



KUKA KRC4 & KRC5 SDK

User & Integration Manual

Controller access, monitoring, files, backups and program control for KRC4 and KRC5

Detailed API guidance with C# and VB.NET examples

RAS Otomasyon - KUKA KRC4 & KRC5 SDK v1.0

October 2026 | English edition

Delivered runtime: .NET Framework 4.8

.NET 8: documentation-only integration profile

www.rasotomasyon.com

info@rasotomasyon.com | +90 224 334 09 81

KUKA is a registered trademark of its respective owner.

Table of Contents

Chapter 1 Introduction	4
Chapter 2 System and Network Requirements	6
Chapter 3 Installation and First Read	8
Chapter 4 Sessions and Controller Identity	11
Chapter 5 Controller Discovery	15
Chapter 6 Reading and Writing Variables	19
Chapter 7 Variable Monitoring	25
Chapter 8 Files and Directory Transfers	29
Chapter 9 Backups and Active Project Download	40
Chapter 10 Robot Program Operations	43
Chapter 11 Runtime Messages	51
Chapter 12 Events and Callback Handling	55
Chapter 13 Windows Forms Integration	57
Chapter 14 Errors and Troubleshooting	60
Chapter 15 .NET 8 Integration Profile	64
Chapter 16 Complete Application Workflows	67
Chapter 17 Deployment and Commissioning	71
Chapter 18 KRC5 Controllers	72
Appendix A Data Model Reference	75
Appendix B API Index and Example Conventions	78

Appendix C Technical Platform References

81

Chapter 1

Introduction

1.1 Who this manual is for

This manual is for automation engineers commissioning a PC connection to a KUKA KRC4 or KRC5 controller and software developers integrating that connection into a C# or VB.NET application. It explains what each application-facing operation does, what data it needs, how to handle its result, and how to combine operations into a dependable workflow.

No prior knowledge of this SDK is required. Readers should be familiar with their controller's file organization, operating modes and local commissioning procedures. Program commands should be integrated only by personnel responsible for the cell's authorized operating workflow.

1.2 What the SDK provides

The RAS KUKA KRC4 & KRC5 SDK is a Windows client library. Applications use the `KRCSDKServerLib` namespace and the KRC session class to discover controllers, read and write variables, monitor changes, transfer files and directory trees, create backups, download the active project, operate the robot interpreter and read runtime messages.

Area	Application capabilities
Controllers	Network discovery, known-address discovery, metadata snapshots and direct target descriptors
Variables	Synchronous read, Task-based read, event-based read, operating-mode read and controlled write
Monitoring	Per-controller watch lists, batch polling and change notifications
Files	Existence checks, listings, directory creation, upload, download and deletion
Backups	Controller archive ZIP and active WorkVisual project download
Programs	Interpreter state/mode, selection, start, stop, reset, deselection and line selection
Messages	Runtime-message snapshots and collection-change notifications

1.3 How an application uses the SDK

A typical application creates a session, selects a `RobotList` descriptor, subscribes to the events it needs, performs operations and disposes the session when the workflow ends. A controller descriptor can come from discovery or from a known address entered by an operator.

The SDK has no application-facing `Connect`, `Disconnect` or `IsConnected` method. Creating a session does not prove network connectivity. The first actual read or service operation verifies that the required controller service can be reached. `CheckScanning` describes discovery activity; it is not a connection-health flag.



1.4 Scope and operating boundaries

This manual covers the KRC application API, KRCLoader, public data models and their events. The `RAS.Contracts` and `RAS.OnLineServicesFacade` namespaces contain protocol and service infrastructure; applications following this manual do not need to construct those types for routine workflows.

The SDK is not a hard real-time controller and does not replace cell safety functions, emergency-stop circuits or controller authorization. `StopProgram` is a program-stop request, not an emergency stop. Program selection, upload, interpreter control and operating-mode selection are distinct operations.

Note: Example IP addresses, file paths and job variables are illustrative. Substitute the commissioned controller address and approved paths. Variables such as `MY_JOB_ID` must already exist and be accessible on the controller.

1.5 Reading this manual

Chapters 2-5 establish a working session. Chapters 6-11 describe every operation with parameters, return behavior and C#/VB.NET examples. Chapter 12 covers events; Chapters 13-14 cover desktop integration and troubleshooting. Chapter 15 documents the .NET 8 integration profile without claiming an implemented .NET 8 SDK build. Chapter 16 supplies complete application workflows, and Chapter 18 covers KRC5 controllers.

Chapter 2

System and Network Requirements

2.1 Runtime requirements

The delivered client library and the supplied VB.NET Windows Forms application target .NET Framework 4.8. Use a Windows installation on which that runtime is available. To build from source, install a compatible .NET SDK or Visual Studio build tools and the .NET Framework 4.8 targeting pack.

Profile	Status in this delivery	Requirements
C# / VB.NET application targeting net48	Implemented client profile	Windows and .NET Framework 4.8
Supplied VB.NET Windows Forms example	Implemented sample application	Windows, .NET Framework 4.8 and the SDK DLL
.NET 8 application profile	Documentation and integration templates only	A separately supplied, validated .NET 8-compatible SDK or integration component is required before deployment

No external controller-vendor DLL or NuGet package is required by the delivered .NET Framework client. Standard Windows/.NET Framework services, including WCF and XML serialization, remain runtime dependencies. The client library is not a self-contained Windows executable.

The source project is not restricted to an x86-only target. Select Any CPU or an explicit architecture appropriate to the rest of the application and its deployed components. Keep application and dependency architecture choices consistent.

2.2 Controller prerequisites

The controller (KRC4 or KRC5) must expose the services used by the requested workflow. IP reachability alone does not guarantee that variable access, file handling, runtime management and project services are all available. Obtain the controller address, its service configuration and the access policy from the commissioning administrator. For KRC5 controllers, see Chapter 18.

The SDK does not install controller services, declare KRL variables or enable controller options. A generic TCP variable-proxy or Ethernet KRL endpoint is not interchangeable with the services documented here.

2.3 Network service checklist

Service group	Default TCP port	Typical operations
FileHandling / Archive / Deployment	49010	Files, directories, ZIP backup and active-project download
DataAccess	49002	Variable reads, writes and monitoring
DeviceInfo	49003	Device information and active-project metadata
RuntimeManager / RuntimeMessageWindow	49004	Interpreter operations and runtime messages

These are the default endpoint settings. A commissioned installation can use different configured ports or endpoint names. Allow the required outbound client traffic and the controller replies. Discovery also uses its discovery exchange; do not assume that allowing TCP ports alone enables automatic discovery across every routed network.

Use an address supplied by the commissioning administrator. Prefer known-address discovery or a direct descriptor when broadcast discovery is not available. If a device appears in discovery but a later operation fails, test the particular service needed by that operation.

2.4 Timeouts and responsiveness

The service configuration uses a 10-second connection-open setting and longer send/receive budgets, commonly 10 minutes. File and backup operations can legitimately run much longer than a status read. The high-level API does not expose a general timeout setter or a cancellation token.

Run synchronous service calls on a worker when building a UI. A timeout implemented in the UI changes how long the UI waits; it does not cancel an already issued remote operation. Do not retry a write, deletion or program-start request blindly after a timeout, because the controller may already have acted.

Chapter 3

Installation and First Read

3.1 Add the client to a project

When building the SDK and the application together, add a project reference to `src/KRC4Sdk/KRC4Sdk.csproj`. When deploying a prebuilt application, reference `KRCSDKServerLib.dll` and copy it beside the application's executable. Do not add both the project reference and a second DLL reference for the same SDK assembly.

C# project example:

Project / configuration XML

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net48</TargetFramework>
    <LangVersion>latest</LangVersion>
  </PropertyGroup>
  <ItemGroup>
    <ProjectReference Include="..\src\KRC4Sdk\KRC4Sdk.csproj" />
  </ItemGroup>
</Project>
```

VB.NET project example:

Project / configuration XML

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net48</TargetFramework>
    <OptionStrict>On</OptionStrict>
    <OptionExplicit>On</OptionExplicit>
  </PropertyGroup>
  <ItemGroup>
    <ProjectReference Include="..\src\KRC4Sdk\KRC4Sdk.csproj" />
  </ItemGroup>
</Project>
```

Adjust the relative reference path to match the application folder. For Windows Forms, add `UseWindowsForms` set to `true` and `OutputType` set to `winExe`. The supplied sample solution already includes the required configuration.

3.2 Namespaces used by the examples

All operation examples assume an initialized sdk and a selected robot. The API names are the same when consumed from C# and VB.NET; language keywords, array notation and optional-argument syntax differ.

C#

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Threading.Tasks;
using KRCSDKServerLib;
```

VB.NET

```
Imports System
Imports System.Collections.Generic
Imports System.Globalization
Imports System.Linq
Imports System.Threading.Tasks
Imports KRCSDKServerLib
```

3.3 Read a known controller address

Use this small read-only check before enabling file modification or program operations. An empty ID is suitable when only the commissioned address is known; use the discovered descriptor and its reported ID when available.

C#

```
using (var sdk = new KRC())
{
    var robot = new RobotList
    {
        BaseAddress = "192.168.0.10",
        ID = string.Empty,
        Name = "Cell A"
    };
    string mode = sdk.GetVariable(robot, "$MODE_OP");
    Console.WriteLine(mode);
}
```

VB.NET

```
Using sdk As New KRC()
    Dim robot As New RobotList With {
        .BaseAddress = "192.168.0.10",
        .ID = String.Empty,
        .Name = "Cell A"
    }
    Dim mode As String = sdk.GetVariable(robot, "$MODE_OP")
    Console.WriteLine(mode)
End Using
```

Expected values include #AUT, #EX, #T1 and #T2. If the call fails, resolve the address, service availability and permissions before extending the application. A successful read confirms the variable-access workflow; it does not independently validate file, backup or interpreter services.

3.4 Deployment files

Application	Files to keep together
Your compiled .NET Framework application	Application executable, its configuration file and KRCSDKServerLib.dll
Supplied VB.NET sample	RAS.KRC4.WinFormsVB.exe, RAS.KRC4.WinFormsVB.exe.config and KRCSDKServerLib.dll

Debug symbols are optional for execution but useful for diagnosis. Controller files and backups are data outputs, not application dependencies. Keep configuration and output directories separate from the installation directory

when your deployment policy requires it.

Chapter 4

Sessions and Controller Identity

4.1 Session lifetime

Create a session for a deliberate application lifetime. A short `using / Using` scope works for one read-only operation. A desktop form usually owns one longer-lived session and disposes it when its active operation has finished.

Do not create and immediately dispose the session inside a method that merely starts monitoring or enables message events. Those features need the owning session to remain alive. Subscribe before starting the feature so that the initial notification is not missed.

4.2 RobotList identity

`BaseAddress` is a host or IP address, without a service URL, port suffix or controller file path. `ID` is controller identity metadata. `Index` is a discovery snapshot position and also the label captured by a variable watcher when it is created; it is not the transport address.

Field	Recommended application use
<code>BaseAddress</code>	Target communication host and visible operator confirmation
<code>ID</code>	Preserve the reported identity when discovery supplies it
<code>Name / HostName</code>	Display a friendly name beside the address
<code>Index</code>	Display the discovery or watcher index; do not persist it as the only identity
Remaining metadata	Display controller, firmware, cell and project information when provided

Store the selected descriptor and its address/ID. A later discovery snapshot may reorder indexes. If mapping `VariableStatusChanged.RobotIndex` to a controller, record the mapping when configuring that watcher; do not overwrite it solely because another scan returns a different order.

4.3 One session or several sessions

One KRC session can own multiple controller watchers. Use separate sessions when message events, asynchronous read notifications or parameterless transfer-completion events need an unambiguous controller/workflow context. Those event payloads do not all carry a robot identifier. A Task returned by `ReadVariableAsync` is directly tied to its caller's target and is useful for request-specific correlation.

4.4 KRC()

Declaration

```
KRC( )
```

Creates an application session. Construct one session for the controller workflow or group of workflows that your application will manage.

Parameters

No parameters.

Result and behavior

A KRC object implementing IDisposable.

Construction does not discover controllers or issue a robot program command. A session owns its discovery watcher, program connections, message subscriptions and variable watchers. Keep it alive while those features are in use.

C# and VB.NET usage

C#

```
using (var session = new KRC())
{
    RobotList[] controllers = session.GetControllers();
    Console.WriteLine(controllers.Length); // Initially zero.
}
```

VB.NET

```
Using session As New KRC()
    Dim controllers As RobotList() = session.GetControllers()
    Console.WriteLine(controllers.Length) ' Initially zero.
End Using
```

Failure handling

Do not retain the session after its Using/using scope has ended.

4.5 KRC(Boolean)

Declaration

```
KRC(bool Loader)
```

Creates a session using the Boolean constructor overload.

Parameters

Parameter	Type	Meaning
Loader	Boolean	A reserved constructor flag. It does not enable a different connection, discovery or operating mode.

Result and behavior

A KRC object implementing IDisposable.

The Boolean overload initializes the same session behavior as the parameterless constructor. Use KRC() in normal application code. Supplying True does not connect to a controller and does not start a program.

C# and VB.NET usage

C#

```
using (var session = new KRC(true))
{
    Console.WriteLine(session.CheckScanning());
}
```

VB.NET

```
Using session As New KRC(True)
    Console.WriteLine(session.CheckScanning())
End Using
```

Failure handling

Dispose this session in the same way as a parameterless session.

4.6 KRCLoader.LoadLibrary

Declaration

```
void KRCLoader.LoadLibrary()
```

Provides an optional initialization entry point before creating the application session.

Parameters

No parameters.

Result and behavior

No return value.

This method creates and closes a temporary KRC session. It does not scan the network, read a variable, load a robot program or select a target. The application still creates its own KRC object afterwards.

C# and VB.NET usage

C#

```
var loader = new KRCLoader();
loader.LoadLibrary();
using (var session = new KRC())
{
    Console.WriteLine("SDK session ready.");
}
```

VB.NET

```
Dim loader As New KRCLoader()
loader.LoadLibrary()
Using session As New KRC()
    Console.WriteLine("SDK session ready.")
End Using
```

Failure handling

The method is optional; routine application sessions can be created directly.

4.7 Dispose

Declaration

```
void Dispose()
```

Closes the session and releases its owned discovery, message, runtime and monitoring resources.

Parameters

No parameters.

Result and behavior

No return value.

Dispose is idempotent: calling it again does not repeat cleanup. The session cannot be used after disposal. All owned resources are attempted even if one close operation fails; cleanup failures can be reported together in an `AggregateException`. Do not close a session while another user operation is still running.

C# and VB.NET usage

C#

```
var session = new KRC();
try
{
    Console.WriteLine(session.GetControllers().Length);
}
finally
{
    session.Dispose();
}
```

VB.NET

```
Dim session As New KRC()
Try
    Console.WriteLine(session.GetControllers().Length)
Finally
    session.Dispose()
End Try
```

Failure handling

Subsequent session operations throw `ObjectDisposedException`. Cleanup may report `AggregateException`.

Chapter 5

Controller Discovery

5.1 Discovery workflow

Subscribe to `ControllersAvailableEvent`, start a scan, wait for the completion notification and obtain the array with `GetControllers`. Check its length before selecting an item. A discovery-completion notification can legitimately correspond to an empty result.

The probe setting is in seconds. A fresh watcher can use twice the probe duration for the initial exchange, and device-information queries may take additional time. For commissioning, allow a practical completion budget such as 30 seconds for a five-second probe setting, rather than assuming exactly five seconds.

The session also reports device removal using `DevicesChangedEvent` and refreshes the availability notification. Update your UI list from the latest snapshot; do not preserve a disconnected discovery entry as though it were still discovered. Manually entered targets can be maintained separately by the application.

5.2 Direct targeting versus discovery

A direct `RobotList` descriptor avoids the discovery step, but it does not bypass the required controller services. A hostname must resolve on the PC; an IP address must route to the intended controller. Use `BaseAddress`, not an HTTP or `net.tcp` URL, because the SDK builds its service endpoints from the descriptor and service configuration.

5.3 ScanControllers()

Declaration

```
void ScanControllers()
```

Starts network discovery using the default scan configuration.

Parameters

No parameters.

Result and behavior

Returns after starting discovery; it does not return the controller list.

Subscribe to `ControllersAvailableEvent` before calling this method, then obtain a snapshot with `GetControllers()`. The default probe setting is five seconds. The initial discovery pass can use a longer window, and device-information queries add time. Do not assume that the list is complete as soon as `ScanControllers` returns.

C# and VB.NET usage

C#

```
sdk.ControllersAvailableEvent += () =>
{
    foreach (RobotList item in sdk.GetControllers())
        Console.WriteLine(item.BaseAddress);
};
sdk.ScanControllers();
```

VB.NET

```
AddHandler sdk.ControllersAvailableEvent, Sub()
    For Each item As RobotList In sdk.GetControllers()
        Console.WriteLine(item.BaseAddress)
    Next
End Sub
sdk.ScanControllers()
```

Failure handling

An empty completed scan is a valid result. Check network reachability and discovery traffic if the expected controller is absent.

5.4 ScanControllers(addresses, scanSeconds)

Declaration

```
void ScanControllers(string[] addresses, int scanSeconds = 5)
```

Starts discovery with known controller addresses and a specified probe duration.

Parameters

Parameter	Type	Meaning
addresses	String[]	Known controller hosts or IP addresses. Null uses the normal discovery configuration.
scanSeconds	Int32	Positive probe-duration setting, in seconds. Default: 5.

Result and behavior

No return value. Discovery completion is reported by an event.

A new scan replaces the session discovery watcher. Previously returned RobotList descriptors remain separate objects; select the required descriptor from the updated snapshot. Supplying known addresses is useful when commissioning a defined set of controllers; it is not a guarantee that every supplied address will respond.

C# and VB.NET usage

C#

```
string[] addresses = { "192.168.0.10", "192.168.0.11" };
sdk.ScanControllers(addresses, scanSeconds: 5);
```

VB.NET

```
Dim addresses As String() = {"192.168.0.10", "192.168.0.11"}
sdk.ScanControllers(addresses, scanSeconds:=5)
```

Failure handling

A zero or negative scanSeconds value throws ArgumentOutOfRangeException. Allow a completion budget longer than the probe setting.

5.5 CheckScanning

Declaration

```
bool CheckScanning()
```

Reports whether the session discovery watcher is currently probing.

Parameters

No parameters.

Result and behavior

True while discovery is probing; False when no watcher exists or probing is not active.

Use this as a status indicator, not as a controller-connection health test. A False value does not mean that a controller is reachable, and True does not identify a controller. Prefer the completion event to a tight polling loop.

C# and VB.NET usage

C#

```
bool scanning = sdk.CheckScanning();
Console.WriteLine(scanning ? "Scanning" : "Not scanning");
```

VB.NET

```
Dim scanning As Boolean = sdk.CheckScanning()
Console.WriteLine(If(scanning, "Scanning", "Not scanning"))
```

Failure handling

A disposed session throws `ObjectDisposedException`.

5.6 GetControllers

Declaration

```
RobotList[] GetControllers()
```

Returns the current controller-discovery snapshot.

Parameters

No parameters.

Result and behavior

An array of `RobotList` descriptors. Before a scan, or when no devices are available, the array is empty.

Read the array after `ControllersAvailableEvent`. Index is the position in that snapshot; `BaseAddress` and `ID` identify the target used by subsequent calls. Store the selected `RobotList`, not only its Index. Robot metadata can be unavailable or empty depending on what the controller reports.

C# and VB.NET usage

C#

```
RobotList[] controllers = sdk.GetControllers();
foreach (RobotList item in controllers)
    Console.WriteLine($"{item.Index}: {item.Name} {item.BaseAddress}");
```

VB.NET

```
Dim controllers As RobotList() = sdk.GetControllers()
For Each item As RobotList In controllers
    Console.WriteLine($"{item.Index}: {item.Name} {item.BaseAddress}")
Next
```

Failure handling

Do not access controllers(0) or controllers[0] until you have checked that the array contains an item.

Chapter 6

Reading and Writing Variables

6.1 Choosing a read pattern

Pattern	Result channel	Typical use
GetVariable	Returned String	Console tools and worker-thread reads
ReadVariableAsync	Awaitable Task of String	Responsive desktop logic and request-specific results
GetVariableAsync	VariableAsyncFinished event	Event-oriented application integration

Text returned by a read is the controller value representation. A numeric value can be parsed with invariant culture. A position or axis structure is not automatically a C# or VB.NET record. Preserve its text or add an application-specific parser that matches the controller representation you intend to support.

6.2 Numeric values and culture

Do not rely on the workstation's decimal separator when parsing controller numeric text. Handle parsing failure explicitly. Not every variable is a scalar number; a structured value or an enum token must be interpreted according to its own format.

C#

```
string text = sdk.GetVariable(robot, "MY_COUNTER");
if (int.TryParse(text, NumberStyles.Integer,
    CultureInfo.InvariantCulture, out int counter))
    Console.WriteLine(counter);
else
    Console.WriteLine("Value is not an integer: " + text);
```

VB.NET

```
Dim text As String = sdk.GetVariable(robot, "MY_COUNTER")
Dim counter As Integer
If Integer.TryParse(text, NumberStyles.Integer,
    CultureInfo.InvariantCulture, counter) Then
    Console.WriteLine(counter)
Else
    Console.WriteLine("Value is not an integer: " & text)
End If
```

6.3 Controller operating modes

Controller text	OpMode value	Meaning for SetVariable
#AUT	Automatic	SDK permits the write request
#EX	ExternalAutomatic	SDK permits the write request
#T1	TestReducedSpeed	SDK rejects the write request
#T2	TestFullSpeed	SDK rejects the write request
Other text	Unknown	SDK rejects the write request

The enum is not a flags enum. Compare values with equality and logical OR (| | / OrElse); do not combine enum values using bitwise OR to test a mode. Even in AUT/EXT, the controller can reject an unavailable variable, invalid text or a write that its access rules do not permit.

6.4 GetVariable

Declaration

```
string GetVariable(RobotList Robot, string RobotVariable)
```

Reads one controller variable and returns its current value as text.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotVariable	String	The name of an existing, accessible controller variable, such as \$MODE_OP or \$POS_ACT. A blank name is rejected.

Result and behavior

The current value converted to a String using invariant culture.

The method makes one read request. Numeric, Boolean and structured controller values are presented as text; the SDK does not automatically turn a position or axis structure into a strongly typed application object. Keep the controller representation when logging structured values. An unavailable value is an error, not an empty-success marker.

C# and VB.NET usage

C#

```
string modeText = sdk.GetVariable(robot, "$MODE_OP");
string positionText = sdk.GetVariable(robot, "$POS_ACT");
Console.WriteLine(modeText);
Console.WriteLine(positionText);
```

VB.NET

```
Dim modeText As String = sdk.GetVariable(robot, "$MODE_OP")
Dim positionText As String = sdk.GetVariable(robot, "$POS_ACT")
Console.WriteLine(modeText)
Console.WriteLine(positionText)
```

Failure handling

Blank variable names are rejected. An unavailable variable or missing current value throws `InvalidOperationException`. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

6.5 ReadVariableAsync

Declaration

```
Task<string> ReadVariableAsync(RobotList Robot, string RobotVariable)
```

Reads a controller variable using a Task-based asynchronous operation.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotVariable	String	The name of an existing, accessible controller variable, such as \$MODE_OP or \$POS_ACT. A blank name is rejected.

Result and behavior

A Task whose result is the current value as text.

Await the Task to obtain the result and observe exceptions. The SDK also raises `VariableAsyncFinished` after obtaining the value, including the controller timestamp. The Task is already asynchronous; do not wrap it in `Task.Run` in a Windows Forms handler. This API does not accept a `CancellationToken`.

C# and VB.NET usage

C#

```
string value = await sdk.ReadVariableAsync(robot, "$MODE_OP");
Console.WriteLine(value);
```

VB.NET

```
Dim value As String = Await sdk.ReadVariableAsync(robot, "$MODE_OP")
Console.WriteLine(value)
```

Failure handling

Observe errors around `Await`. A UI timeout does not cancel the underlying controller request.

6.6 GetVariableAsync

Declaration

```
string GetVariableAsync(RobotList Robot, string RobotVariable)
```

Starts an event-based asynchronous variable read.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotVariable	String	The name of an existing, accessible controller variable, such as \$MODE_OP or \$POS_ACT. A blank name is rejected.

Result and behavior

An empty String immediately. This return value is not the variable value.

Subscribe to VariableAsyncFinished and OperationFailed before sending the request. A successful read publishes the name, text value and timestamp. A failed read is reported through OperationFailed. For application code that needs a result or exception tied to a particular request, prefer ReadVariableAsync.

C# and VB.NET usage

C#

```
sdk.VariableAsyncFinished += (sender, e) =>
    Console.WriteLine($"{e.Name} = {e.Value} at {e.Timestamp:0}");
sdk.OperationFailed += (sender, failure) =>
    Console.Error.WriteLine(failure.Message);
sdk.GetVariableAsync(robot, "$MODE_OP");
```

VB.NET

```
AddHandler sdk.VariableAsyncFinished, Sub(sender, e)
    Console.WriteLine($"{e.Name} = {e.Value} at {e.Timestamp:0}")
End Sub
AddHandler sdk.OperationFailed, Sub(sender, failure)
    Console.Error.WriteLine(failure.Message)
End Sub
sdk.GetVariableAsync(robot, "$MODE_OP")
```

Failure handling

Do not wait for a meaningful return String. Keep the session alive for the completion event.

6.7 GetOpMode

Declaration

```
OpMode GetOpMode(RobotList Robot)
```

Reads \$MODE_OP and maps the controller value to the OpMode enumeration.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

ExternalAutomatic, Automatic, TestReducedSpeed, TestFullSpeed or Unknown.

The mapping is #EX to ExternalAutomatic, #AUT to Automatic, #T1 to TestReducedSpeed and #T2 to TestFullSpeed. Unknown means that a returned mode string was not recognized; it does not suppress communication exceptions. A mode read is a snapshot and can change before a later command.

C# and VB.NET usage

C#

```
OpMode mode = sdk.GetOpMode(robot);
if (mode == OpMode.Automatic || mode == OpMode.ExternalAutomatic)
    Console.WriteLine("Automatic operating mode.");
```

VB.NET

```
Dim mode As OpMode = sdk.GetOpMode(robot)
If mode = OpMode.Automatic OrElse mode = OpMode.ExternalAutomatic Then
    Console.WriteLine("Automatic operating mode.")
End If
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

6.8 SetVariable

Declaration

```
void SetVariable(RobotList Robot, string RobotVariable, string Value)
```

Writes a text value to an existing controller variable.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotVariable	String	The name of an existing, accessible controller variable, such as \$MODE_OP or \$POS_ACT. A blank name is rejected.
Value	String	The controller-compatible value text. Null is rejected. The variable must already exist.

Result and behavior

No return value. Success means the write completed without a reported controller error.

The SDK reads the operating mode and permits this operation only in AUT or EXT. It then reads the target variable, replaces its current value with the supplied text, and writes it back. The controller determines whether that value is valid for the variable and whether the variable is writable. A successful call is not a robot-motion authorization or a transaction covering later commands.

C# and VB.NET usage

C#

```
// MY_JOB_ID must already exist and be writable.
sdk.SetVariable(robot, "MY_JOB_ID", "12");
Console.WriteLine("Write completed.");
```

VB.NET

```
' MY_JOB_ID must already exist and be writable.
sdk.SetVariable(robot, "MY_JOB_ID", "12")
Console.WriteLine("Write completed.")
```

Failure handling

Outside AUT/EXT, InvalidOperationException is raised and no write request is sent. Variable availability, controller validation and permissions can also reject the write.

Chapter 7

Variable Monitoring

7.1 Configure, start, consume, stop

Configure the watch list with `AddVariableWatch`, subscribe to `VariableStatusChanged` and `OperationFailed`, then call `StartVariableMonitor`. Each controller has a separate watch list and polling connection. Stop a controller with `StopVariableMonitor` when its live display is no longer needed.

The standard watcher reads a batch of identifiers every 500 ms under normal conditions. This is an interval setting, not a guaranteed end-to-end sampling period. A slow controller read lengthens the update period. The high-level KRC API does not expose a polling-interval argument.

7.2 Change semantics

The first read of an identifier can produce a notification because the cached value is initially empty. Subsequent callbacks describe changed values. A timestamp-only update with the same value is not necessarily a value-change notification. Do not use callback count as the count of all controller samples.

Each changed variable is reported separately through the KRC event. `FullName` is useful as the displayed key; `RobotIndex` labels the watcher target. The event does not include an application request ID or the full `RobotList` metadata. Maintain any extra correlation in the application.

7.3 Errors and recovery

A polling read failure is reported through `OperationFailed`. The failed connection is closed, and later polling attempts can create a new connection. Keep the application error handler subscribed while the monitor runs. A repeated access error can indicate an unavailable variable or persistent permissions issue, not merely a network interruption.

Keep handlers short. Copy the values you need and dispatch work to the UI or a processing queue. Do not issue slow program or file operations directly inside a watch callback.

7.4 AddVariableWatch

Declaration

```
void AddVariableWatch(RobotList Robot, string VariableName)
```

Adds a variable to the watch list for a controller.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. <code>BaseAddress</code> must contain a non-empty host or IP address. Use the descriptor itself; do not use <code>Index</code> to look up another robot.
VariableName	String	The existing variable name to monitor. A blank name is rejected.

Result and behavior

No return value.

Each controller has its own watcher within the session. Adding the same identifier again does not create a duplicate entry. Adding a watch does not start the monitor; call `StartVariableMonitor` after configuring the list. For a changing watch configuration, serialize user actions rather than issuing conflicting add/remove requests.

C# and VB.NET usage

C#

```
sdk.AddVariableWatch(robot, "$MODE_OP");
sdk.AddVariableWatch(robot, "$POS_ACT");
```

VB.NET

```
sdk.AddVariableWatch(robot, "$MODE_OP")
sdk.AddVariableWatch(robot, "$POS_ACT")
```

Failure handling

Variable existence is evaluated by reading during monitoring; adding a name alone is not a connectivity or availability test.

7.5 RemoveVariableWatch

Declaration

```
void RemoveVariableWatch(RobotList Robot, string VariableName)
```

Removes a variable from that controller watch list.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
VariableName	String	The name of the watch entry to remove.

Result and behavior

No return value.

The removed identifier is excluded from subsequent batches. Removing a name that is not in the list has no effect on the list. This does not stop monitoring other identifiers or remove a watcher for another controller. `StopVariableMonitor` closes the polling connection when monitoring is no longer required.

C# and VB.NET usage

C#

```
sdk.RemoveVariableWatch(robot, "$POS_ACT");
```

VB.NET

```
sdk.RemoveVariableWatch(robot, "$POS_ACT")
```

Failure handling

A blank name is rejected. A callback already in progress may still reach an application queue; use your session-lifetime guard when closing a UI.

7.6 StartVariableMonitor

Declaration

```
void StartVariableMonitor(RobotList Robot)
```

Starts periodic batch reads for the controller watch list.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value; changed values arrive through VariableStatusChanged.

The standard watcher uses a 500 ms interval and reads all watched identifiers in one batch. Only changed values are reported; the first available read is a change relative to the initial empty cache. Calling Start again while running does not create another timer. Read failures are reported through OperationFailed and the polling connection is recreated for later attempts.

C# and VB.NET usage

C#

```
sdk.VariableStatusChanged += (sender, e) =>
    Console.WriteLine($"Robot {e.RobotIndex}: {e.FullName} = {e.Value}");
sdk.AddVariableWatch(robot, "$MODE_OP");
sdk.StartVariableMonitor(robot);
```

VB.NET

```
AddHandler sdk.VariableStatusChanged, Sub(sender, e)
    Console.WriteLine($"Robot {e.RobotIndex}: {e.FullName} = {e.Value}")
End Sub
sdk.AddVariableWatch(robot, "$MODE_OP")
sdk.StartVariableMonitor(robot)
```

Failure handling

This is software polling, not a hard real-time sampling guarantee. A slow read extends the actual update period; keep event handlers short.

7.7 StopVariableMonitor

Declaration

```
void StopVariableMonitor(RobotList Robot)
```

Stops polling for the selected controller and closes its polling connection.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value.

The watch names remain configured, so a later StartVariableMonitor can resume monitoring. Stopping one controller does not stop another. Dispose ends all watchers owned by the session. Because stopping synchronizes with an active read, do not call this directly on a UI thread if controller communication can be slow.

C# and VB.NET usage

C#

```
await Task.Run(() => sdk.StopVariableMonitor(robot));
Console.WriteLine("Monitoring stopped.");
```

VB.NET

```
Await Task.Run(Sub() sdk.StopVariableMonitor(robot))
Console.WriteLine("Monitoring stopped.")
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

Chapter 8

Files and Directory Transfers

8.1 Two independent path spaces

Path space	Example	Interpretation
Controller file	KRC:\R1\Program\Test.src	A file accessed through the controller file-handling service
Controller directory	KRC:\R1\Program\CustomerJobs	A remote directory root
Local file	C:\RAS\Upload\Test.src	A Windows file read or written by the PC application
Local directory	C:\RAS\RobotFiles	A directory in the PC filesystem

Never use a local PC path where a controller path is expected. In C#, verbatim string literals such as `@"KRC:\R1\Program"` preserve backslashes. VB.NET strings do not use the C# backslash escape syntax.

8.2 Single-file versus tree transfer

Single-file download separates the local destination directory and the relative filename. Directory transfer copies the source directory's contents under the specified destination root. For example, copying CustomerJobs into a local destination already named CustomerJobs does not create a second nested CustomerJobs automatically.

Tree transfers preserve empty subdirectories. Upload does not synchronize away remote files absent from the local source. If a workflow requires an exact mirror, design and confirm the deletion stage separately; it is not implied by UploadDirectoryToRobot.

8.3 Completion and partial output

Completion events indicate success after the requested operation, not a percentage of bytes transferred. The high-level API does not expose a progress-percentage event. Synchronous transfer methods should run on a worker in UI applications.

Warning: Upload can replace controller data, and deletion removes it. Show the exact controller and target path before the action. Directory operations are not transactional; inspect partial results before retrying a failed operation.

An existing destination file can be overwritten. Use a new output folder when preserving earlier downloads matters. Do not mark a failed transfer complete merely because its destination file exists.

8.4 FileExist

Declaration

```
bool FileExist(RobotList Robot, string RobotPath)
```

Checks whether a controller-side file exists.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side file path, for example KRC:\R1\Program\Test.src. This is not a path on the PC.

Result and behavior

True if the file exists; False if it does not.

This is a point-in-time existence check. The file may change or be deleted before a following download. A False result does not identify a communication failure; communication or permission failures are exceptions. Use a file path rather than a directory path.

C# and VB.NET usage

C#

```
bool exists = sdk.FileExist(robot, @"KRC:\R1\Program\Test.src");
Console.WriteLine(exists);
```

VB.NET

```
Dim exists As Boolean = sdk.FileExist(robot, "KRC:\R1\Program\Test.src")
Console.WriteLine(exists)
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.5 DirectoryExist

Declaration

```
bool DirectoryExist(RobotList Robot, string RobotPath)
```

Checks whether a controller-side directory exists.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side directory path, for example KRC:\R1\Program. This is not a path on the PC.

Result and behavior

True if the directory exists; False if it does not.

Use this for directory workflows such as validating a listing root or deciding whether an upload target needs to be created. It does not reserve the directory or grant write access. DirectoryExist is the exact method name; the high-level API uses the singular spelling.

C# and VB.NET usage

C#

```
bool exists = sdk.DirectoryExist(robot, @"KRC:\R1\Program");
Console.WriteLine(exists);
```

VB.NET

```
Dim exists As Boolean = sdk.DirectoryExist(robot, "KRC:\R1\Program")
Console.WriteLine(exists)
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.6 GetFilesList

Declaration

```
IList<string> GetFilesList(RobotList Robot, string RootPath, bool WithSubFiles = true)
```

Lists files below a controller directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RootPath	String	The controller-side directory to list.
WithSubFiles	Boolean	True includes files in subdirectories. Default: True.

Result and behavior

An IList of controller-side file path strings.

Returned values are paths, not file contents. Recursive listing can take longer for large trees. Use each returned complete path for a later download, existence check or deletion. Results are snapshots and are not guaranteed to remain unchanged. The API does not provide a search-pattern parameter on this high-level method.

C# and VB.NET usage

C#

```
IList<string> files = sdk.GetFilesList(
    robot, @"KRC:\R1\Program", WithSubFiles: true);
foreach (string path in files) Console.WriteLine(path);
```

VB.NET

```
Dim files As IList(Of String) = sdk.GetFilesList(
    robot, "KRC:\R1\Program", WithSubFiles:=True)
For Each path As String In files
    Console.WriteLine(path)
Next
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.7 GetDirectoriesList

Declaration

```
IList<string> GetDirectoriesList(RobotList Robot, string RootPath,
    bool WithSubDirectories = true)
```

Lists directories below a controller directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RootPath	String	The controller-side listing root.
WithSubDirectories	Boolean	True includes nested directories. Default: True.

Result and behavior

An IList of controller-side directory paths.

This method returns directory names independently of file listing. Use a non-recursive call for a browser showing one level, and a recursive call when planning a complete transfer. Do not assume a particular sort order; sort the returned values in the application if order matters.

C# and VB.NET usage

C#

```
IList<string> folders = sdk.GetDirectoriesList(
    robot, @"KRC:\R1\Program", WithSubDirectories: false);
foreach (string path in folders) Console.WriteLine(path);
```

VB.NET

```
Dim folders As IList(Of String) = sdk.GetDirectoriesList(
    robot, "KRC:\R1\Program", WithSubDirectories:=False)
For Each path As String In folders
    Console.WriteLine(path)
Next
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.8 CreateDirectoryOnRobot

Declaration

```
bool CreateDirectoryOnRobot(RobotList Robot, string RobotPath)
```

Creates a controller-side directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side directory path, for example KRC:\R1\Program. This is not a path on the PC.

Result and behavior

True when the creation operation completes; failures are exceptions.

If your workflow accepts an existing directory, check DirectoryExist first. Parent-directory requirements and access rights are enforced by the controller. The method does not create a directory on the PC. Do not treat False as the documented failure path for a rejected creation request.

C# and VB.NET usage

C#

```
string folder = @"KRC:\R1\Program\CustomerJobs";
if (!sdk.DirectoryExist(robot, folder))
    sdk.CreateDirectoryOnRobot(robot, folder);
```

VB.NET

```
Dim folder As String = "KRC:\R1\Program\CustomerJobs"
If Not sdk.DirectoryExist(robot, folder) Then
    sdk.CreateDirectoryOnRobot(robot, folder)
End If
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.9 DownloadFileToPC

Declaration

```
void DownloadFileToPC(RobotList Robot, string RobotPath, string DestinationPath,
    string DestinationName)
```

Copies a controller file to a PC directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side file path, for example KRC:\R1\Program\Test.src. This is not a path on the PC.
DestinationPath	String	Local destination directory. It is created if missing.
DestinationName	String	A relative local filename, optionally including subdirectories. It must stay within DestinationPath.

Result and behavior

No return value. DownloadFileCompletedEvent is raised after a successful transfer.

DestinationPath and DestinationName are combined as separate path components. Absolute filenames and paths escaping the destination directory are rejected. Transfer is streamed; the application does not receive all bytes in a return value. An existing local file at the resolved target can be overwritten. Completion means the transfer completed, not that the program was validated or selected on the robot.

C# and VB.NET usage

C#

```
sdk.DownloadFileToPC(robot,
    @"KRC:\R1\Program\Test.src",
    @"C:\RAS\RobotFiles", "Test.src");
```

VB.NET

```
sdk.DownloadFileToPC(robot,
    "KRC:\R1\Program\Test.src",
    "C:\RAS\RobotFiles", "Test.src")
```

Failure handling

Local path traversal is rejected with ArgumentException. Local filesystem errors and controller faults propagate. A failed transfer must not be assumed complete.

8.10 UploadFileToRobot

Declaration

```
void UploadFileToRobot(RobotList Robot, string SourcePath, string DestinationPath)
```

Copies an existing PC file to a controller file path.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
SourcePath	String	An existing local file to read.
DestinationPath	String	The complete controller-side file path, including the filename.

Result and behavior

No return value. UploadFileCompletedEvent is raised after success.

The transfer requests target-directory creation. Controller permissions and restrictions still apply. Uploading data is separate from program selection and execution. Treat the target as an explicit file path and make an overwrite decision in the application before sending a file with an existing name.

C# and VB.NET usage

C#

```
sdk.UploadFileToRobot(robot,
    @"C:\RAS\Upload\Test.src",
    @"KRC:\R1\Program\Test.src");
```

VB.NET

```
sdk.UploadFileToRobot(robot,
    "C:\RAS\Upload\Test.src",
    "KRC:\R1\Program\Test.src")
```

Failure handling

A missing or unreadable SourcePath is a local filesystem error. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.11 DownloadDirectoryToPC

Declaration

```
void DownloadDirectoryToPC(RobotList Robot, string RobotPath, string DestinationPath)
```

Copies a complete controller directory tree to a local directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side directory path, for example KRC:\R1\Program. This is not a path on the PC.
DestinationPath	String	The local root into which the controller directory contents are copied.

Result and behavior

No return value. DownloadDirectoryCompletedEvent is raised when the full operation succeeds.

Files and empty subdirectories are preserved beneath the local destination root. The content of RobotPath is placed under DestinationPath; the SDK does not automatically add another directory named after the robot source root. Returned remote paths are checked against the requested root before resolving local destinations. This is a recursive operation and can partially populate the destination before a failure.

C# and VB.NET usage

C#

```
sdk.DownloadDirectoryToPC(robot,
    @"KRC:\R1\Program\CustomerJobs",
    @"C:\RAS\RobotFiles\CustomerJobs");
```

VB.NET

```
sdk.DownloadDirectoryToPC(robot,
    "KRC:\R1\Program\CustomerJobs",
    "C:\RAS\RobotFiles\CustomerJobs")
```

Failure handling

A missing controller directory throws DirectoryNotFoundException. Local path checks, filesystem errors or remote failures can also stop the transfer.

8.12 UploadDirectoryToRobot

Declaration

```
void UploadDirectoryToRobot(RobotList Robot, string SourcePath, string DestinationPath)
```

Copies the contents of a local directory tree to a controller directory.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
SourcePath	String	Existing local source directory.
DestinationPath	String	Controller-side destination root for the source contents.

Result and behavior

No return value. UploadDirectoryCompletedEvent is raised after the tree upload succeeds.

The method creates the destination when needed, preserves empty subdirectories and uploads files relative to SourcePath. It does not automatically add another level named after SourcePath. A transfer can change some controller files before a later failure; it is not an all-or-nothing transaction. The operation does not delete destination files that are absent from the local source.

C# and VB.NET usage

C#

```
sdk.UploadDirectoryToRobot(robot,
    @"C:\RAS\Upload\CustomerJobs",
    @"KRC:\R1\Program\CustomerJobs");
```

VB.NET

```
sdk.UploadDirectoryToRobot(robot,
    "C:\RAS\Upload\CustomerJobs",
    "KRC:\R1\Program\CustomerJobs")
```

Failure handling

A missing local source throws DirectoryNotFoundException. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.13 DeleteFileFromRobot

Declaration

```
void DeleteFileFromRobot(RobotList Robot, string RobotPath)
```

Deletes the specified controller-side file.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side file path, for example KRC:\R1\Program\Test.src. This is not a path on the PC.

Result and behavior

No return value. DeleteFileCompletedEvent is raised after success.

Deletion acts on the remote path, not a local PC file. The application should display the selected controller and complete file path before executing this operation. Keep a suitable backup for files that must be recoverable. Existence checks do not prevent another actor from changing the file between the check and deletion.

C# and VB.NET usage

C#

```
// Execute only after the user confirms the remote target.
sdk.DeleteFileFromRobot(robot,
    @"KRC:\R1\Program\ObsoleteJob.src");
```

VB.NET

```
' Execute only after the user confirms the remote target.
sdk.DeleteFileFromRobot(robot,
    "KRC:\R1\Program\ObsoleteJob.src")
```

Failure handling

The controller can reject deletion of a protected or in-use file. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

8.14 DeleteDirectoryFromRobot

Declaration

```
void DeleteDirectoryFromRobot(RobotList Robot, string RobotPath)
```

Deletes the controller directory and its contents.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
RobotPath	String	A complete controller-side directory path, for example KRC:\R1\Program. This is not a path on the PC.

Result and behavior

No return value. DeleteDirectoryCompletedEvent is raised after the entire operation succeeds.

The SDK lists and deletes files, then deletes nested directories from the deepest paths upward, and finally deletes the requested root. This is recursive and destructive. A failure can leave part of the tree deleted and part still present. Confirm the target and do not use a broad program root when only one customer-job folder should be removed.

C# and VB.NET usage

C#

```
// Confirm the exact controller and directory first.  
sdk.DeleteDirectoryFromRobot(robot,  
    @"KRC:\R1\Program\ObsoleteJobs");
```

VB.NET

```
' Confirm the exact controller and directory first.  
sdk.DeleteDirectoryFromRobot(robot,  
    "KRC:\R1\Program\ObsoleteJobs")
```

Failure handling

Deletion is not rolled back on failure. Missing paths, permissions or controller faults can stop the operation.

Chapter 9

Backups and Active Project Download

9.1 Choose the backup output

An archive ZIP and an active WorkVisual project package are different outputs. Backup can request either or both. GetProject downloads only the active project. A file-tree download is different again and should not be described as a complete controller archive.

Use a distinct output directory for each backup run. Repeating a backup for the same robot Name in the same directory can overwrite the archive with the same name. Record the robot address/ID, requested options, destination and application correlation number in your own job log.

9.2 Correlating completion

The Index argument to Backup is supplied by the caller. BackupCompletedEvent reports that same number. It is not an index used by the SDK to choose a controller. Pass the selected RobotList as the target and use Index as a job identifier or as your own display correlation.

When both outputs are requested, the completed event represents total success. If an archive succeeded and the project download failed, an archive file may be present even though the combined job failed. Check expected outputs and retain the exception detail before retrying.

9.3 What this API does not do

The high-level SDK described here does not expose archive restore or active-project deployment. A downloaded archive or project package is a backup artifact; downloading it does not validate a restore workflow or change the active controller project.

9.4 GetProject

Declaration

```
void GetProject(RobotList Robot, string DestinationFolder)
```

Downloads the active WorkVisual project from the selected controller.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
DestinationFolder	String	Local directory to receive the controller-provided project file. Created if missing.

Result and behavior

No return value. Return from the call indicates success; there is no dedicated project-completion event.

The SDK obtains the active project from the device-information service and requests its download. The controller determines the project filename. This is a project package download, not a general-purpose directory copy, archive restore or project deployment operation. Use a separate backup folder for each run when existing project packages must be retained.

C# and VB.NET usage

C#

```
sdk.GetProject(robot, @"C:\RAS\Backups\Run_001\Project");
Console.WriteLine("Active project downloaded.");
```

VB.NET

```
sdk.GetProject(robot, "C:\RAS\Backups\Run_001\Project")
Console.WriteLine("Active project downloaded.")
```

Failure handling

An invalid returned filename is rejected. Project-service errors, filesystem errors and checksum validation can also fail the download.

9.5 Backup

Declaration

```
void Backup(RobotList Robot, string BackupFolder, int Index, bool ZipArchive = true,
            bool ProjectBackup = true)
```

Creates an archive backup, downloads the active project, or performs both operations.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
BackupFolder	String	Local output directory. Created if missing.
Index	Int32	Caller-defined correlation number reported by BackupCompletedEvent. It does not select the target robot.
ZipArchive	Boolean	Create a controller archive ZIP. Default: True.
ProjectBackup	Boolean	Download the active project. Default: True.

Result and behavior

No return value. BackupCompletedEvent carries Index on total success; BackupFailedEvent carries the failure text.

At least one option must be True. Archive files use the robot Name, with invalid filename characters replaced; an empty name uses controller.zip. When both options are requested, completion means both succeeded. The archive may already exist if the project download subsequently fails. Failure raises BackupFailedEvent and also propagates the exception to the caller; do not count these as two separate failed jobs.

C# and VB.NET usage

C#

```
sdk.BackupCompletedEvent += (sender, index) =>
    Console.WriteLine($"Backup job {index} completed.");
sdk.Backup(robot, @"C:\RAS\Backups\Run_001", Index: 101,
    ZipArchive: true, ProjectBackup: true);
```

VB.NET

```
AddHandler sdk.BackupCompletedEvent, Sub(sender, index)
    Console.WriteLine($"Backup job {index} completed.")
End Sub
sdk.Backup(robot, "C:\RAS\Backups\Run_001", Index:=101,
    ZipArchive:=True, ProjectBackup:=True)
```

Failure handling

Both options False throws ArgumentException. Partial backup output is possible; use the completion result and verify expected files.

Chapter 10

Robot Program Operations

10.1 Three concepts that must stay separate

Concept	Read or control API	Purpose
Controller operating mode	GetOpMode	AUT, EXT, T1 or T2 controller mode
Interpreter execution mode	GetProgramMode / SetProgramMode	MStep, IStep, Go or BStep program execution behavior
Interpreter state	GetProgramState	Free, Stop, Reset, Active, End or Unknown

None of these enums is a flags enum. Use equality comparisons. The high-level program methods operate on the robot interpreter; they do not select a submit interpreter.

10.2 State conditions used by the SDK

Operation	SDK condition before forwarding
SelectProgram	Interpreter must be Free
DeselectProgram	Interpreter must be Stop or Reset
StartProgram / StopProgram / ResetProgram / SetLine	Interpreter must not be Unknown or Free
SetProgramMode	NewMode must be a defined mode other than Unknown

These checks are necessary application conditions, not guarantees that the controller will accept a command. The controller enforces its own authorization, operating and program-validity rules. A state can also change between your display update and a later command.

10.3 Operator-driven workflow

Prepare the file on the controller, confirm the target, read the interpreter state, select the program if it is Free and obtain an explicit application start request. StartProgram starts the already selected program; it does not use a filename from a textbox unless the application previously called SelectProgram with that path.

For stop/deselect, the SDK checks for the requested state transition after the command. It makes a bounded series of state reads separated by short waits. A TimeoutException means that the expected transition was not confirmed; do not treat it as proof that the command was never received.

Warning: Program-stop and program-reset APIs are application commands. They are not safety-rated stopping functions. Keep the cell's safety and authorization procedures independent of the PC application.

10.4 GetProgramState

Declaration

```
ProgramStates GetProgramState(RobotList Robot)
```

Reads the robot interpreter state.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

Unknown, Free, Stop, Reset, Active or End.

The state is obtained from the robot interpreter, not a submit interpreter. It is a snapshot, so another actor or the controller can change the state before a command is sent. Unknown indicates a reported state; a missing interpreter is a separate InvalidOperationException. Use states to present the expected next operation, while still handling controller rejection.

C# and VB.NET usage

C#

```
ProgramStates state = sdk.GetProgramState(robot);
Console.WriteLine(state);
```

VB.NET

```
Dim state As ProgramStates = sdk.GetProgramState(robot)
Console.WriteLine(state)
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

10.5 GetProgramMode

Declaration

```
ProgramModes GetProgramMode(RobotList Robot)
```

Reads the robot interpreter program-execution mode.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

Unknown, MStep, IStep, Go or BStep.

ProgramModes is different from the controller OpMode. For example, Go describes interpreter execution behavior; it does not imply that the controller is in AUT. Display these values separately in the application and avoid using one as a substitute for the other.

C# and VB.NET usage

C#

```
ProgramModes mode = sdk.GetProgramMode(robot);
Console.WriteLine($"Program mode: {mode}");
```

VB.NET

```
Dim mode As ProgramModes = sdk.GetProgramMode(robot)
Console.WriteLine($"Program mode: {mode}")
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

10.6 SetProgramMode

Declaration

```
void SetProgramMode(RobotList Robot, ProgramModes NewMode)
```

Sets the robot interpreter program-execution mode.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
NewMode	ProgramModes	MStep, IStep, Go or BStep. Unknown and undefined numeric values are not accepted.

Result and behavior

No return value.

The SDK validates the enumeration value and forwards the request. The controller still decides whether the mode change is permitted in its current condition. This does not change the controller operating-mode selector and is not a program start request.

C# and VB.NET usage

C#

```
sdk.SetProgramMode(robot, ProgramModes.Go);
```

VB.NET

```
sdk.SetProgramMode(robot, ProgramModes.Go)
```

Failure handling

Unknown or an undefined enum value throws `ArgumentOutOfRangeException`. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

10.7 SelectProgram

Declaration

```
void SelectProgram(RobotList Robot, string KRCPath)
```

Selects a controller-resident program in the robot interpreter.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
KRCPath	String	Non-empty controller path to the program to select.

Result and behavior

No return value.

The interpreter must be Free before this call. Selection does not upload the file or start execution. When preparing a new program, upload it separately, verify the intended controller path, then select it. The controller can reject files that are missing, inaccessible or invalid.

C# and VB.NET usage

C#

```
sdk.SelectProgram(robot, @"KRC:\R1\Program\CustomerJob.src");
```

VB.NET

```
sdk.SelectProgram(robot, "KRC:\R1\Program\CustomerJob.src")
```

Failure handling

A blank path is rejected. An interpreter that is not Free throws `InvalidOperationException`. Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

10.8 DeselectProgram

Declaration

```
void DeselectProgram(RobotList Robot)
```

Removes the selected robot program from the interpreter.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value.

The SDK accepts this operation only in Stop or Reset. It requests deselection and checks for Free afterwards. If the program is Active, first issue a permitted StopProgram and confirm its state. Do not assume that deselection deletes the program file; the file remains on the controller.

C# and VB.NET usage

C#

```
ProgramStates state = sdk.GetProgramState(robot);
if (state == ProgramStates.Stop || state == ProgramStates.Reset)
    sdk.DeselectProgram(robot);
```

VB.NET

```
Dim state As ProgramStates = sdk.GetProgramState(robot)
If state = ProgramStates.[Stop] OrElse state = ProgramStates.Reset Then
    sdk.DeselectProgram(robot)
End If
```

Failure handling

Other states throw `InvalidOperationException`. Failure to reach the confirmed deselected state can throw `TimeoutException`.

10.9 StartProgram

Declaration

```
void StartProgram(RobotList Robot)
```

Requests execution of the selected robot program.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value.

Unknown and Free are rejected by the SDK. Other states are forwarded to the controller, which enforces its own start conditions. This function does not choose a program from an application textbox; it starts the program already selected in the interpreter. Require explicit operator authorization in the application and verify the selected controller/workflow before sending the request.

C# and VB.NET usage

C#

```
// Call only in an explicitly authorized start workflow.
sdk.StartProgram(robot);
```

VB.NET

```
' Call only in an explicitly authorized start workflow.
sdk.StartProgram(robot)
```

Failure handling

Unknown or Free throws InvalidOperationException. Controller conditions can reject a start even when a program is selected.

10.10 StopProgram

Declaration

```
void StopProgram(RobotList Robot)
```

Requests a stop of the selected robot program.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value.

Unknown and Free are rejected. After sending the stop request, the SDK checks that the interpreter is no longer Active. This is an application program-stop command; it is not an emergency-stop or safety-rated function.

Monitor the actual machine response and operating procedures independently.

C# and VB.NET usage

C#

```
sdk.StopProgram(robot);
Console.WriteLine(sdk.GetProgramState(robot));
```

VB.NET

```
sdk.StopProgram(robot)
Console.WriteLine(sdk.GetProgramState(robot))
```

Failure handling

A selected-program check or controller request can fail. If the expected interpreter transition is not confirmed, TimeoutException can be raised.

10.11 ResetProgram

Declaration

```
void ResetProgram(RobotList Robot)
```

Requests reset of the selected robot interpreter program.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value.

Unknown and Free are rejected by the SDK. Reset is a program operation and does not delete or upload program files. This method forwards the request; it does not automatically stop an Active program first. Build a state-aware workflow and let the controller validate whether reset is allowed.

C# and VB.NET usage

C#

```
sdk.ResetProgram(robot);
Console.WriteLine(sdk.GetProgramState(robot));
```

VB.NET

```
sdk.ResetProgram(robot)
Console.WriteLine(sdk.GetProgramState(robot))
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

10.12 SetLine

Declaration

```
void SetLine(RobotList Robot, int LineNumber)
```

Requests a program line change in the selected robot interpreter.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.
LineNumber	Int32	Requested line number. Must be at least 1.

Result and behavior

No return value.

The SDK rejects non-positive line numbers and requires a selected interpreter state rather than Unknown or Free. It does not validate that the requested line belongs to the currently selected file or is a permitted execution position. The controller performs those checks. Changing a line can alter the next program action, so keep it inside an explicitly controlled program workflow.

C# and VB.NET usage

C#

```
sdk.SetLine(robot, LineNumber: 12);
```

VB.NET

```
sdk.SetLine(robot, LineNumber:=12)
```

Failure handling

A value below 1 throws ArgumentOutOfRangeException. An unselected state or controller validation can also reject the request.

Chapter 11

Runtime Messages

11.1 Snapshot and changes

Use `ReadLogMessages` to obtain a runtime-message snapshot and `EnableMessageEvents` to receive later collection changes. The returned snapshot has `Count` and parallel arrays, not an enumerable collection of message objects. Index every array using the same valid entry number.

`MessagesChangedEventArgs` describes additions, removals, replacements, moves and resets. Its `NewItems/OldItems` entries are strings suitable for diagnostic display. Do not build a permanent message view that only appends new items and ignores removal/reset actions.

11.2 Controller correlation

Message-change event data does not contain a `RobotList` or robot index. If separate controllers need separate message panels, give each controller its own KRC session and bind the event handler to that panel's known target. Do not infer the target from message text.

11.3 Timestamp and visibility

Store the timestamp reported by the controller alongside the application's own receipt time if latency or log correlation matters. Display `IsVisible` and message type according to your operator interface requirements. A message ID or number should not be treated as a globally unique key across all controllers and all time.

11.4 ReadLogMessages

Declaration

```
RuntimeDisplayMessage ReadLogMessages(RobotList Robot)
```

Reads the controller runtime-message collection.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. <code>BaseAddress</code> must contain a non-empty host or IP address. Use the descriptor itself; do not use <code>Index</code> to look up another robot.

Result and behavior

A `RuntimeDisplayMessage` containing parallel arrays and `Count`.

The method name does not promise a complete historical event log. It reads the runtime message service collection. When the collection is empty, `Count` is zero and the arrays are empty. Iterate from zero up to `Count - 1`; use the same index for `Id`, `Text`, `Type`, `TimeStamp`, `IsVisible`, `MessageCode` and `MessageNumber`.

C# and VB.NET usage

C#

```
RuntimeDisplayMessage messages = sdk.ReadLogMessages(robot);
for (int i = 0; i < messages.Count; i++)
    Console.WriteLine(messages.MessageToString(i));
```

VB.NET

```
Dim messages As RuntimeDisplayMessage = sdk.ReadLogMessages(robot)
For i As Integer = 0 To messages.Count - 1
    Console.WriteLine(messages.MessageToString(i))
Next
```

Failure handling

Communication errors, service faults and controller permissions are reported as exceptions. Do not interpret an exception as an empty or successful result.

11.5 EnableMessageEvents

Declaration

```
void EnableMessageEvents(RobotList Robot)
```

Subscribes the session to runtime-message collection changes for a controller.

Parameters

Parameter	Type	Meaning
Robot	RobotList	The target controller descriptor. BaseAddress must contain a non-empty host or IP address. Use the descriptor itself; do not use Index to look up another robot.

Result and behavior

No return value. Updates arrive through MessagesChangedEvent.

Subscribe your application event handler before enabling events. Enabling again for the same address/ID does not create another session subscription. The SDK has no matching high-level disable method; Dispose releases owned message subscriptions. Event data describes collection changes, and its payload does not include a robot identifier. Use one session per controller if your application must route message changes unambiguously.

C# and VB.NET usage

C#

```
sdk.MessagesChangedEvent += (sender, e) =>
{
    Console.WriteLine(e.ToString());
    foreach (string message in e.NewItems)
        Console.WriteLine(message);
};
sdk.EnableMessageEvents(robot);
```

VB.NET

```
AddHandler sdk.MessagesChangedEvent, Sub(sender, e)
    Console.WriteLine(e.ToString())
    For Each message As String In e.NewItems
        Console.WriteLine(message)
    Next
End Sub
sdk.EnableMessageEvents(robot)
```

Failure handling

The controller message service and its callback channel must be reachable. Handlers must not assume execution on the UI thread.

11.6 RuntimeDisplayMessage.MessageToString

Declaration

```
string RuntimeDisplayMessage.MessageToString(int Index)
```

Formats one entry from a runtime-message snapshot for display or logging.

Parameters

Parameter	Type	Meaning
Index	Int32	Zero-based entry index in the snapshot. Use $0 \leq \text{Index} < \text{Count}$.

Result and behavior

Formatted text containing Id, Text, Type, TimeStamp, IsVisible, MessageCode and MessageNumber.

Call this on the RuntimeDisplayMessage object returned by ReadLogMessages, not on KRC. It does not perform another network read. Use the formatted string for diagnostics; use individual fields when storing structured message records.

C# and VB.NET usage

C#

```
RuntimeDisplayMessage snapshot = sdk.ReadLogMessages(robot);
if (snapshot.Count > 0)
    Console.WriteLine(snapshot.MessageToString(0));
```

VB.NET

```
Dim snapshot As RuntimeDisplayMessage = sdk.ReadLogMessages(robot)
If snapshot.Count > 0 Then
    Console.WriteLine(snapshot.MessageToString(0))
End If
```

Failure handling

An out-of-range index can cause IndexOutOfRangeException. Always check Count before indexing.

11.7 MessagesChangedEventArgs.ToString

Declaration

```
string MessagesChangedEventArgs.ToString()
```

Formats the collection-change metadata carried by MessagesChangedEvent.

Parameters

No parameters.

Result and behavior

Text showing Action, the NewItems/OldItems counts, and starting indexes.

The summary is useful for a compact log, but it does not replace reading the NewItems and OldItems entries. Use the change action and collection contents together when updating an application message view. Reset and removal changes must be handled; a view that only appends new messages will become stale.

C# and VB.NET usage

C#

```
sdk.MessagesChangedEvent += (sender, e) =>
    Console.WriteLine(e.ToString());
```

VB.NET

```
AddHandler sdk.MessagesChangedEvent, Sub(sender, e)
    Console.WriteLine(e.ToString())
End Sub
```

Failure handling

Keep the handler alive while events are enabled, and release the owning KRC session when it is no longer needed.

Chapter 12

Events and Callback Handling

12.1 Event reference

Event	Payload / completion meaning
ControllersAvailableEvent	No arguments. Refresh the controller snapshot on completed discovery or availability change.
DevicesChangedEvent	sender and String Ret. Device-removal/disconnection description.
DownloadFileCompletedEvent	No arguments. The single-file download completed.
UploadFileCompletedEvent	No arguments. The single-file upload completed.
DeleteFileCompletedEvent	No arguments. The remote file deletion completed.
UploadDirectoryCompletedEvent	No arguments. The directory upload completed.
DownloadDirectoryCompletedEvent	No arguments. The directory download completed.
DeleteDirectoryCompletedEvent	No arguments. The directory deletion completed.
MessagesChangedEvent	sender and MessagesChangedEventArgs. Collection change details.
BackupFailedEvent	sender and String. Backup failure text; the call also throws.
BackupCompletedEvent	sender and Int32. Caller-supplied Backup Index.
VariableStatusChanged	sender and VariableEventArgs. Name, FullName, Value and RobotIndex.
VariableAsyncFinished	sender and VariableAsyncEventArgs. Name, Value and TimeStamp.
OperationFailed	sender and Exception. Event-based read or monitor failure.

12.2 Which events to subscribe to

Subscribe only to the events your workflow consumes, but always observe the failure channel for event-based asynchronous reads and monitoring. A synchronous method can throw even when an event handler is installed; event subscription does not replace a try/catch around the call.

The sender identifies the event source/session context; it is not a universal controller-identity object. Parameterless file events have neither a path nor a request ID. Serialize file jobs per session or bind separate sessions to specific jobs when completion must be correlated precisely.

12.3 Callbacks and thread ownership

Assume callbacks can arrive from worker or controller-service threads. Do not read or write Windows Forms controls directly in those handlers. Copy primitive values first and post a short UI update using BeginInvoke. Keep exception handling inside application callback code so that an application failure does not escape into a service callback.

C#

```

sdk.OperationFailed += (sender, failure) =>
{
    Console.Error.WriteLine(failure.GetType().Name);
    Console.Error.WriteLine(failure.Message);
};
sdk.BackupFailedEvent += (sender, text) =>
    Console.Error.WriteLine("Backup failed: " + text);

```

VB.NET

```

AddHandler sdk.OperationFailed, Sub(sender, failure)
    Console.Error.WriteLine(failure.GetType().Name)
    Console.Error.WriteLine(failure.Message)
End Sub
AddHandler sdk.BackupFailedEvent, Sub(sender, text)
    Console.Error.WriteLine("Backup failed: " & text)
End Sub

```

12.4 Subscription lifetime

Subscribe once when establishing a workflow, not each time a Refresh button is pressed. Store named handlers if the application needs to detach them explicitly. Retain the session while events are enabled, and guard posted UI actions when the form is closing or when an old session has been replaced.

No high-level disable method is supplied for message-change events. Disposing the owning session releases the subscription. Variable monitoring has its own Stop method and can later be restarted with the same configured watch list.

Chapter 13

Windows Forms Integration

13.1 Keep the operator interface responsive

Use `Await sdk.ReadVariableAsync(...)` for a Task-based variable read. Use `Task.Run` for synchronous file, backup or program methods when they are initiated from a button handler. Capture every textbox, checkbox, numeric value and selected RobotList on the UI thread before entering the worker.

C# button-handler pattern:

C#

```
private async void ReadButton_Click(object sender, EventArgs e)
{
    RobotList target = selectedRobot;
    string name = variableNameTextBox.Text.Trim();
    readButton.Enabled = false;
    try
    {
        valueTextBox.Text = await sdk.ReadVariableAsync(target, name);
    }
    catch (Exception failure)
    {
        statusLabel.Text = failure.Message;
    }
    finally
    {
        readButton.Enabled = true;
    }
}
```

VB.NET button-handler pattern:

VB.NET

```
Private Async Sub ReadButton_Click(sender As Object, e As EventArgs)
    Dim target As RobotList = selectedRobot
    Dim name As String = variableNameTextBox.Text.Trim()
    readButton.Enabled = False
    Try
        valueTextBox.Text = Await sdk.ReadVariableAsync(target, name)
    Catch failure As Exception
        statusLabel.Text = failure.Message
    Finally
        readButton.Enabled = True
    End Try
End Sub
```

The fields in these handlers belong to your form. The examples are patterns for a form with an initialized session, selected target and named controls; they are not standalone console methods.

13.2 Run a synchronous transfer on a worker

C#

```
RobotList target = selectedRobot;
string source = localFileTextBox.Text.Trim();
string destination = remoteFileTextBox.Text.Trim();
await Task.Run(() =>
    sdk.UploadFileToRobot(target, source, destination));
```

VB.NET

```
Dim target As RobotList = selectedRobot
Dim source As String = localFileTextBox.Text.Trim()
Dim destination As String = remoteFileTextBox.Text.Trim()
Await Task.Run(Sub()
    sdk.UploadFileToRobot(target, source, destination)
End Sub)
```

Do not move textbox reads into the Task.Run lambda. The worker should use the captured strings and target. If several buttons can change the same interpreter, use a per-workflow SemaphoreSlim to serialize commands rather than letting them race.

13.3 Marshal a variable event to the UI

C#

```
sdk.VariableStatusChanged += (sender, e) =>
{
    string text = e.FullName + " = " + e.Value;
    if (IsDisposed || !IsHandleCreated) return;
    BeginInvoke(new Action(() =>
    {
        if (!IsDisposed) statusLabel.Text = text;
    }));
};
```

VB.NET

```
AddHandler sdk.VariableStatusChanged, Sub(sender, e)
    Dim text As String = e.FullName & " = " & e.Value
    If IsDisposed OrElse Not IsHandleCreated Then Return
    BeginInvoke(New Action(Sub()
        If Not IsDisposed Then statusLabel.Text = text
    End Sub))
End Sub
```

A form can close between checking its handle and calling BeginInvoke. Production UI code should also catch InvalidOperationException from that race and use a closing/session-replacement guard. The supplied sample implements those guards.

13.4 Use the supplied VB.NET sample

Open RAS.KRC4.WinFormsVB.sln and choose the VB.NET application as the startup project. The sample includes Controller, Variable, Variable Monitoring, File/Directory, Backup/Project, Program, Message and API Coverage tabs. The controller selected in the top list is used by the operation buttons.

It demonstrates every KRC method, both scan overloads, the two session constructors, KRCLoader and message formatting. It subscribes to all 14 events. Its API Coverage tab lists 43 usage entries. The sample does not connect automatically when opened, and modifying actions display the target for confirmation.

The sample stores manually entered controllers separately from discovered entries. Its UI remains responsive during service calls, and it waits for the active user operation before closing the session. It cannot forcibly cancel a controller request because the high-level API does not expose cancellation.

Chapter 14

Errors and Troubleshooting

14.1 Observe every result channel

Use try/catch around synchronous calls and around Await. Subscribe to OperationFailed for the event-based read and monitoring paths. Backup reports failure by an event and by an exception; use one application job record to avoid duplicate failure entries.

C#

```
try
{
    string text = await sdk.ReadVariableAsync(robot, "$MODE_OP");
    Console.WriteLine(text);
}
catch (Exception failure)
{
    Console.Error.WriteLine(failure.GetType().Name);
    Console.Error.WriteLine(failure.Message);
}
```

VB.NET

```
Try
    Dim text As String = Await sdk.ReadVariableAsync(robot, "$MODE_OP")
    Console.WriteLine(text)
Catch failure As Exception
    Console.Error.WriteLine(failure.GetType().Name)
    Console.Error.WriteLine(failure.Message)
End Try
```

14.2 Input and state errors

Exception category	Typical trigger	Application response
ArgumentNullException / ArgumentException	Missing target, empty variable/path argument, null write value or invalid local destination	Validate operator input and correct the request
ArgumentOutOfRangeException	Non-positive scan duration, line below 1, Unknown/undefined program mode	Enforce valid numeric or enum choices
InvalidOperationException	Unavailable variable, write outside AUT/EXT, interpreter state disallows command or missing interpreter	Refresh state/availability and display the rejected action
DirectoryNotFoundException / IOException	Missing source directory, file access failure or invalid transfer output	Check local and remote path spaces separately
TimeoutException / service communication fault	Endpoint not responding or expected interpreter transition not confirmed	Inspect target state before retrying
ObjectDisposedException	A closed session is reused	Create a new session; do not reuse the disposed object
AggregateException during cleanup	More than one resource close failure	Inspect contained exceptions and retain the shutdown log

Controller service faults may provide more specific text. Preserve the operation name, target address/ID, relevant path/variable and exception type/message in an application log. Do not put confidential cell data or credentials into a public diagnostic export.

14.3 Troubleshooting by symptom

Symptom	Check
Discovery returns no controllers	Probe completion budget, controller discovery availability, subnet/routing and known-address scan
A discovered controller cannot read variables	DataAccess endpoint, address resolution, access policy and variable availability
Read works but SetVariable fails	Current AUT/EXT mode, variable writability and supplied controller-compatible value
File operation fails while variable read works	FileHandling endpoint, exact remote path and controller file permissions
GetOpMode returns Unknown	The returned \$MODE_OP text is not one of the recognized tokens
GetVariableAsync appears to return nothing	Its immediate return is empty; observe VariableAsyncFinished or use ReadVariableAsync
Monitoring reports nothing	Add identifiers, start that controller watcher, keep the session alive and observe OperationFailed
Monitoring repeatedly fails	Persistent availability/access error or failed network service; inspect the exception before changing timing
Callback causes a cross-thread UI error	Dispatch the UI update with BeginInvoke and copy values before posting
Start/Select/Deselect is rejected	Actual robot interpreter state, selected program and controller authorization
ZIP exists but backup job failed	The active project part may have failed; combined success requires both requested outputs
Message view contains removed messages	Process removal/reset actions or refresh the runtime-message snapshot

14.4 Retry policy

Read-only operations are suitable candidates for a bounded retry after a transient connection problem. Modifying operations need an explicit reconciliation step. After a timed-out upload, inspect the target file; after a timed-out program request, read the interpreter state; after a failed tree transfer, inspect what was changed. Avoid automatic loops that repeat StartProgram, SetVariable or deletion until no exception is raised.

14.5 Endpoint configuration

An application can provide a servicesConfiguration section when commissioned endpoints differ from the defaults. Configuration belongs to the application's executable configuration file, not an arbitrary DLL-named configuration file.

Project / configuration XML

```
<configuration>
  <configSections>
    <section name="servicesConfiguration"
      type="RAS.Services.ServicesSectionHandler, KRCSDKServerLib" />
  </configSections>
  <servicesConfiguration>
    <services>
      <add name="DataAccess" protocol="net.tcp"
        port="49002" endpointname="DataAccess" />
      <add name="FileHandling" protocol="net.tcp"
        port="49010" endpointname="FileHandling" />
    </services>
  </servicesConfiguration>
</configuration>
```

Do not copy these settings to a site with different commissioned service names. The high-level client uses the controller TCP service protocols. It has no username/password constructor; do not assume it can satisfy an arbitrary custom authentication policy. Confirm the supported service access configuration with the administrator.

Chapter 15

.NET 8 Integration Profile

15.1 Availability and interpretation

Note: This chapter is a documentation-only integration specification. No .NET 8 SDK binary, adapter or local host is supplied or implemented as part of this manual. The delivered client and sample remain .NET Framework 4.8 applications. The templates below must not be described as tested .NET 8 controller connectivity.

A .NET 8 application needs a .NET 8-compatible SDK implementation or an explicitly supplied integration component before it can use the controller functions in this manual. Merely changing a project TargetFramework from net48 to net8.0 does not provide the required service runtime.

The application logic in Chapters 4-12 describes the intended controller workflows. A .NET 8 delivery must validate its own transport, data serialization, callbacks, file streaming and resource cleanup against those behaviors. Do not directly load the delivered full-framework client and assume that all WCF service operations will execute on .NET 8.

15.2 Windows Forms target templates

Use a Windows-specific target for a .NET 8 Windows Forms application. These project templates configure the application runtime only; an SDK reference is intentionally absent until a compatible integration component is delivered.

C# project template:

Project / configuration XML

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
  </PropertyGroup>
</Project>
```

VB.NET project template:

Project / configuration XML

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <RootNamespace>CustomerKrcApplication</RootNamespace>
    <OptionStrict>On</OptionStrict>
    <OptionExplicit>On</OptionExplicit>
  </PropertyGroup>
</Project>
```

For a non-UI library or console application, select the framework appropriate to the component's real platform dependencies. A generic net8.0 declaration does not make a Windows-specific controller integration cross-platform.

15.3 Application boundary example

The following interface is an application-defined boundary for planning a .NET 8 integration. It is not an exported type in KRCSDKServerLib.dll and has no controller-communication implementation here. Supply its implementation from the validated .NET 8 component before executing the workflow.

C# template:

C# - .NET 8 design template

```
public interface IKrc8VariableReader
{
    Task<string> ReadVariableAsync(
        string controllerAddress, string variableName);
}

public async Task<string> ReadModeAsync(
    IKrc8VariableReader reader, string address)
{
    return await reader.ReadVariableAsync(address, "$MODE_OP");
}
```

VB.NET template:

VB.NET - .NET 8 design template

```
Public Interface IKrc8VariableReader
    Function ReadVariableAsync(controllerAddress As String,
        variableName As String) As Task(Of String)
End Interface

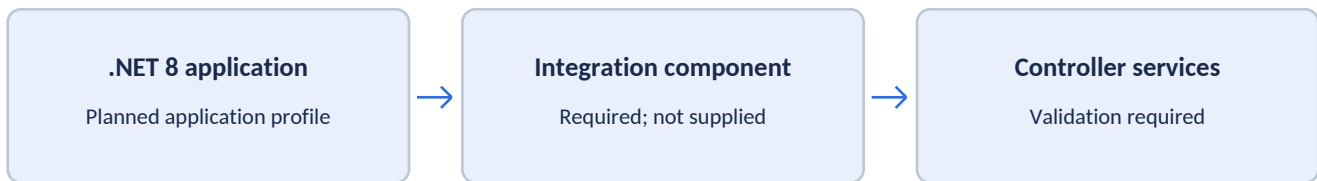
Public Async Function ReadModeAsync(reader As IKrc8VariableReader,
    address As String) As Task(Of String)
    Return Await reader.ReadVariableAsync(address, "$MODE_OP")
End Function
```

The application can prepare its async button handlers, target selection, logging and state model against this boundary. It cannot claim controller connectivity until a compatible implementation is available and commissioned.

15.4 Integration acceptance requirements

Requirement	What must be validated for a .NET 8 delivery
Controller operations	Every supported read/write, file, backup and interpreter operation has equivalent documented behavior
Data models	Robot identity, text values, message arrays, enum values and timestamps are preserved
Event routing	All required callbacks have a defined delivery context and target/correlation policy
Desktop use	Async work does not block the UI; posted callbacks respect form/session shutdown
Transfers	Streaming, overwrite rules, path boundaries and partial-failure outcomes are documented and tested
Lifecycle	Session disposal closes communication resources and prohibits later use
Deployment	The exact SDK/integration files and required runtime components are identified

An integration can be delivered as a native .NET 8 service client or as an explicitly packaged process boundary. Those are implementation choices, not features delivered by this document. The service stack and deployment requirements must be assessed before selecting either design.



15.5 Microsoft platform references

Microsoft documents Windows Forms project targeting with `net8.0-windows` and `UseWindowsForms`. Modern WCF Client is a separately supported component, distinct from the full-framework service runtime. See the primary references in Appendix C before choosing transport packages or making compatibility claims.

For deployment planning, Microsoft's published .NET 8 support end date is November 10, 2026. Review the applicable runtime lifecycle before committing an application deployment schedule; a target-framework template is not a product-support commitment.

Chapter 16

Complete Application Workflows

16.1 Read-only session with error handling

This workflow explicitly receives a controller address. It reads mode and position, reports the result, and closes the session. It neither uploads files nor starts a program.

C#

```
public static async Task ReadStatusAsync(string address)
{
    using (var sdk = new KRC())
    {
        var robot = new RobotList
        {
            BaseAddress = address,
            ID = string.Empty,
            Name = "Selected controller"
        };
        try
        {
            string mode = await sdk.ReadVariableAsync(robot, "$MODE_OP");
            string position = await sdk.ReadVariableAsync(robot, "$POS_ACT");
            Console.WriteLine("Mode: " + mode);
            Console.WriteLine("Position: " + position);
        }
        catch (Exception failure)
        {
            Console.Error.WriteLine(address + ": " + failure.Message);
        }
    }
}
```

VB.NET

```
Public Shared Async Function ReadStatusAsync(address As String) As Task
    Using sdk As New KRC()
        Dim robot As New RobotList With {
            .BaseAddress = address,
            .ID = String.Empty,
            .Name = "Selected controller"
        }
        Try
            Dim mode = Await sdk.ReadVariableAsync(robot, "$MODE_OP")
            Dim position = Await sdk.ReadVariableAsync(robot, "$POS_ACT")
            Console.WriteLine("Mode: " & mode)
            Console.WriteLine("Position: " & position)
        Catch failure As Exception
            Console.Error.WriteLine(address & ": " & failure.Message)
        End Try
    End Using
End Function
```

16.2 Wait for known-address discovery

Subscribe before scanning and detach the handler when the workflow completes or times out. This timeout limits application waiting only; disposal or subsequent session management governs the remaining discovery lifetime.

C#

```
public static async Task<RobotList[]> DiscoverAsync(KRC sdk,
    string address)
{
    var ready = new TaskCompletionSource<RobotList[]>(
        TaskCreationOptions.RunContinuationsAsynchronously);
    KRC.ControllersAvailableEventEventHandler handler = () =>
        ready.TrySetResult(sdk.GetControllers());
    sdk.ControllersAvailableEvent += handler;
    try
    {
        sdk.ScanControllers(new[] { address }, 5);
        Task done = await Task.WhenAny(ready.Task, Task.Delay(30000));
        if (done != ready.Task)
            throw new TimeoutException("Discovery did not finish in time.");
        return await ready.Task;
    }
    finally
    {
        sdk.ControllersAvailableEvent -= handler;
    }
}
```

VB.NET

```
Public Shared Async Function DiscoverAsync(sdk As KRC,
    address As String) As Task(Of RobotList())
    Dim ready As New TaskCompletionSource(Of RobotList())(
        TaskCreationOptions.RunContinuationsAsynchronously)
    Dim handler As KRC.ControllersAvailableEventEventHandler =
        Sub() ready.TrySetResult(sdk.GetControllers())
    AddHandler sdk.ControllersAvailableEvent, handler
    Try
        sdk.ScanControllers(New String() {address}, 5)
        Dim done As Task = Await Task.WhenAny(ready.Task, Task.Delay(30000))
        If done IsNot ready.Task Then
            Throw New TimeoutException("Discovery did not finish in time.")
        End If
        Return Await ready.Task
    Finally
        RemoveHandler sdk.ControllersAvailableEvent, handler
    End Try
End Function
```

16.3 Download program data to a new local folder

This example checks the remote source and copies a directory tree. The chosen local folder should identify the run so that earlier results are retained.

C#

```
public static void DownloadJob(KRC sdk, RobotList robot)
{
    string remote = @"KRC:\R1\Program\CustomerJobs";
    string local = Path.Combine(@"C:\RAS\Downloads",
        DateTime.Now.ToString("yyyyMMdd_HHmss"));
    if (!sdk.DirectoryExist(robot, remote))
        throw new DirectoryNotFoundException(remote);
    sdk.DownloadDirectoryToPC(robot, remote, local);
    Console.WriteLine("Downloaded to " + local);
}
```

VB.NET

```
Public Shared Sub DownloadJob(sdk As KRC, robot As RobotList)
    Dim remote As String = "KRC:\R1\Program\CustomerJobs"
    Dim local As String = Path.Combine("C:\RAS\Downloads",
        DateTime.Now.ToString("yyyyMMdd_HHmss"))
    If Not sdk.DirectoryExist(robot, remote) Then
        Throw New DirectoryNotFoundException(remote)
    End If
    sdk.DownloadDirectoryToPC(robot, remote, local)
    Console.WriteLine("Downloaded to " & local)
End Sub
```

16.4 Prepare a program without automatically starting it

Uploading and selecting are separate steps. The example only prepares a controller-resident program. A subsequent start action should be a separate operator-authorized workflow; do not hide StartProgram inside an automatic upload-completion handler.

C#

```
public static void PrepareJob(KRC sdk, RobotList robot)
{
    string target = @"KRC:\R1\Program\CustomerJobs";
    sdk.UploadDirectoryToRobot(robot, @"C:\RAS\ApprovedJob", target);
    if (sdk.GetProgramState(robot) != ProgramStates.Free)
        throw new InvalidOperationException("Interpreter is not free.");
    sdk.SelectProgram(robot, target + @"\CustomerJob.src");
    Console.WriteLine("Program selected; start was not requested.");
}
```

VB.NET

```
Public Shared Sub PrepareJob(sdk As KRC, robot As RobotList)
    Dim target As String = "KRC:\R1\Program\CustomerJobs"
    sdk.UploadDirectoryToRobot(robot, "C:\RAS\ApprovedJob", target)
    If sdk.GetProgramState(robot) <> ProgramStates.Free Then
        Throw New InvalidOperationException("Interpreter is not free.")
    End If
    sdk.SelectProgram(robot, target & "\CustomerJob.src")
    Console.WriteLine("Program selected; start was not requested.")
End Sub
```

Keep paired program data files and required referenced files in the approved upload set when the selected program depends on them. A successful upload is not a syntax or dependency validation of the program.

16.5 Backup with a caller job identifier

The caller logs the robot and output path and passes a correlation number. In a UI, run this synchronous workflow on a worker. If either requested output fails, retain the exception and inspect partial outputs before scheduling another backup.

C#

```
public static void BackupController(KRC sdk, RobotList robot, int jobId)
{
    string folder = Path.Combine(@"C:\RAS\Backups",
        "Job_" + jobId.ToString(CultureInfo.InvariantCulture));
    Console.WriteLine(robot.BaseAddress + " -> " + folder);
    sdk.Backup(robot, folder, jobId, true, true);
    Console.WriteLine("Archive and active project completed.");
}
```

VB.NET

```
Public Shared Sub BackupController(sdk As KRC,
    robot As RobotList, jobId As Integer)
    Dim folder As String = Path.Combine("C:\RAS\Backups",
        "Job_" & jobId.ToString(CultureInfo.InvariantCulture))
    Console.WriteLine(robot.BaseAddress & " -> " & folder)
    sdk.Backup(robot, folder, jobId, True, True)
    Console.WriteLine("Archive and active project completed.")
End Sub
```

Chapter 17

Deployment and Commissioning

17.1 Prepare the application package

Build the application for its documented runtime. Include the application executable, configuration file and SDK DLL. Verify that the deployment folder is complete and that the workstation runtime matches the target framework. Treat the .NET 8 profile in Chapter 15 as an integration specification, not as a deployable binary in this package.

Use configuration or operator input for controller addresses rather than compiling a production controller address into a shared test tool. Keep local data output folders writable by the deployed account. No administrative elevation is required merely by the API design; the workstation's folder/network policies can impose additional site requirements.

17.2 Commission in stages

1. Confirm the commissioned controller address and the services needed by the workflow.
2. Open a session and perform a read-only \$MODE_OP read.
3. Read metadata/discovery results and confirm the intended target identity.
4. Test one approved variable watch and verify both values and the failure channel.
5. List an approved remote directory and download a non-critical test file to a new local folder.
6. Verify backup output in a dedicated commissioning folder if archive/project services are required.
7. Only then commission approved upload, deletion or interpreter actions under the site's operating procedure.
8. Close the session and confirm that the application does not continue to post live callbacks into a closed UI.

Each stage validates a different service path. A status read does not prove that a backup or interpreter command is authorized.

17.3 Application logging

For each operation, record a job/correlation identifier, controller address/ID, operation name, variable or path, start/end receipt times and success/failure. For backup jobs, log both requested options. For message data, retain controller timestamps separately from PC receipt timestamps when required for correlation.

Bound live UI logs so that repeated monitoring errors do not exhaust memory. Avoid treating a network exception as a empty collection or a default scalar value. A production log should distinguish an operator cancellation before sending a request from a service failure after a request was sent.

17.4 Verification boundary

The documented client behaviors and examples have local validation against the KRC4 service model; KRC5 controllers are confirmed during commissioning (Chapter 18). Controller discovery, controller-side access policy, file transfer, backup output, message callbacks and physical cell workflows still require commissioning against the intended controller installation. No hard real-time update period, fixed transfer throughput or universal firmware-version compatibility is promised by this manual.

Chapter 18

KRC5 Controllers

18.1 KRC4 and KRC5 at a glance

The KUKA KRC4 & KRC5 SDK is offered for both controller generations. Your application uses the same session, controller descriptor and event model for either one, and a single application can work with KRC4 and KRC5 controllers side by side.

	KUKA KRC4	KUKA KRC5
Architecture	Modular and flexible architecture; high precision motion control	More compact and powerful design
Energy	Intelligent power management for energy efficiency	Optimized energy consumption
Communication	Wide range of connection options supporting various communication protocols	Advanced communication and connection options
Software and integration	Windows-based system with wide software compatibility	High level of compatibility for Industry 4.0 and IoT integration

18.2 Network access to a KRC5

No options or installation are required on the robot. The PC reaches the controller directly over the network through the KLI and KONI ports, in the same way as for a KRC4. Use the address supplied by the commissioning administrator as the BaseAddress of the RobotList descriptor, exactly as described in Chapter 4.

As with every controller, the services required by a workflow must be reachable. IP reachability alone does not prove that variable access, file handling, runtime management and project services are available.

Note: The default service ports, variable names, controller path examples and behaviors in Chapters 2-12 were documented for KRC4 installations. Before relying on any of them for a KRC5, confirm them against that installation using the checklist in section 18.3.

18.3 KRC5 commissioning checklist

Commission a KRC5 in the same stages as a KRC4 (section 17.2). Each stage validates a different service path, so a successful status read does not prove that files, backups or interpreter commands are available.

Item	What to confirm for the KRC5
Address and network	Controller address, and that the PC reaches it through the KLI or KONI port

Item	What to confirm for the KRC5
Service endpoints	Which services the controller exposes and on which ports; compare with section 2.3 and, when they differ, configure them as described in section 14.5
Variable access	A read-only \$MODE_OP read succeeds, and every variable your application uses exists and is accessible
Operating modes	The returned \$MODE_OP text matches the tokens in section 6.3, so that GetOpMode does not return Unknown
File services	The controller-side path style used by your workflow; list a directory and download a non-critical test file first
Backup and project	Archive ZIP and active-project download produce the expected output in a dedicated folder
Interpreter	State, mode and message reads work before any select, start, stop, reset or line operation is enabled
Software versions	The controller software version in use is recorded in the commissioning log

18.4 Working with KRC4 and KRC5 together

One session can own a watch list and a monitor for each controller. The example below watches the operating mode of one KRC4 and one KRC5. Record which RobotIndex belongs to which controller when you configure the watchers, as explained in Chapter 4.

C#

```
var krc4 = new RobotList { BaseAddress = "192.168.0.10", ID = string.Empty,
    Name = "KRC4 Cell A" };
var krc5 = new RobotList { BaseAddress = "192.168.0.20", ID = string.Empty,
    Name = "KRC5 Cell B" };

sdk.VariableStatusChanged += (sender, e) =>
    Console.WriteLine($"Robot {e.RobotIndex}: {e.FullName} = {e.Value}");
sdk.OperationFailed += (sender, failure) =>
    Console.Error.WriteLine(failure.Message);

foreach (RobotList robot in new[] { krc4, krc5 })
{
    sdk.AddVariableWatch(robot, "$MODE_OP");
    sdk.StartVariableMonitor(robot);
}
```

VB.NET

```
Dim krc4 As New RobotList With {.BaseAddress = "192.168.0.10", .ID = String.Empty,
    .Name = "KRC4 Cell A"}
Dim krc5 As New RobotList With {.BaseAddress = "192.168.0.20", .ID = String.Empty,
    .Name = "KRC5 Cell B"}

AddHandler sdk.VariableStatusChanged, Sub(sender, e)
    Console.WriteLine($"Robot {e.RobotIndex}: {e.FullName} = {e.Value}")
End Sub
AddHandler sdk.OperationFailed, Sub(sender, failure)
    Console.Error.WriteLine(failure.Message)
End Sub

For Each robot As RobotList In New RobotList() {krc4, krc5}
    sdk.AddVariableWatch(robot, "$MODE_OP")
    sdk.StartVariableMonitor(robot)
Next
```

Monitoring is software polling and not a hard real-time sampling guarantee, on either controller generation. Stop each monitor with `StopVariableMonitor` when the live display is no longer needed.

18.5 Typical application areas

- Automation in production lines.
- Smart factories and Industry 4.0 integration.
- IoT-based data collection and analysis.
- Integration with PLC and SCADA systems.

18.6 Scope for KRC5

The scope in section 1.4 applies equally to KRC5 controllers: the SDK is a Windows client library, not a hard real-time system, and it does not replace cell safety functions, emergency-stop circuits or controller authorization. Controller services, access policy and software versions can vary from site to site, and no universal firmware-version compatibility is promised. RAS Otomasyon confirms compatibility together with you during commissioning.

Appendix A

Data Model Reference

A.1 RobotList properties

Property	Type	Interpretation
Index	Int32	Snapshot position / watcher label; not a stable network identity
BaseAddress	String	Controller host or IP address
ID	String	Controller identity when supplied
Name	String	Friendly controller name
HostName	String	Reported hostname
CellName	String	Reported cell name
CellDescription	String	Cell description
CellMembers	String[]	Reported cell membership
CurrentProjectName	String	Reported active/current project name
CurrentProjectDescription	String	Project description
Description	String	Device description
Manufacturer	String	Reported manufacturer metadata
Type	String	Reported device type
Version	String	Reported device version text
FirmwareType	String	Firmware type text
FirmwareVersion	String	Firmware version text
FirmwareVersionAppendix	String	Additional firmware version text
RoboterType	String	Reported robot type
RoboterAxesCount	Int32	Reported robot-axis count
ExternalAxesCount	Int32	Reported external-axis count
Serialnumber	String	Reported serial number

Property names use the exact API spelling, including `Roboter`, `Serialnumber` and `ID`. Reported text and arrays can be missing; a manually created descriptor normally only supplies addressing and display fields. Do not reject a reachable controller solely because nonessential description metadata is empty.

A.2 VariableEventArgs

Property	Type	Interpretation
Name	String	Variable identifier name
FullName	String	Full displayed identifier
Value	String	Current value text
RobotIndex	Int32	Target index captured when the watcher was created

A.3 VariableAsyncEventArgs

Property	Type	Interpretation
Name	String	Variable name for the completed read
Value	String	Read value text
TimeStamp	DateTime	Timestamp associated with the controller value

No robot index is present in this event payload. Use a request's awaited Task or a separate session when per-controller correlation is required.

A.4 RuntimeDisplayMessage

Member	Type	Interpretation
Count	Int32	Number of entries in the snapshot
Id	Int32[]	Entry IDs
IsVisible	Boolean[]	Visibility flags
MessageCode	String[]	Codes
MessageNumber	Int32[]	Numbers
Text	String[]	Message text
Type	Object[]	Message types as exposed by the service snapshot
TimeStamp	DateTime[]	Entry timestamps
MessageToString(Index)	String-returning method	Format one entry without another controller read

Every parallel array uses the same valid entry index. Loop using Count, not the length of an independently selected array. Empty snapshots have no valid index zero.

A.5 MessagesChangedEventArgs

Property	Type	Interpretation
Action	Nested RuntimeDisplayMessageChangeType enum	Add, Remove, Replace, Move or Reset
NewItems	IList of String	New entries in the change
OldItems	IList of String	Old/removed entries in the change
NewStartingIndex	Int32	New-side collection index reported by the service
OldStartingIndex	Int32	Old-side collection index reported by the service

The nested change-action type on MessagesChangedEventArgs is the type of its Action property. A same-named top-level enum also exists; do not assume that two enum types are interchangeable merely because their member names match. ToString() formats the collection metadata.

A.6 Enumerations

Type	Defined values in declaration order
OpMode	ExternalAutomatic, TestReducedSpeed, TestFullSpeed, Automatic, Unknown
ProgramModes	Unknown, MStep, IStep, Go, BStep
ProgramStates	Unknown, Free, Stop, Reset, Active, End
RuntimeDisplayMessageChangeType	Add, Remove, Replace, Move, Reset

Use named enum values in application code. These are ordinary enums, not bit flags. In VB.NET, escape keyword-like enum members when necessary, for example ProgramStates.[Stop] and ProgramStates.[End].

Appendix B

API Index and Example Conventions

B.1 Reading signatures

The API reference uses C# signatures to show the declared return types, parameter names and defaults. Each item also contains a VB.NET usage example. `string` maps to `String`, `int` to `Integer`, `bool` to `Boolean`, `RobotList[]` to `RobotList()` and `Task<string>` to `Task(Of String)`.

C# named arguments use `name: value`; VB.NET named arguments use `name:=value`. Optional flags shown in signatures can be omitted when their defaults are appropriate. The direct method spelling is significant: the high-level API uses `FileExist` and `DirectoryExist`.

API usage fragments assume a live KRC `sdk / sdk As KRC` and a valid `RobotList robot / robot As RobotList`. Callback fragments demonstrate subscription; create them once for the owning workflow and keep the session alive. The method examples that upload, write, delete or run programs are not an automatic test sequence.

B.2 Complete application-facing index

Entry	Result	Section
KRC()	KRC session	4.4
KRC(Boolean)	KRC session	4.5
KRCLoader.LoadLibrary	void	4.6
Dispose	void	4.7
ScanControllers()	void	5.3
ScanControllers(addresses, scanSeconds)	void	5.4
CheckScanning	bool	5.5
GetControllers	RobotList[]	5.6
GetVariable	string	6.4
ReadVariableAsync	Task<string>	6.5
GetVariableAsync	string	6.6
GetOpMode	OpMode	6.7
SetVariable	void	6.8
AddVariableWatch	void	7.4
RemoveVariableWatch	void	7.5
StartVariableMonitor	void	7.6
StopVariableMonitor	void	7.7
FileExist	bool	8.4
DirectoryExist	bool	8.5
GetFilesList	ICollection<string>	8.6
GetDirectoriesList	ICollection<string>	8.7
CreateDirectoryOnRobot	bool	8.8
DownloadFileToPC	void	8.9
UploadFileToRobot	void	8.10
DownloadDirectoryToPC	void	8.11
UploadDirectoryToRobot	void	8.12
DeleteFileFromRobot	void	8.13
DeleteDirectoryFromRobot	void	8.14
GetProject	void	9.4

Entry	Result	Section
Backup	void	9.5
GetProgramState	ProgramStates	10.4
GetProgramMode	ProgramModes	10.5
SetProgramMode	void	10.6
SelectProgram	void	10.7
DeselectProgram	void	10.8
StartProgram	void	10.9
StopProgram	void	10.10
ResetProgram	void	10.11
SetLine	void	10.12
ReadLogMessages	RuntimeDisplayMessage	11.4
EnableMessageEvents	void	11.5
RuntimeDisplayMessage.MessageToString	string	11.6
MessagesChangedEventArgs.ToString	string	11.7

Appendix C

Technical Platform References

The operation reference in this manual describes the delivered application-facing API. The following primary platform references support runtime targeting, Windows Forms configuration and the .NET 8 integration planning notes. They do not imply that a .NET 8 controller transport is delivered here.

1. [Microsoft desktop SDK project properties](#) - Windows-specific target frameworks and the UseWindowsForms project property.
2. [Microsoft WCF Client support policy](#) - Support policy for the modern WCF Client component.
3. [Microsoft .NET support policy](#) - Runtime lifecycle and servicing information.
4. [Microsoft .NET 8 and .NET 9 support-end announcement](#) - Published November 10, 2026 support end date.

Platform references were checked for the October 2026 edition of this manual. Validate the currently applicable runtime and product support policy when commissioning a deployment.