Technical Overview

Agilent OpenLab

Agilent Instrument Control Framework (Agilent ICF)

An open-system approach to simplify control of Agilent Instruments with a non-Agilent chromatography data system.

Introduction

This Technical Overview describes the Agilent Instrument Control Framework (ICF) software that enables control of Agilent liquid chromatography (LC) and gas chromatography (GC) instruments with non-Agilent chromatography data systems (CDS) or other software solutions. The overview explains the technical concepts that underlie ICF and RapidControl.NET (RC.NET) instrument drivers and how the software components interact in an open-systems environment.

In today's competitive environment, many labs must reduce costs, limit training time, and simplify regulatory compliance. The implementation of a single vendor CDS is one way to accomplish these goals. If a lab is standardized on a non-Agilent CDS, it can still use world-class Agilent instruments. To achieve this, the CDS vendor can take advantage of ICF, which simplifies the development effort needed to control Agilent LC and GC instruments with a non-Agilent system.

Agilent

Trusted Answers

ICF advantages

ICF provides users with the following advantages:

- A common look and feel across multiple CDS
- Support for more than 150 different Agilent LC, CE, GC and Headspace modules including all their options and features
- Regular maintenance updates with new features and support for additional instrumentation
- Comprehensive user interface that seamlessly incorporates all instrument features
- Combined system view that provides overall status, instead of module-specific views

ICF reflects Agilent's commitment to open systems. Before ICF was introduced, software developers used low-level instrument control codes to write native-mode drivers for Agilent instruments. Agilent ICF minimizes driver development effort. ICF includes the necessary instrument drivers and user interfaces (Figure 1), and provides a simple programming interface for software connectivity. For full control of current and future Agilent instruments, the software vendor simply develops an adapter from its own software to ICF, with no need to write more instrument control code.

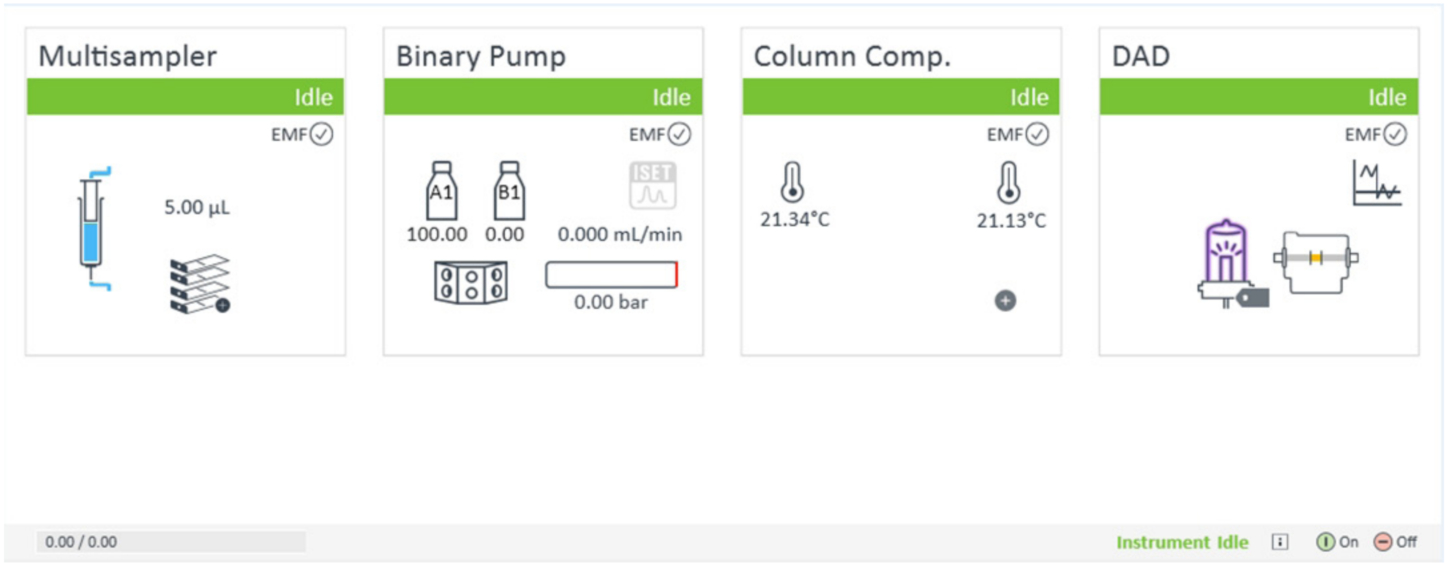

Figure 1. Instrument control user interface for an Agilent 1290 Infinity LC System.

Instrument Control

Overview

A typical analytical instrument control program is characterized by its:

- Overall architecture
- Functional blocks
- Deployment strategies

The architecture of an analytical application is defined by the abstraction layers, which are used to implement instrument control for either modular or monolithic instruments (Figure 2). Table 1 provides definitions for the terms used in Figure 2.

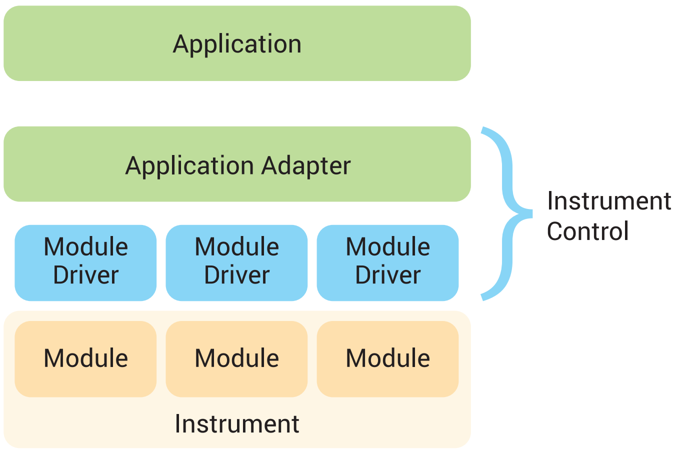

Figure 2. Overview of instrument control.

Table 1. Definition of instrument control terms.

Application	The core analytical application. Typically involves control of automation such as the execution of sequences.
Application Adapter	Software adapter, which translates the instrument drivers to the application interface.
Module Drivers	Software layer, which encapsulates module-control-specific functionality, such as communication protocol handling.
Instrument	Analytical hardware. Can be a monolithic instrument with hardware options or a modular instrument, built by combining multiple hardware modules.

Functional blocks

From a functional perspective, instrument control is typically comprised of the functional blocks listed in Table 2. See Appendix A for more information about ICF functional blocks.

Table 2. Functional blocks.

Instrument configuration	The description of the physical instrument with regards to options and setup.
Instrument method handling	Defines the handling of a set of parameters relevant to an analytical run.
Instrument status representation	Reflects the status of an instrument.
Instrument run control	Handles both physical communication and control of an analytical run, including the acquisition of the measured data from the instrument.

Deployment models

The deployment strategy of an analytical application may vary from a single node (workstation) to a distributed (client/server) deployment (Figure 3a and b). See Appendix B for more information about ICF deployment models.

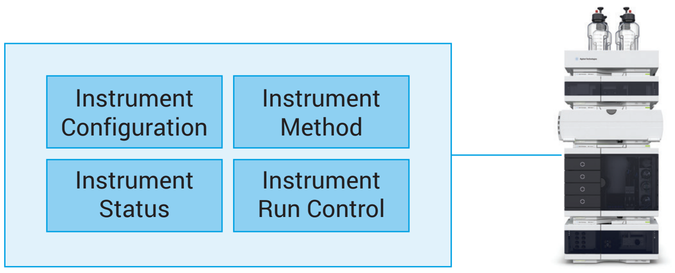

Figure 3a. Example of a typical workstation deployment. All functional blocks reside on a single machine, including instrument run control.

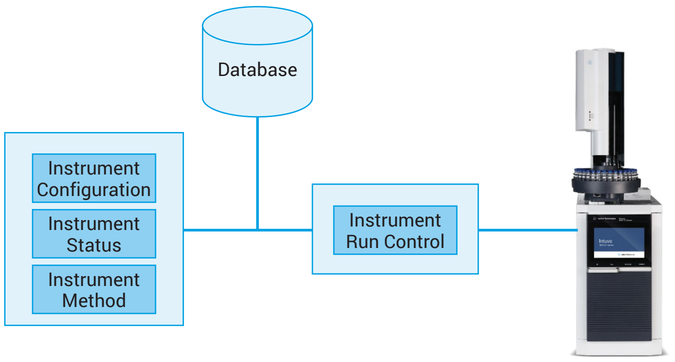

Figure 3b: Example of a client/server deployment. All functional blocks—except the instrument run control—are distributed on client machines. In this example, the instrument run control resides on a distinct machine usually close to the instrument, which acts as the acquisition controller. Instrument-related data, such as the instrument configuration or acquisition method, may be stored remotely.

ICF and RC.NET instrument control

Agilent has developed two related standards, RC.NET and ICF, to address the requirements of fully integrated instrument control. RC.NET defines a standard for the implementation of hardware drivers on a module level, whereas ICF defines an interface for the integration of instrument control into an application on an instrument level. In other words, ICF aggregates multiple RC.NET drivers to produce a combined instrument view.

Overall architecture

When the RC.NET and ICF standards are included, the general abstraction layers are as shown in Figure 4. While the application and instrument layers remain the same, the instrument driver and the application adapter can be modeled with finer granularity if necessary (Figure 4). To control the instrument modules independent of the host CDS or workstation architecture, the instrument-driver layer is substituted with the RC.NET drivers. See Appendix C for more information about ICF abstraction layers.

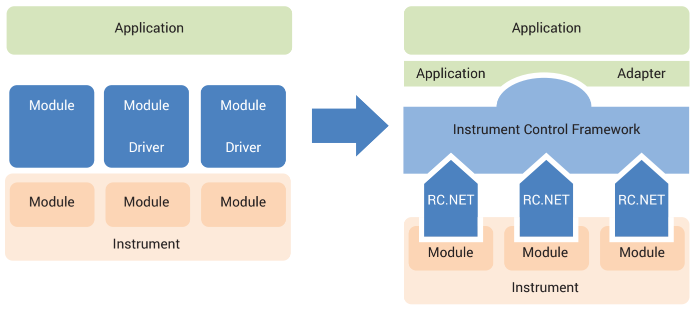

Figure 4. Instrument control abstraction layers when using ICF and RC.NET.

The instrument drivers are supplied by Agilent and are specific to the respective modules. See Appendix D for more information about ICF driver packages. The application adapter is required to transparently integrate the functions of ICF into the host CDS. The adapter is not complex, as it only integrates ICF, which provides the integrated functionality needed for full instrument control. **Note:** ICF is not the tool to use to change or extend capabilities on a module level. See Appendix E for more information about ICF cooperation with non-RC.NET modules.

Benefits

- RC.NET facilitates implementation of drivers for hardware modules independent of the application
- ICF builds layers on the RC.NET drivers, which aggregate multiple RC.NET drivers, resulting in a combined instrument view
- Because multiple drivers can be aggregated, and some of the functionality and synchronization tasks are included in the ICF layer, the application adapter layer is lightweight

ICF and RC.NET architecture

RC.NET and ICF are designed to operate in a wide array of applications, independent of deployment and application infrastructure. In addition to being independent building blocks in the hosting application, RC.NET and ICF share the following qualities and considerations:

- Communication between different components is based on XML exchange between the components via the underlying application infrastructure.
- User interface components are separated from functional components, providing the maximum flexibility for the hosting application usage scenarios.
- Data containers (configuration, method, pretreatment, status) are encapsulated and handled as black boxes.
- Data containers are modeled as XML fragments to support the communication paradigms of both standards. **NOTE: the structure and content of these XML fragments must not be used by the host application.**
- The structure of data containers is the responsibility of the hosting application, to ensure that the components are agnostic to storage type and structure.
- The host application is decoupled from driver-specific settings. This removes the need to know and use driver-specific commands on a programmatic level, including driver version dependencies.
- Data migration interfaces provide capabilities to transfer contents of the data containers from older to newer versions and between different configurations.
- Data access interfaces provide generic data access to the data containers, using a reflection-like mechanism if necessary. As a result, the data structures can be accessed dynamically; for example, queried and modified as required.
- Run control is decoupled from the sequence execution and data analysis functionality.
- The scope of run control is single sample analysis, but there are provisions for overlapping injections, dual simultaneous injection, or headspace operation. Further automation, such as sequence control and execution, must be handled by the application, resulting in the possibility of component use in different automation scenarios.
- Data from sample acquisitions are returned transparently to the host application in an unprocessed format, delivered by the instrument. Further data manipulation and presentation steps are completely separated from the interfaces and are within the scope of the application layer.

In addition, ICF provides an instrument-centric design. The components and the data handled are always presented at the instrument level as opposed to the single-module level. At the instrument level means that any module controlled by an RC.NET driver is part of the ICF, which provides a plug-in architecture for RC.NET drivers. As a result, new modules can be easily deployed without more implementation effort. Once installed, a new module is automatically available for the application. The customer only has to install the new RC.NET driver to use the new module within their application.

Conclusion

ICF and RC.NET are complementary instrument control components.

- ICF requires an RC.NET driver to support a defined hardware set
- ICF introduces an abstraction layer to provide a combined view of all RC.NET drivers that link together to represent a complete instrument
- RC.NET is designed to control a single module. RC.NET allows the development of drivers to support modules independently from the application.

RC.NET provides a means for implementing application-independent instrument hardware support. ICF provides a technique for providing generic support for analytical instruments within a specific user application.

ICF provides connection to non-Agilent CDSs, which in turn enables transparent connection of Agilent instrument drivers. From an instrument control perspective, ICF provides functionality that is identical to that of Agilent CDS.

Together these components standardize how the CDS interacts with analytical instruments, enabling multivendor instrument control using a non-Agilent CDS.

Appendixes

Appendix A.
ICF functional blocks

ICF provides components that cover typical data system capabilities (Figure 5.):

Method component: Method-handling functions (audit trail creation for two methods, reporting of method)

Method UI component: UI for editing method parameters (method parameter, timetable), including help and tool tips

Control component: Interaction with the hardware and run control functions (downloading methods, start/stop run, gathering chromatography data)

Configuration component: Configuration handling functionality (audit trail creation for two configurations, reporting of configuration)

Configuration UI component: UI for editing the configuration, including help and tool tips

Status UI component (optional): UI showing the overall status of instrument hardware, including direct interaction for dedicated features (switch on/off UV lamp)

Pretreatment component (optional): Pretreatment handling parameters, if supported by the hardware. Pretreatment defines customized sample handling steps for samplers. In some applications, this feature is also known as the injector program.

Pretreatment UI component (optional): UI for editing the pretreatment parameters

In addition to the components above, ICF supports for extended-use cases, such as actions, or well plate editing.

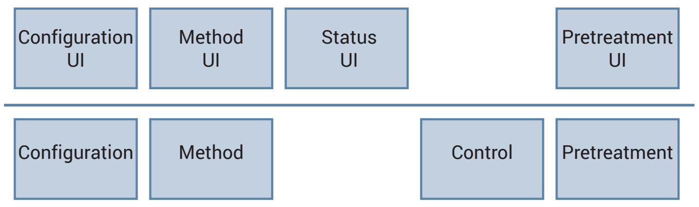

Figure 5. ICF functional blocks.

Appendix B.
ICF deployment models

ICF defines the components as independent building blocks that can be used and wired together by an application with application-dependent infrastructure and deployment strategy (for example, workstation, or client/server environment). Communication occurs by passing XML fragments between the different components. There are no other direct linkages between components, such as DCOM or RPC. Therefore, components are compatible with different deployment strategies. Communication uses a separate acquisition controller as illustrated in Figure 6.

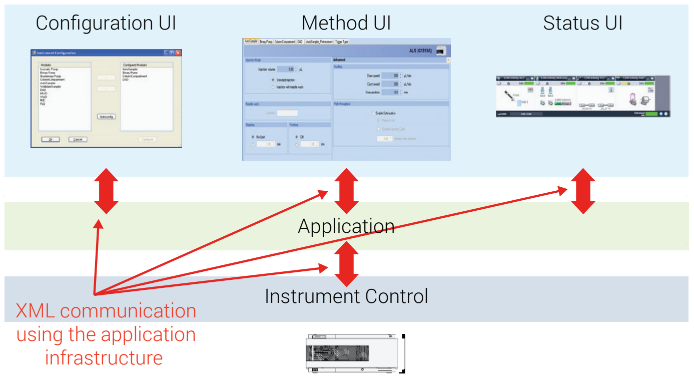

Figure 6. Communication using separate acquisition controller.

Appendix C.
ICF abstraction layers

The different levels of communication protocols between the layers show increasing simplification of the protocol with a modular instrument (Figure 7).

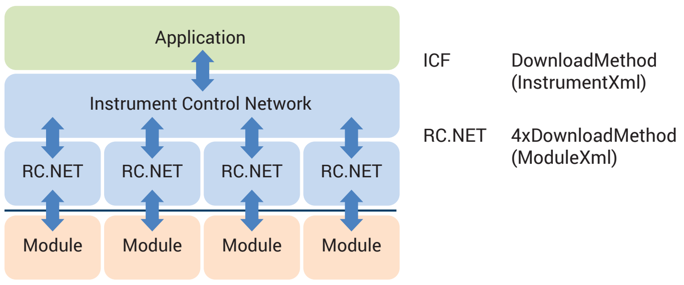
Application
Instrument Control Network
RC.NET
RC.NET
RC.NET
RC.NET
Module
Module
Module
Module
ICF
DownloadMethod (InstrumentXml)
RC.NET
4xDownloadMethod (ModuleXml)

Figure 7. ICF abstraction layers.

Appendix D.

ICF driver packages

The deployment strategy for the ICF is based on two parts: framework installation and driver package installation.

Framework installation installs a version of the ICF components within the context of the host application. Installing the framework allows an application to use ICF driver packages. The framework can be installed without user interaction as part of the host application.

Driver package installation installs driver packages. A driver package defines the available modules that can be used if that package has been chosen during instrument configuration.

During framework startup, the framework scans the system for installed packages, which are available for use within ICF.

Appendix E.

ICF cooperation with non-RC.NET modules

Although ICF provides the system view for RC.NET-capable modules, it is possible to run "mixed" instruments containing modules without RC.NET drivers. In this case, the host application must synchronize the ICF instrument with the participating non-RC.NET modules. In this scenario, the ICF layer acts as a "complex" module driver (Figure 8).

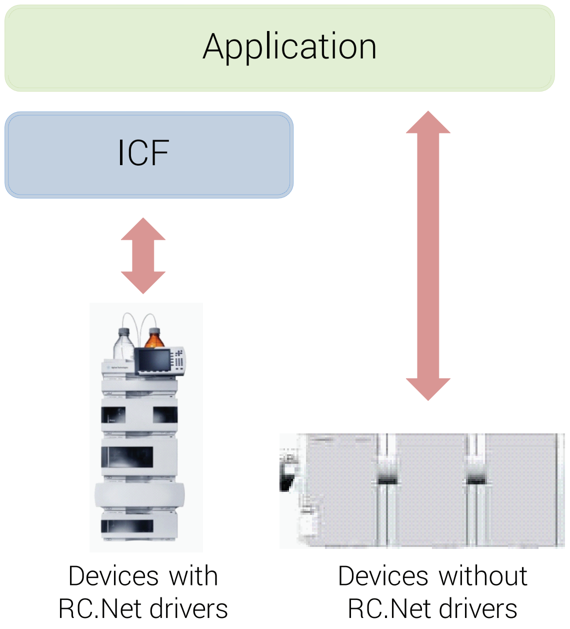

Figure 8. Mixed instrument environment with RC.NET capable and non-RC. NET drivers.

To learn more about the Agilent Instrument Control Framework, visit **https://www.agilent.com/en/products/software-informatics/partner-program-for-openlab-software/instrument-control-program** www.agilent.com

Agilent
Trusted Answers