in the IEC 61970-4XX series, and Technology Mappings in the IEC 61970-5XX series.

This part of the IEC 61970 series also includes general capabilities that are needed by the integration infrastructure to facilitate the exchange of information via the component interfaces specified by the CIS. While the integration infrastructure itself is not part of this standard, it is expected to provide certain essential services to support the EMS-API interface standards. These services are enumerated in Clause 6.

This part of the IEC 61970 series does not specify individual implementations or products, nor does it constrain the representation of information within a computer system application. This standard specifies the externally visible interfaces, including semantics and syntax, required to support the interoperability of products supplied by different vendors.

2 Normative References

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

IEC 61970-2, *Energy management system application program interface (EMS-API) – Part 2: Glossary*

IEC 61970-301, *Energy management system application program interface (EMS-API) – Part 301: Common Information Model (CIM) base*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 61970-2 apply.

4 System integration

4.1 Integration scenarios

Energy management systems are an assemblage of various software subsystems (SCADA, generation control, load forecast, etc.). The guidelines created for this area (or portions thereof) apply to several different integration scenarios. This work recognizes that existing systems with their unique interfaces will need to evolve to use the interfaces specified in this work. The following different types of integration situations are envisaged. While these are typical scenarios, they are only a subset of the possible examples.

a) Integration of applications developed by different suppliers into one homogenous system:

   In this scenario, independently developed application components (such as an Optimal Power Flow (OPF)) are integrated into a system (which includes supporting infrastructure). This allows the system integrator to more easily integrate an individual component into any system, whether it is an existing system or a newly developed system.

b) On-line data exchange between separate systems:

   The control center needs to communicate with other, separate systems within the corporation and between enterprises. Loosely coupled integration schemes are needed to exchange information. Examples include information exchanges with a DMS, the corporate accounting system, or another EMS. Enterprise integration scenarios generally fit into this category.

c) Integration of separate systems sharing some engineering data:

   This corresponds to the situation where application packages from different vendors use partly overlapping engineering modeling data (such as the impedance of a line segment).

d) Exchange of serialized data between the same applications in different systems:

   Limited integration can be achieved by using file transfer techniques. An agreed upon format is used for this type of export/import exchange. The use of File Transport Protocol (FTP) and an eXtensible Markup Language (XML)-based format file (or document) for the exchange of power sysem models is an example of this scenario.

e) Developing a new application in a homogenous system:

   This corresponds to vendors or utilities developing new applications for integration into their existing systems (as opposed to integration in other systems) using these standards for the application interface.

4.2 Integration considerations

4.2.1 General

The scope of integration to be supported by these standards falls into two loosely defined categories:

a) Integration of software components to implement an EMS or similar system.

b) Integration of independent systems.

To properly interact, both software components and systems need a Common Information Model (CIM) to provide a common, consistent meaning (i.e., semantics) to the information exchanged or accessed. For example, to exchange information about a transformer impedance, the classification of the piece of equipment as a transformer is needed together with the attribute name for the impedance value. With this information, a particular instance of a transformer can be identified with its corresponding impedance. The semantic name of these real world objects (together with their attributes, descriptions, and relationships to other real world objects) is provided by the CIM (see 6.2).

However, in other respects, these two categories of integration as discussed in the following clauses have slightly different needs. The intent of the EMS-API series of standards is to address both, and both are viewed as presenting component interfaces.

4.2.2 Integration of software components into a system

4.2.2.1 General

To integrate independently developed software components into a system, several issues need to be addressed.

a) Software component interactions:

   Software components need to interact in a collaborative fashion for the system to function properly. This interaction can be in the form of property access, method invocation, and event handling. The public interfaces consisting of properties, methods, and events shall be identified, and a contract on their use shall be specified in order to support integration of software components. Where similar interaction scenarios exist within the system, the specification of consistent interface patterns simplify system integration and maintenance.

b) Public engineering model data for initialization:

   The physical power system and related information is simulated with engineering data as reflected in the CIM. Software components share aspects of this engineering data to perform their functions. Upon software component start-up, a component needs to be initialized with a consistent, accurate model of the real-world system it is simulating. A common interface for access to this public, shared data provides a consistent mechanism for software components to initialize their internal models. Once initialized, software component interaction mechanisms can be used to keep their engineering models up to date.

c) Packaging for deployment:

   Software components need to be realized in specific forms for packaging and delivery to system integration. A few major technology frameworks exist in the software industry today, each with their own format for packaging. The specification of standards should promote flexibility in implementation possibilities while facilitating the integration of independently developed software. To accomplish this, software component interactions should be specified in an abstract form that is capable of particular technology and language specializations.

To support this type of integration scenario, the properties, methods, and events used at the component interface need to be standardized as well as the data content of the information exchange.

4.2.2.2 Application categories

The scope of the EMS-API standards includes all applications typically found within a control center environment as well as interfaces to external systems needed to support real-time operations. However, since the intent of the EMS-API standard is to define interface standards rather than to define standard applications, the scope of the project can best be understood by considering the list of application categories that will be supported by the EMS-API standards. The actual packaging of component interfaces specified in the CIS into applications is left to the application suppliers; therefore, any attempt to define a list of individual applications by name would unnecessarily constrain suppliers.

The list of application categories and typical applications supported by this standard can be found in Table B.1. The following is a summary of the list:

- supervisory Control and Data Acquisition (SCADA);
- alarm processing;
- topology processing;
- network applications;
- load management;
- generation control;
- load forecast;
- energy/transmission scheduling;
- accounting settlements;
- maintenance scheduling;
- historical information archival;
- data engineering;
- generic user interface;
- dynamic simulation;
- dispatcher training simulator;
- external systems (e.g., DMS, weather, wholesale power marketing, etc.).

4.2.3 Integration of systems

In the integration of heterogeneous systems, where there is likely to be no similarity in the operating environment nor control over it, the emphasis is more on supporting loosely coupled, asynchronous "document" exchanges. In this context, a "document" may be thought of as a large, rich data structure, such as an XML document. Document exchange implies that the data exchanged is complex, structured, and self-describing. The exchange is more likely to involve individual, atomic information transfers where all information regarding how to handle the information and/or action requested in the transfer is self-contained, rather than multi-step transactions where the handling of the information transfer may be contingent upon previous information transfers or events.

4.3 Component-based interfaces

One goal of the EMS-API standards is to encourage the independent development of reusable software components and facilitate their integration in the construction of control center systems through the development of component interface standards. The software industry, including the major Energy Management System (EMS) vendors and suppliers of application software for an EMS, has undergone an evolution from basing software engineering concepts on top down modular software design to object-oriented approaches to the latest refinement using component-based architectures. The component models embraced by the Common Object Request Broker Architecture (CORBA®)[4], Sun®[5]'s Enterprise Java Beans® (EJB®), and Microsoft®[6]'s Distributed Common Object Modeling (DCOM) best exemplify this trend. (See Annex A for a description of these three component models).

These component-based approaches also facilitate the integration of software and complete systems from various sources. For this type of any-to-any integration, XML Web Services provide another Internet-based integration model. XML Web services allow applications to communicate and share data over the Internet using the type of information exchange earlier referred to as "document exchange," regardless of operating system or programming language. These services provide another example of a component execution environment becoming more prevalent in Business-to-Business (B2B) information exchanges. (See Annex A for a description of XML Web Services).

The effect on the EMS-API is to shift the focus to developing standards for software component interfaces for exchanging and accessing public information rather than on standardizing the integration framework services that provide these capabilities. The expectation is that applications that adhere to these standards can then be individually delivered and reused in multiple systems. While other infrastructure services may be needed in an actual system, such as directory services and security, these are not the goal of this standard. In fact, they may be more properly considered the domain of the system integrator or system supplier (i.e., part of the EMS system platform, not a reusable plug-and-play component).

The goal is not to develop standard interfaces for middleware. In fact, the goal is just the opposite – to be independent of any particular set of middleware services. This allows integrators to select the right type and scale of infrastructure for each system. It allows service designs to evolve and innovate, and it simplifies component development as well.

It also means that a software developer does not have to deal directly with these services. The system integrator provides the "glue" to insert these components into the system environment. This gives the integrator more freedom to configure components and choose the services best suited to the needs of the system being implemented (i.e., performance, availability, etc.).

---

4 CORBA is the trade name of a product supplied by OMG. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

5 Sun is shorthand for Sun Microsystems, a USA-based corporation. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the company named.

6 Microsoft is shorthand for Microsoft Corporation, a USA-based corporation. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the company named.

The following two examples illustrate this independence of components:

– CORBA components specifically do not require the use of CORBA Notification (or any particular service) as its event system.

– COM+ components are written exactly the same way whether they are deployed in an environment with or without Distributed Transaction Coordinator (DTC) and/or a Microsoft Message Queue (MSMQ).

Table 1 lists some of the benefits of this component-based approach to interface standards.

Table 1 – Benefits of Component-based Interfaces

Explicitly addresses the EMS-API project software plug-and-play goal
Does not prescribe an overall system design nor the choice and design of services
Aligns with overall software industry direction to permit the use of mainstream tools to develop components and configure systems
Frees the EMS-API project from rediscovering and reinventing solutions for the many problems of "plug-and-play" software
Provides a more complete solution including important areas such as packaging (what goes on the component disc CD), documentation, versioning, etc.
Does not require designing and/or prescribing middleware services, particularly event, naming and transaction services, but rather permits the use of commercially-available products to provide these services
Provides a canonical form for the utility specific standards developed on the project, enabling developers to directly machine-translate interface definitions into their preferred interface language: Object Management Group (OMG)/International Standards Organization (ISO) Interface Definition Language (IDL), Java®[7] interface syntax or Microsoft IDL
Allows the EMS-API project to focus on designing interfaces and events for utility applications without having to wait for all the middleware problems to be solved. This permits vendors to get to market quicker with EMS-API-compliant interfaces in their application products

4.4 Relationship to IEC 61968 series of standards

The IEC 61968 series of standards deals with system interfaces for distribution management systems. There is a great deal of similarity between these standards and those contained in the IEC 61970 series of standards, not only because of some overlap in scope, but because the IEC 61968 series of standards are also based on the CIM. The IEC 61968 series of standards build on the CIM Base specified in IEC 61970-301 wherever possible by extending it to include additional specializations of existing classes, but also adding entirely new sets of classes to model objects found in the distribution problem domain. Therefore, to comprehend the entire scope of the CIM, it is necessary to review both the IEC 61970 and IEC 61968 series of standards which deal with the CIM.

The IEC 61968 series of standards are also concerned with defining standard information exchanges between distribution business functions, similar to the IEC 61970-4XX standards, but do not attempt to define application program interfaces (i.e., services to be implemented by components). However, the IEC 61968 series of standards envision that the standard messages defined therein can be transferred over the APIs defined in these IEC 61970 series of standards.

---

[7] Java is the trade name of a product supplied by Sun Microsystems. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

5 EMS-API reference model

5.1 General

The EMS-API reference model is an abstract architecture that provides a visualization of the problem space being addressed, provides a language for describing and discussing solutions, defines terminology, and provides other similar aids toward achieving a mutual understanding of the problem being solved with the EMS-API standards.

The reference model is not a design, nor is it intended to describe software layers, although a layering approach is hard to avoid and is implied. The primary function is to show clearly which parts of the problem space are the subject of the EMS-API standards, and by contrast, which are outside the domain of the EMS-API project and why. It is also intended to show how different parts of the standard relate to each other.

Figure 1 is a diagram of the reference model, with the shaded areas representing those portions of the reference model that are the subject of this standard. The non-shaded areas represent parts of a system that are essential for creating a framework for reusable application components. Each part of the model is discussed later in this document.

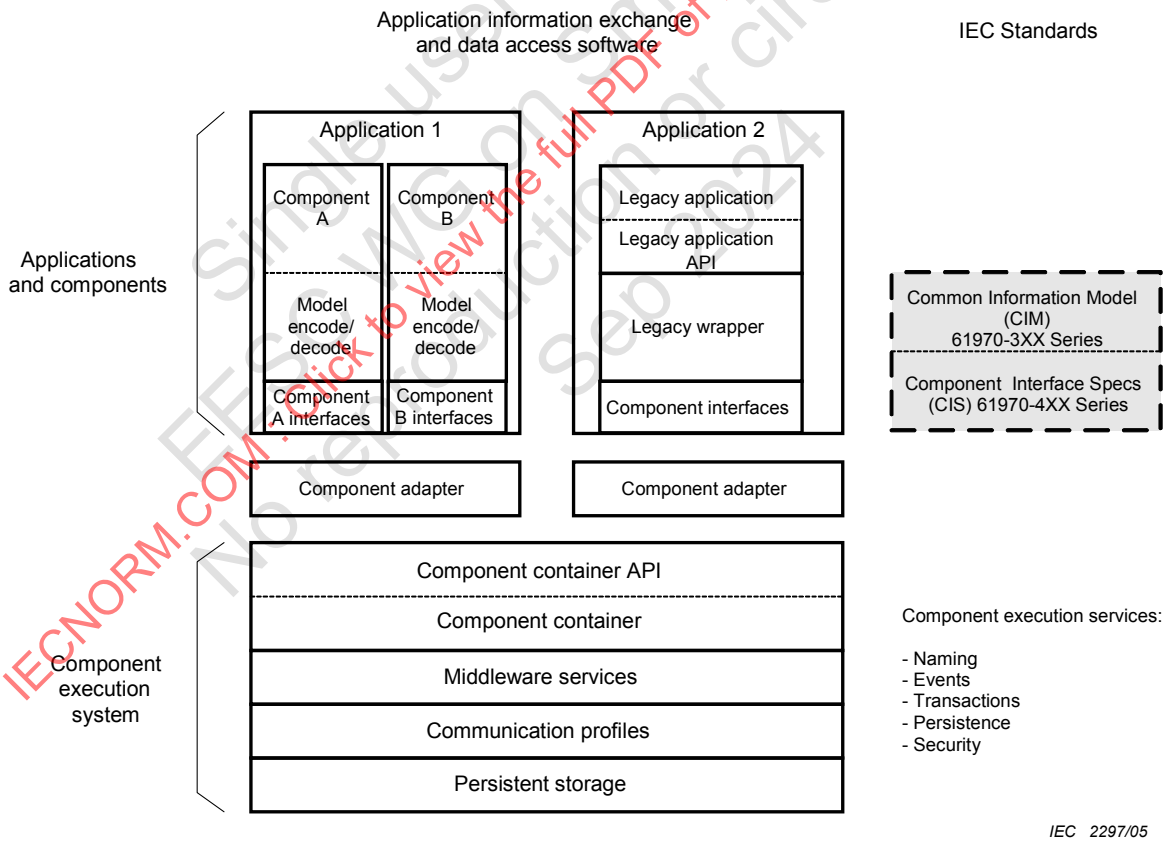

Figure 1 – EMS-API Reference Model

5.2 Control center environment

The reference model is specifically intended to apply to control center environments, which typically comprise networks of computers connected by Local Area Networks (LANs) and sometimes Wide Area Networks (WANs). A control center may include a variety of systems in support of utility operations, including EMS, DMS, and other systems needed for ISO and RTO business functions. It supports a number of different user groups and organizational functions, including on-duty operators, supervisors, operator training, operations planning, database maintenance, and software development. In an EMS, many applications are used in a number of these contexts, and it is important that an application can be easily configured (preferably automatically) for use in multiple contexts. This is accomplished through the use of *properties* that are associated with each component interface rather than through changes to the internal code of the component.

5.3 Application context

An application context comprises a collection of applications working together as an organizational unit to accomplish some high-level objective.[8] An application context defines a time frame and an environment for execution. Table 2 includes examples of contexts for EMS applications:

Table 2 – Examples of EMS application contexts

Real Time	On-line control of the power system
Operations Study	Execution of network applications to study and/or analyze operations practices (near-term)
Extension Planning Study	Execution of network and/or simulation studies to evaluate alternatives (future/long term)
Training	Provide training environment for operators, requiring simulation and analysis applications

While not shown explicitly in the reference model, it is understood that several contexts may exist simultaneously, each involving potentially the same applications in different data exchanges. Furthermore, there may be multiple instances of a particular context coexisting. For example, there may be two or more studies running in parallel for two different operators in the Operations Study context, while concurrently the same applications are executing in the Real Time context.

5.4 Application

An application comprises one or more components that perform some business function in a given domain. It is designed and written by a domain expert. The granularity of the components comprising the application is the designer’s choice. Application builders can combine components from different developers or different vendors to construct an application.

8 It should be noted that the term *context* is used differently in the discussion of component models for EJB and CORBA, for example, where context is used in connection with a component container providing component implementations with access to the runtime services implemented by the container. These services include transactions, security, events, and persistence.

An application developer should be able to make full use of a component without requiring access to its source code. Components can be customized to suit the specific requirements of an application through a set of external property values. For example, the button component has a property that specifies the name that should appear on the button. Of course, the amount of customization allowable depends on the foresight of the component developer in providing sufficient external property values. This is somewhat equivalent to the older concept in program design of customizing a program through specifying appropriate values for configurable parameters rather than having to modify source code.

Refer to Table B.1 for a list of application categories, abstract names, and functions performed.

5.5 Component

A component is a reusable software building block. It is a pre-built piece of encapsulated application code that can be combined with other components and with handwritten code to rapidly produce a custom application.

In order to qualify as a component, application code shall provide a standard interface that enables other parts of the application to invoke its functions and to access and manipulate the data within the component. The component model defines the structure of the interface.

Components vary in their granularity. A component can be very small, such as a simple Graphic User Interface (GUI) widget (e.g., a button) or it can implement an entire complex application, such as a state estimator application. In the latter case, the application could be designed from scratch as a single component, or it could comprise a legacy application wrapped to conform to component interface standards (see below for a discussion of legacy applications).

A component can be put on a transportable medium, such as a CD, and shipped for use in a system that provides component containers.

Generally, components publicly expose methods, properties, and "events" via an integration infrastructure. Events are of particular interest because they allow the integration of independently-developed components. By using a standard set of events, Component A need not know any other details of the interface to Component B or even if Component B exists. The key point is that it is the interface to the component that is standardized, not the integration infrastructure.

5.6 Legacy application and wrappers

A legacy application is quite different from the definition of an application given earlier. A legacy application can be a single application performing some business function that a utility may have purchased or developed itself prior to establishing any component model for integration purposes, or it could be an entire system that is a source/sink for data needed/published by other systems that needs to be integrated together to facilitate information exchange.

Examples include:

– a Unit Commitment application implemented in FORTRAN without regard for component interfaces;

– an entire transmission EMS without open, published interfaces that is needed to supply SCADA data to distribution systems and in turn receive updates to its power system model from outage management systems.

A legacy wrapper is used to encapsulate a legacy application or system that does not conform to the component interface standards. It converts a legacy program input/output into one or more component interfaces so it can participate in information exchange in a component-based system architecture.

The purpose of using a legacy wrapper, then, is to permit a legacy application or system to operate as a plug-and-play component capable of exchanging information with other components via a common infrastructure or framework.

Legacy wrappers may be designed and implemented by the domain expert who developed or owns the legacy application or system (e.g., the EMS vendor), who then supplies the wrapper for use in multiple system installations where component technology is used. Alternatively, where the legacy program is a one-of-a-kind custom "enterprise" application, the system integrator may have to write the wrapper and it would end up being used in only one system.

5.7 Component model

A component model defines the basic architecture of a component, specifying the structure of its interfaces and the mechanisms by which it interacts with its container and with other components. The component model provides guidelines to create and implement components that can work together to form a larger application.

A component builder should not have to implement multithreading, concurrency control, resource-pooling, security, and transaction management in every component. Furthermore, if these services were implemented in each component, achieving true plug-and-play application assembly would be very difficult. A component model standardizes and automates the use of these services.

There are four primary component models with wide acceptance within the software industry today. These component models are described in Annex A. The software industry has decided, as specified in all four of these component models, that removing container or infrastructure awareness from components is a major step toward a world where components can be independently developed, assembled and deployed. However, since there *are* four separate models, component vendors may want to provide a slightly different version for each to facilitate the use of all available features of the underlying container and execution system, with the inherent performance advantages that may entail. This is not strictly necessary, however. As long as component vendors constrain their component design to ensure correct operation in any of the four models, the differences can be made up by component adapters supplied by the system integrator who has chosen the specific component technology to be used for a specific system implementation. Alternatively, the system integrator may choose to combine more than one component technology and use bridging technologies to permit interoperation of the different technologies (e.g., CORBA execution system interoperating with a Microsoft DCOM execution system).

5.8 Component container

Components execute within a container. A container provides a context for one or more components and provides management and control services for the components. In practical systems, a container provides an operating system process or thread in which to execute the component. It insulates the component from the runtime platform. When a client invokes a server component, the container automatically allocates a process thread and initiates the component. The container manages all resources on behalf of the component and manages all interactions between the component and the external systems.

Component containers are typically provided by the system supplier as part of the component execution system. Typical services provided on behalf of the components are naming, events, transactions, security, and persistence. All services needed by a utility real-time application may not be provided as part of the standard container services offered by commercial software suppliers. Annex C describes how these services may be provided in an actual implementation.

5.9 Component adapter

The operations and behaviors of a container are defined and dictated by its component model. The component model provides a contract for how the container services and interfaces shall be provided. As a result, components developed for one type of execution system or environment are usually not directly portable to any other type of execution environment. Therefore, in order to achieve reuse with multiple execution systems, a component adapter is required for any execution system except the one for which the component was designed. Alternatively, component interfaces could be defined according to some neutral standard, and then a component adapter would be required for all containers to map the standard interfaces onto those provided by the container. This is somewhat equivalent to the “portability layer” that is part of the Java Enterprise Platform, which is the component execution system for EJB.

A component adapter is defined as a piece of software that sits between the application (or component) and the component container and integration infrastructure, which provides fundamental component support services (e.g., publish/subscribe, message queuing, naming, etc). The adapter can, if needed, take care of protocol differences, data transformation, and data translation. An adapter can additionally provide a component support for security, transactions and persistence (at least from the perspective of the world outside the application) if the component execution environment itself does not provide these.

Component adapters deal with both the differences between a) the component interface and the chosen execution system environment and container technology and b) a component interface and other component interfaces used in the rest of the system. This includes differences in event definitions. These differences arise because:

– The system is made up of independently developed components whose interfaces were not coordinated in advance (i.e., not standardized, which is probably the usual case).

– The system contains partially standardized components, but the non-standard parts are incompatible.

– The standardized parts are still incompatible because they reflect different versions of the standard.

Depending on the environment and tools provided by the component container, the component adapter might also connect the component's event portals to the appropriate event topics and locate the correct dependency components, if any.

Component adapters may also implement the utility specific services needed for components to operate in the EMS contexts identified earlier that are not available with off-the-shelf component execution systems.

A system integrator or execution system supplier typically provides component adapters. They typically consist of code written to 1) match the event types and data types and services expected by the component to those expected by other components in the particular system being implemented, and 2) configure the system information flows for the component. As a result, they are custom crafted per system and per component by the system integrators.

5.10 Component execution system

Server components execute in an environment provided by an application or component execution system. The component execution system term encompasses the entire reference model from the container layer down, including the component container, middleware services, and communication profiles. It includes the other normal platform-supplied services not shown as well, including the operating system, persistent storage, etc. These are also referred to as container systems since the container provides the primary interface to the interface standards that are the subject of this standard.

Container systems include existing middleware products that adhere to the container contract and policies for the component model embraced by the system. Any execution system frameworks that adhere to the container contract for supporting components qualify. For example, an EMS supplier could design the EMS application execution system to support containers, thereby permitting the use of components either developed internally or purchased from another component supplier. Other commercial examples are contained in Annex D.

The system supplier typically provides component execution systems.

5.11 Middleware

The term middleware is used to describe a diverse group of software products that function as an integration, conversion, and/or translation layer. Middleware provides generic interfaces for events, messaging, data access, transactions, etc.

Middleware vendors provide products to support some combination of these generic services via their own proprietary interfaces. Generally, they do not target a particular industry, although they often provide converters for widely used applications, such as PeopleSoft®[9], SAP, etc. They may or may not provide all the services needed in a utility real-time operations environment.

9 PeopleSoft is the trade name of a product supplied by Oracle. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

Each utility may choose a different middleware vendor as a result of an enterprise-wide Information Technology (IT) decision to set a corporate standard, which may not be easily changed. Therefore, the EMS-API component interface standards shall be written so that they can be deployed with a variety of middleware products.

Middleware products are constantly evolving. The EMS-API component interface standards should not preclude the use of the best available products when an integration effort is undertaken. Annex D provides some examples of commercial products.

5.12 Communication profiles

Communication profiles specify the particular protocols and protocol services that are to be used for information exchange between separate server platforms in a component execution system. Some middleware products operate over standard profiles, such as CORBA and EJB that use Internet Inter-ORB Protocol (IIOP) over Transport Control Protocol/Internet Protocol (TCP/IP) for operation over either the Internet or private intranets. Others use proprietary protocols and services.

The intent of the EMS-API interface specifications is to be independent of any particular communication profile.

5.13 Reference model examples

Two examples are provided to illustrate the application of the reference model concepts to actual systems.

5.13.1 An example of an EMS built with EMS-API standards

Figure 2 illustrates an EMS system that conforms to the EMS-API reference model. Individual application components are interconnected via a component execution system and component adapters that provide the infrastructure services needed by the components to discover and communicate with each other and with the public data stores in the various EMS contexts. For this example, a CORBA component execution system that directly supports CORBA components is assumed. Applications that conform to the CORBA component model then do not need component adapters. Real-time operational SCADA data is obtained from 1) a legacy SCADA system that is wrapped with a legacy wrapper to provide the necessary component interfaces and 2) an Inter-Control Center Communication Protocol (ICCP) data server connected to other control centers or substations. Both sources of SCADA data publish updates using identical component interfaces as they become available, and applications that need real-time updates subscribe to receive them independent of which source supplies the data. A DMS system external to the control center is also shown and could receive the same SCADA data updates as they arrive by also subscribing. Many other application interactions are supported as well.

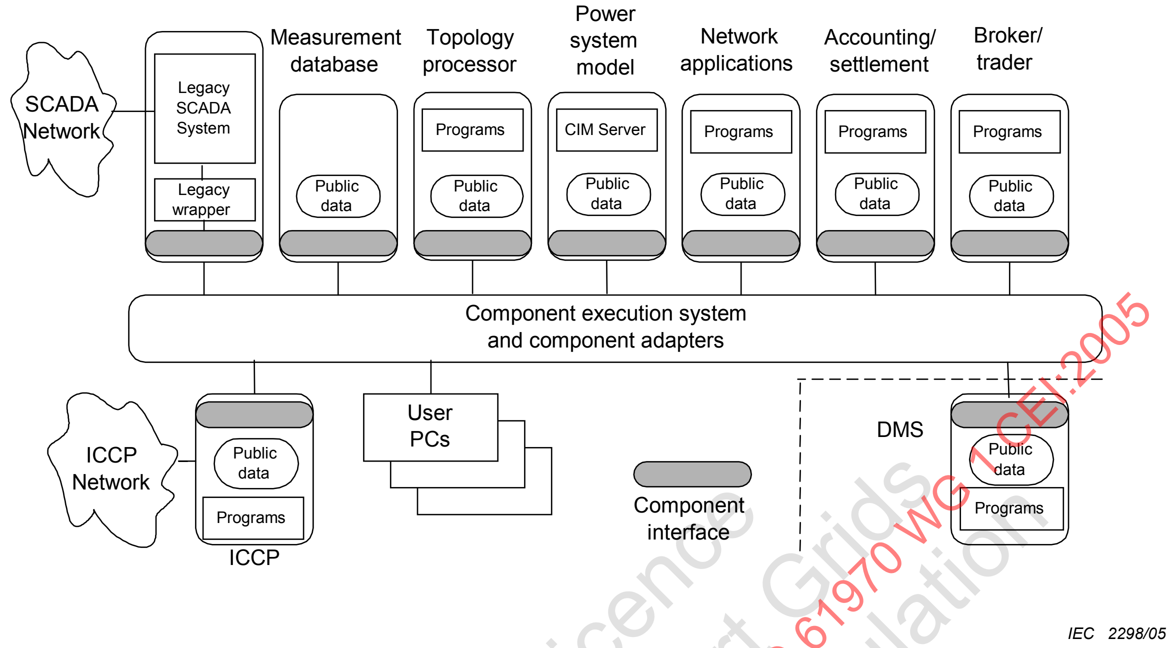

Figure 2 – EMS using EMS-API component standard interfaces

5.13.2 A scenario using both legacy wrappers and component adapters

An alternative example is of a SCADA vendor who has an existing system (for example, a typical SCADA data acquisition and control application) that needs to be packaged as a CORBA component so it can be sold as a plug-and-play application to be interfaced to a CORBA container system. The existing SCADA application already has a proprietary API, which has existed for years, is proven and well tested, and used in a variety of installations. There is little cost justification in rewriting any portion of the application just to make it a CORBA component.

The logical approach then is to front-end this “legacy application” with a wrapper that supports the needed component interfaces. The wrapper presents the application (or more likely a small part of the application) as a standard CORBA component. This wrapper would likely be responsible for mapping the existing proprietary API into that specified for the component. This mapping could be quite elaborate.

Now a customer that has purchased the SCADA vendor's componentized SCADA application wants to integrate it into his enterprise. What is almost certain today is that the majority of the applications to be integrated with the SCADA application will not be compliant to a single standard component model, or for that matter, any standard component model.

The customer has chosen an integration middleware product to help with the overall integration task. One of the primary reasons the utility bought the integration middleware product is that it facilitates the adaptation of the various application interfaces, component or otherwise, to the common services and information exchange model provided by the integration middleware product. The enterprise integration middleware product may not support the component model used by the SCADA component directly, or may not support a component model at all. The middleware vendor or the system integrator thus introduces a component adapter.

This scenario illustrates the distinction between a legacy wrapper and a component adapter in the reference model. Here, the wrapper is the responsibility of the SCADA vendor and the adapter, the middleware vendor (obviously the utility could have chosen to use the proprietary SCADA interface, and the wrapper would not even be introduced). To the extent the wrapper presents the SCADA application as a standard component, the middleware vendor's adapter can be standardized. In fact, integration middleware vendors offer standard adapters, typically for Relational Data Base Management Systems (RDBMS) (e.g., Oracle, Structured Query Language (SQL) Server, etc.), popular standard applications (e.g. SAP, Peoplesoft, etc.), and other common resources (e.g., MQSeries, etc.).

6 EMS-API standards

6.1 General

As shown in the reference model, the Common Information Model (CIM) and Component Interface Specification (CIS) are the major parts of the model that are being standardized. In fact, these are parts of the IEC 61970 series of standards being prepared in the IEC Technical Committee 57 under the name EMS-API. The actual document structure for IEC 61970 is as follows:

Part 1: Guidelines and general requirements

Part 2: Glossary

Part 3XX: Common Information Model (CIM)

Part 4XX: Component Interface Specifications (CIS)

Part 5XX: CIS technology mappings

6.2 CIM (IEC 61970-3XX)

6.2.1 General

SCADA/EMS/DMS application components in general require an elaborate model including measurements, network connectivity, device characteristics, etc. to be able to operate. The CIM provides this model, supplying these application components with a comprehensive logical view of a power system. The CIM is an abstract model that represents all the major objects in an electric utility enterprise typically contained in an EMS information model that are needed by multiple applications. This model includes public classes and attributes for these objects, as well as the relationships between them (see IEC 61970-301). This model is described in the Unified Modeling Language (UML) and maintained as single unified Rational ROSE model file. Major portions of the IEC 61970-3XX documents are auto-generated from this model file using Rational SODA.

The CIM is part of the overall EMS-API framework. By providing a standard way of representing power system resources as object classes and attributes, along with their relationships, the CIM facilitates the integration of EMS applications developed independently by different vendors, between entire EMS systems developed independently, or between an EMS system and other systems concerned with different aspects of power system operations, such as generation or distribution management. This is accomplished by defining a common language (i.e., semantics and syntax) based on the CIM to enable these applications or systems to access public data and exchange information independent of how such information is represented internally.

6.2.2 Intended use of the CIM

The objects represented in the CIM are abstract in nature and may be used in a wide variety of applications. The use of the CIM goes far beyond its application in an SCADA/EMS/DMS. This standard should be understood as a tool to enable integration in any domain where a common power system model is needed to facilitate interoperability and plug compatibility between applications and systems independent of any particular implementation. Specific anticipated uses of the CIM include:

– Initialize application components with configuration data.

  Typically, before an application component can be made operational, it shall be initialized with current state and event information in a configuration phase (sometimes called data engineering) based on this model. During the lifetime of an application component, extensions are made to the power system requiring changes to the model and thus to the configuration data.

– Reuse existing configuration data from legacy systems.

– Incorporate existing configuration data from foreign systems.

– Provide base data to support on-line exchange of data between application components.

  Data produced by application components during on-line operation is presented to operators and made available as input to other applications. Application components are also receivers of data from other application components. While this on-line data exchange is the subject of the CIS standards described in 6.3, the content of the data exchange is based on the CIM.

6.2.3 Context and time

The CIM definition for all entities implicitly allows multiple instances of objects at multiple times.

To distinguish between the CIM entities instantiated in identical application components, information exchanged between components is expected to include a context or environment. This could be implemented as an alphanumeric string, e.g., TRAINING, REALTIME, OFFLINE, STUDY, and REPLAY. However, the components themselves should not be aware of the context in which they are operating.

All data exchange events between two CIM based applications shall include a concept of time. Depending on the application, this can either be implicit (e.g., the time is the time of the receiving application) or explicit (i.e., data items have time tags, either individually or for entire blocks).

Within an application, CIM entities such as PowerSystemResource or MeasurementValue may include a start date and time, end date and time, and current date and time in an unspecified way. Different applications may hold time in different ways, e.g., a global variable, held as an attribute in individual objects, as an associated object, etc. This is considered to be a design issue and thus not in scope.

In addition, the data in an operational system will typically evolve over time. Objects will be created or deleted and their property values will be changed. Hence, each object will have a creation time and possibly a deletion time. Property values will have one or more times when they have been created or changed. The CIM does not explicitly model this except for the MeasurementValue that has an explicit time-stamp for the last updated time. The lack of explicit time-stamps for the remaining objects and property values does not prevent the use of the model in applications where time is recorded, e.g. in data warehousing applications. For such applications it is recommended that

– all objects have a “creation” time-stamp and a “deletion” time-stamp;
– all property values have a “last updated” time-stamp;
– a history is maintained for all property “last updated” time-stamps together with the corresponding values.

The data values together with associated time stamps can be accessed through APIs as specified in the IEC 61970-4XX CIS standards.

6.2.4 Consistency of relationships and attributes

The CIM represents a set of entities, relationships and attributes of an electrical network at a particular state. This document does not specify how an application should control consistency of the relationships and attributes of objects instantiated within it. For example, if a switch status is changed, it may be necessary to recalculate the topological relationships and to obtain a new set of measurement values. This may involve more than one application.

Application interfaces are responsible for providing consistency status information required to qualify information exchanges (e.g., an enumeration 'ModelValid', 'TopologyValid', 'LoadFlow Valid').

For a particular network configuration, conducting equipment can be grouped into feeders, and connectivity nodes may be grouped into topology nodes. It is part of an information exchange context to define whether the information refers to 'ASBUILT, 'CURRENT’ etc. This is equally true for measurement values and other simple attributes as it is for the topological relationships.

For example, a database application may hold instances in their ASBUILT state, and a SCADA application would hold instances in their ‘CURRENT’ state. The CIM model is identical for both applications. The instances within each application may have different values.

6.2.5 Relationship to IEC 61968 series of standards

As stated earlier in Clause 4.4, the IEC 61968 series of standards also build on the CIM, by extending the CIM Base contained in IEC 61970-301 wherever possible by extending it to include additional specializations of existing classes, but also adding entirely new sets of classes to model objects found in the distribution problem domain. So to understand the entire scope of the CIM, it is necessary to consider the CIM as described in both the IEC 61970 and IEC 61968 series of standards[10].

[10] The extended CIM UML model supporting both sets of standards is maintained by IEC TC57 WG14, which is responsible for the IEC 61968 series.

6.3 CIS (IEC 61970-4XX)

The purpose of the IEC 61970-4XX CIS documents is to specify the interfaces that a component shall use to facilitate integration with other independently developed components. Although typical applications and functions are identified in Annex B to assist in defining the types of information that shall be transferred, the purpose is not to attempt to define components *per se*. The component vendors should be free to package different collections of component interfaces into component packages while still remaining compliant with the EMS-API standards.

The CIS specifies the two major parts of a component interface:

a) The interfaces that a component (or application) should implement to be able to exchange information with other components (or applications) and/or to access publicly available data in a standard way. The component interfaces describe the specific events, methods, and properties that can be used by applications for this purpose.

b) The information content or messages that a component exchanges with other components. This is sometimes referred to as an information exchange model.

IEC 61970-4XX is organized to separately document these two parts:

a) Parts 401 to 449:

These are reserved to specify the generic services to be supported by component interfaces. These specifications describe in narrative terms with text, Unified Modelling Language (UML) notation, and IDL the interface functionality that is standardized. These specifications define the generic services that can be used by any application to exchange information with another application or for public data access.

b) Parts 450 to 499:

These are reserved for specifications that address the specific information exchange requirements for typical application categories, as defined in Annex B. These specifications define the information content of the standard information exchanges between applications. These are defined as events but may be exchanged in a variety of ways, including being published as messages, notifications followed by a request, or exchanged as XML documents in some cases. The properties and methods to be supported by each interface, if required, will also be identified. Supporting documentation includes use cases and event sequence diagrams.

Depending on the type of exchange envisioned for the application category, specific generic services may be specified to ensure interoperability between components developed by different suppliers. The intent in application of these standards is to provide as much flexibility as possible in the middleware chosen to actually accomplish the information exchanges while still ensuring interoperability.

The information content is documented in an Information Exchange Model (IEM) for each application category.

The IEC 61970-4XX CIS specifications are documented in a manner that is independent of the underlying technology used to implement them. Thus the IEC 61970-4XX series of documents are sometimes referred to as Level 1 CIS. The IEC 61970-5XX series of CIS documents described in 6.4 provide the mappings for the IEC 61970-4XX CIS specifications to specific infrastructure technologies, and are thus sometimes referred to as Level 2 CIS.

6.4 CIS technology mappings (IEC 61970-5XX)

Since the CIS documents are independent by design of the underlying infrastructure technology, they shall be mapped to specific technologies for implementation purposes. To ensure interoperability, there shall be a standard mapping for each interface to each technology. For example, if Java is the chosen implementation technology, then there needs to be a standard mapping of the publishing and event subscription services specified in the CIS document to Java services.

Similarly, the event definitions compiled in the IEM from the CIS documents for each application category need to be mapped to the specific language used for transmission of the information. For example, if a message broker is to be used to deliver XML messages, then the CIS events need to be mapped to XML.

It is anticipated that the following mappings or specializations will become companion standards in this series as these standards are deployed. Some are language-specific and some are middleware-specific:

- C++ language;
- C language;
- CORBA®;
- DCOM;
- Java®;
- XML. The XML specialization will provide interoperability between independently developed components with a GID interface when XML based messaging is used as the integration technology.

7 General expected infrastructure functionality

7.1 General

This clause describes utility inter-application infrastructure capabilities necessary to integrate distributed components. These services are provided by the Component Execution System described in the EMS-API reference model in Clause 4. The services and functionality described are independent of the underlying technology used for this infrastructure.

In the context of the following requirements, an “event” is a unit of information exchange which is issued asynchronously by its source (“push”). A “component” is a reusable module of application software on an integration bus serving as either a publisher or subscriber (receiver) of an information exchange.

The following is a list of capabilities that should be provided by the Component Execution System to have a full-featured inter-application integration capability. However, this is not meant to imply that every implementation shall support each and every function. The list is provided as a starting point in preparing a specification for an actual system deployment of these standards.

The specific requirements of importance to a deployment will depend to a large extent on the specific operational and performance needs of the interconnected systems.

A Component Execution System:

a) should allow components to exchange information of arbitrary complexity;

b) should be able to be implemented using various forms of distributed component technology (e.g., CORBA, DCOM, message brokers, message oriented middleware, relational databases, object-oriented databases, or others) (refer to Clause 6);

c) should allow publisher and/or subscriber components to be deployed by system administrators independently of other components;

d) should ensure that published information is completely re-usable in the sense that once a given type of event is published, any new authorized entity may acquire the event without having to make any changes or additions in the publisher component;

e) should provide a generic event history facility as a component. This allows all or selected information exchanges to be saved in a permanent store;

f) should support an Event History schema based on a metadata model for information exchange;

g) should provide an Event History component to record the time at which the publishing component issued each event;

h) should be capable of supporting event information model versions and component versions. (This allows a complete audit trail to be preserved which is capable of supporting rigorous reconstruction of history, if that should become a requirement.);

i) should provide an Inter-application Supervisor component that analyzes the state of any application component interface connected to the utility services. It may be enabled and disabled, and has the capability to provide performance monitoring capabilities. Those elements will help to provide statistics in order to identify bottlenecks or areas subject to improvement in the future. The information is required to help the administrators configure information exchanged among components and to ensure availability;

j) should allow a component to send or request information without knowing where the receiving component is physically located or if it is currently connected. The receiver may be unreachable because of a network problem, or be naturally disconnected as in the case of mobile users who only connect periodically. Components may be unavailable because they have failed or because they only run during certain hours; when the network becomes available or the receiving application is ready to process requests, the waiting information shall be delivered.

The following subclauses provide general capabilites expected for the different parts of the EMS-API component execution system.

7.2 Component Container

Component Containers provide a set of APIs for components to invoke the services supported by the containers. The following services are generally provided as part of commercially-available off-the-shelf Component Containers. The list of such common distributed computing services needed to support the EMS-API are:

a) Component Life Cycle Service: this service allows the starting, stopping, and control of components to be executed.

b) Naming Service: this service provides a component naming service that supports a hierarchical structure and allows a component to locate other components using a human readable name. The naming service supports the use of existing utility names, as well as the creation, removal and aliasing of names.

c) Time Service: this service provides a way for distributed components to all have the same time with a configurable accuracy.

d) Concurrency Control Service: this service facilitates the management of shared, similar items that are distributed, for example when multiple components take care of different parts (exchanges, exchange types) of the same business object in real life.

e) Security Service: this service allows an application to set and verify the privilege level of components and users with which exchanges are being performed, as well as encryption and decryption of individual exchanges. This service also supplies host authentication, i.e., authentication of node(s).

f) Transactional Service: this service allows an application to declare the beginning and end of a multi-step transaction that either succeeds or fails as an atomic unit.

g) Component Interaction Service: this service provides reliable message transfer with a selectable Quality of Service. It also provides life-cycle management of interaction services (create, delete, copy, and move) and querying of established interaction (mainly valid for Publish and Subscribe interactions).

h) Publish and Subscribe Messaging Service: this service provides synchronous and asynchronous message transfer between decoupled (anonymous) component instances.

i) Request/Reply Messaging Service: this service provides reliable synchronous message transfer between coupled, identified component instances.

j) Publish/Reply Messaging Service: this service provides decoupled initiation of a message transfer (Publish), which is then finished by a coupled transfer (Reply).

k) Workflow Service: this service allows application level programmers to create and run business process automation agents.

7.3 Middleware

Middleware services are needed to support the component containers with fundamental services required for distributed applications independent of the use of component models. The demarcation between middleware and component containers is abstract rather than concrete.

7.4 Communication Profile Services

Integrating two components requires a connection between them. Since there is more than one kind of network, different resources speak different protocols, such as IIOP and Hyper-Text Transport Protocol (HTTP). To connect multiple components, an integration system shall reconcile network and protocol differences transparently to the components.

IEC 61970 series of standards requires that the Communication Profile services should:

a) guarantee delivery of network messages to their network destination (if active);

b) provide guaranteed delivery, ensuring that network messages are delivered exactly once, regardless of network failures or changes;

c) provide guaranteed ordering, preserving the sending sequence of the source when delivering messages, regardless of network failures or changes;

d) guarantee that if a network message cannot be delivered to a network destination, the network source will receive a message indicating the non-delivery;

e) provide a selectable quality of service for prioritization of network messages or delivery via specific network paths;

f) provide dynamic adaptation to the speed of processing network messages by the network destination to allow slow destinations to work on the services.

7.5 Utility-specific services

There are a number of utility-specific services that may be needed to support components in an EMS that are not available from commercially-available component execution systems. A list of possible services that fall in this category are listed in Annex C as examples along with alternatives for providing such services. It is expected that if any of these services are required to support information exchange between applications, they will be identified in the process of developing the IEC 61970-4xx series CIS and will become part of those standards.

Annex A
(informative)

Component models

A.1 Component model

A component model defines the basic architecture of a component, specifying the structure of its interfaces and the mechanisms by which it interacts with its container and with other components. The component model provides guidelines to create and implement components that can work together to form a larger application.

There are four primary component models with wide acceptance within the software industry today that are described in the following clauses.

A.2 Enterprise JavaBeans®[11]

The Enterprise JavaBeans (EJB) specification defines a server component model for JavaBeans® [2][12]. Enterprise JavaBeans are specialized, non-visual JavaBeans that run on a server rather than a client. An Enterprise JavaBean can be assembled with other beans to create a new application. An enterprise bean can be manipulated and customized through its property table and customization methods.

The Enterprise JavaBeans component model defines the relationship between an enterprise bean component and an enterprise bean container system. Enterprise JavaBeans do not require the use of any specific container system. Any application execution system can be adapted to support Enterprise JavaBeans by adding support for the services defined in the specification. The services define a contract between an enterprise bean and the container, thus creating a portability layer. This corresponds to the component adapter for Enterprise JavaBeans that is required for any execution system except the Enterprise JavaBeans execution system, called an Enterprise JavaBeans Server (EJB server). This permits any enterprise bean to run in any application execution system or container that supports the Enterprise JavaBeans contracts.

The EJB server provides a standard set of services to support enterprise bean components. Since Enterprise JavaBeans are transactional, the EJB server should provide access to a distributed transaction management service. It should also provide a container for the enterprise bean, called an Enterprise JavaBeans container (EJB container). The EJB container implements the management and control services for a class of Enterprise JavaBeans classes. It also provides the following services:

– life-cycle management;
– transaction control;
– persistence management;
– transparent distribution services;
– security services.

11 Enterprise JavaBeans is the trade name of a product supplied by Sun Microsystems. This information is given for the convenience of users of this International Standard and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

12 Numbers in square brackets refer to the bibliography.

A.3 CORBA components

CORBA has defined a component model comparable to the EJB specification [3]. The CORBA® component model defines:

a) A component, which acts as a provider of one or multiple CORBA interfaces. A component:
   - can provide and use one or more individual CORBA interfaces;
   - can emit or consume one or more specific events;
   - has configurable design-time properties;
   - has runtime attributes;
   - declares a factory interface that can be employed to create instances of the component.

b) A container model for introducing system services into the runtime environment of a CORBA component. The container model specifies:
   - locality constrained interfaces for a component to interact with its container;
   - a simplified version of CORBA transactions;
   - lifecycle support to optimize resource usage within a process;
   - security policies which provide authorization based on client identity and delegation as described by the CORBA security service;
   - persistent state management.

A mapping to EJB makes it possible for EJB to be supported as a CORBA component within a container that provides activation, transactions, security and persistence. To support EJBs natively within a CORBA container, a transformation between the EJB Java APIs and the Java mappings of the corresponding CORBA APIs is required. Conversely, a CORBA component that restricts itself to the functionality defined by the EJB specification and is written in Java should be deployable in an EJB container.

CORBA Containers are run-time contexts that provide interfaces to clients and bi-directional (locality constrained) interfaces to component instances that abstract system services for component implementations. Containers will provide policies and/or APIs to support transactions, security, state management and persistence. For example, transaction and security policies are defined in the component's deployment descriptor. The container uses these descriptions to make the proper calls to the CORBA transaction service and to CORBA security before invoking the requested operation on the component.

A container provides the following functions for its component:

- All component instances are created and managed at runtime by its container.
- Certain metadata (e.g., transactions, security, events, and persistence) are separated from the component's implementation to allow them to be manipulated prior to deployment.
- A component can be customized by editing its metadata.
- Client access to interfaces provided by a component are delegated by the container in which the component is deployed.
- Containers provide a standard set of system services to a component, enabling the component to be hosted by different container implementations.

A.4 Microsoft® COM/DCOM

Microsoft's Component Object Model (COM) was originally developed to support the integration of desktop applications on the Windows platform [4]. COM came out as one of the results of Microsoft's effort to improve OLE (Object Linking and Embedding) from version 1 to version 2. COM originally did not support distributed environments. The version of COM supporting distribution is called DCOM.

The problem solved by COM/DCOM is to facilitate partitioning a software system consisting of one or more classes into separate packages that can be developed and maintained independently of each other.

The component support in COM is fairly simple and consists of two major parts:

1) an interface description language called Microsoft Interface Definition Language (MIDL) and tools supporting the translation of MIDL code into C++ or Visual Basic interface descriptions;
2) a runtime environment supporting look up and activation of components.

DCOM then adds remote communication support to the runtime environment.

On top of COM/DCOM, Microsoft has added a set of standard interfaces exposing various system functions and services such as callbacks (connectable objects), persistency, a simple naming service (monikers), type library and dynamic interface invocation (OLE Automation), etc. These interfaces, their corresponding runtime support and COM/DCOM are what makes OLE 2.0.

COM is now being extended to a complete component execution system called COM+. COM+ supports simplified component development, simplified component installation, transactioning, events, publish and subscribe, and message queuing.

With COM+, Microsoft defines logical and physical packaging. Logical packaging means splitting up the classes and interfaces into separate namespaces, thus reducing human-readable class name collisions. Physical packaging is the unit of deployment corresponding to the actual component or module, typically one or more executables (.EXE) or dynamically linked load modules (.DLL). In the case where a DLL implements classes (in contrast to pure proxy DLLs), the process loading can be regarded as a container, especially if the loading process only contains basic or framework functionality. A component or module typically implements one or more classes as well as interfaces. There are no restrictions as to how classes or interfaces are implemented by components or modules. This is the choice of the implementer.

COM/DCOM is also available on other computer platforms like Digital Unix and Solaris. The low level support for COM/DCOM is fairly portable. Functionality associated with the Windows desktop (e.g., the registry) is less portable.

A.5 XML Web Services and Microsoft .NET

XML Web Services provide an Internet-based integration model for any-to-any integration. XML Web services allow applications to communicate and share data over the Internet, regardless of operating system or programming language. They are like components.

XML Web Services provide an Internet-based integration model for any-to-any integration. The main services are:

- UDDI (Universal Description and Discovery Information) – Publish and find other services.
- XML and HTTP – Communications and data content.
- SOAP (Simple Object Access Protocol) – Transaction-based service for exchanging information and invoking services across distributed applications.

.NET is Microsoft's strategy and offerings to embrace XML Web Services everywhere, and is essentially a platform for XML Web Services. There are five key areas that need to be addressed to create an XML Web Services platform:

a) Clients – all types of PCs and smart devices that are XML-aware.

b) Services – such as the Microsoft .NET Passport Authentication service that offers single sign-on capability for any Web site.

c) Servers – Integrated suite for running, managing, orchestrating Web services and applications that can Store, route, transform, and bridge to legacy data.

d) Developer tools – such as Visual Studio .NET and the .NET Framework.

e) Experiences and solutions – combine the best of local applications and Internet services.

.NET is Microsoft’s software platform that addresses all of these areas.

The Microsoft .NET® Platform includes a comprehensive family of products, built on XML and Internet industry standards, that provide for each aspect of developing, managing, using, and experiencing XML Web Services.

The .NET Framework is a high-productivity, standards-based, multi-language application execution environment that provides essential infrastructure services and eases deployment. It provides an application execution environment that manages memory, addresses versioning issues, and improves the reliability, scalability, and security of your application. The .NET Framework consists of several parts, including the Common Language Runtime, a rich set of class libraries for building XML Web Services, and ASP .NET.

Annex B
(informative)

Typical applications and functions

Table B.1 – Typical applications and functions

Application category			Actor (operator)
	Application name	**Comment/explanation**	
SCADA			Field operator
	Time synchronization	Synchronize RTU time tags	
	RTU and PLC communication	Field data acquisition using RP570, IEC 60870-5-101, …	Substation operator
	Remote center communication	Inter -site communications using ICCP, ELCOM, …	
	Signal processing	Filtering, scaling, engineering unit conversion, quality checking, limit checking	
	Data processing	Generalized calculations, status, analog, and accumulator value, quality processing	
	Tagging	Tagging is used to attach tags at arbitrary real world objects represented in a SCADA	
	Single line diagrams	Page based graphical process displays	
	Status lists	SCADA/EMS object status reports	
	Supervisory control	Device and sequence control (interlocked and non-interlocked)	
	System control		
	Intersite control		
	Limit management	Dynamic conditional selection on limit sets	
	Time tagged data processing	Collection, storage and presentation of time tagged data in tabular/trend displays	
	Disturbance data collection	Sequence of event recording and Post mortem review	
	Equipment statistics	Accumulation of equipment operations	
	Switching schedules		
Alarm Processing		Alarm and event processing	
Topology processing		Analyzes equipment connectivity and energized status and builds bus model to support network functions and for connectivity presentation on user displays. May include ground connectivity analysis for fault analysis.	Transmission operator, control area operator