ETAS

DRIVING EMBEDDED EXCELLENCE

INCA-MIP V7.3

User's Guide

www.etas.com

Copyright

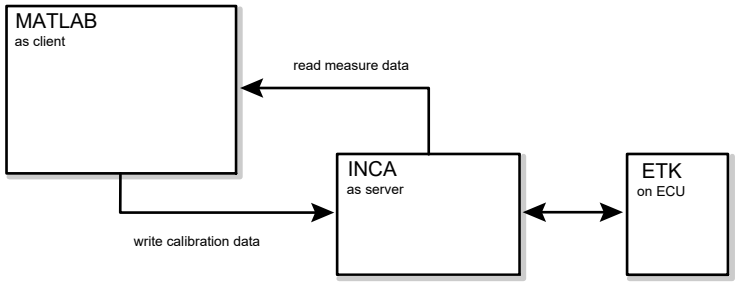
INCA-MIP V7.3 - User's Guide R02 EN - 06.2020

Content

1 Introduction

The INCA-MIP Add-On (INCA MATLAB Integration Package) is an application programming interface that controls INCA's functionality from within MATLAB. Within this context, MATLAB acts as a client accessing INCA's resources, which in this case is the server.

The following chart illustrates a typical application for the INCA-MIP API, using INCA to address an ETK.



The following overview outlines the INCA functionality that can be accessed from within MATLAB.

Memory Page Management

Switching back and forth between memory pages and downloading memory pages to the control unit are supported.

Calibrating

All the calibration variables for an INCA experiment can be modified. The values can be read and updated for each element and for the associated break point distributions, where applicable.

Measuring

All measurement variables from an INCA experiment can be read. In addition, measurements can be started and stopped from within MATLAB. All performance data that are available in INCA can also be accessed from MATLAB. The performance data throughput at the INCA-MIP interface has been optimized.

Since INCA generates measurement and calibration variables as doubles, conversion formulas for reprocessing in MATLAB are not required.

The INCA API functions described in this document are invoked from MATLAB scripts (so-called M files), which can be used to define the entire control flow for INCA experiments.

1.1 Safety Information

WARNING

Calibration activities influence the behavior of the ECU and the systems controlled by the ECU. This may result in unexpected behavior of the vehicle and thus can lead to safety critical situations.
Only well trained personnel should be allowed to perform calibration activities.

1.2 About this Manual

The following sections describe the underlying architecture for the INCA-MIP API and the available API functions as well as the installation procedures. MATLAB or INCA operation falls outside the scope of this manual.

To use the INCA-MIP API, you should be familiar with both INCA and MATLAB. You should also be familiar with using scripts in MATLAB.

1.3 Typographic Conventions

This manual uses the following conventions to describe the API functions in addition to the usual typographic conventions found in the INCA User Manual:

The name of the API function is written in a non-proportional font. Parentheses are used for the functions argument; braces for the optional input arguments.

Example: Reading a Calibration Variable

```
value = IncaGetCalibrationValue(deviceName, cal-
ibrationName {, start, size} {, valueType})
```

Output Arguments:

`value`	Current value of the calibration variable; it must match the data types specified below: • Scalars: a (1,1) matrix • Curves: an (x,1) matrix • Maps: an (x,y) matrix • Break point distributions: an (x,1)-matrix

Input Arguments:

`deviceName`	name of the device
`calibrationName`	the name of the calibration element
`valueType`	Selection of the output argument (string). The function either returns the value of the calibration variable (default) or the X- and Y-break point distribution. Possible settings: • `v`: value • `x`: x break point (curves and maps) • `y`: y break point (maps)

The values `v`, `x` and `y` in `non-proportional font` represent the possible values that can be generated. These values have to be enclosed in "".

Example:

```
yMap = IncaGetCalibrationValue('anEtk', 'Map', 'y');
```

1.4 INCA Glossary

The API description uses certain technical terms that experienced INCA users should be familiar with. Below is a brief definition of these terms.

Calibration Variable

A calibration variable is an element that can be read and modified. Calibration variables can be scalars, vectors, matrices, curves and maps. The associated break point distributions can also be read and modified.

Data record

A record consists of a time stamp and all the measurement values in a signal group for a single acquisition. The measurement data for a signal group consists of several records that are generated throughout the entire measurement process.

Device

A measuring device used for capturing measurement variables within a certain measurement grid. Some measuring devices also support calibration for corresponding variables. For example, SMB devices can be used for measurements only, while the ETK is suitable both for measurements and calibrations.

Measure Data

All the records captured in one measurement for individual measure rasters.

Measurement Raster

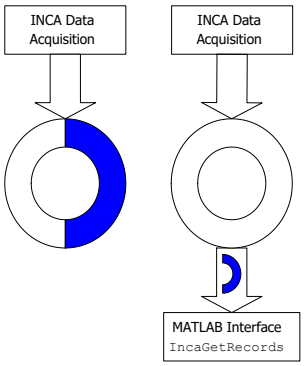
Acquisition rate (measuring frequency) used for measuring one or more signals in a signal group.

It is possible to combine two or more rasters in a so-called multi-raster. This is done by simply combining the raster names by means of a '+' character, e.g. '`10ms+100ms`'. When using such a multi-raster, a new virtual raster is created. Each signal can only be measured in exactly one raster or multi-raster.

Ring Buffer

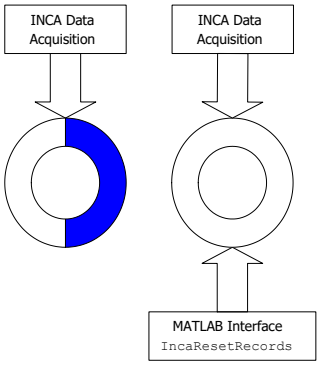
To ensure a reliable transfer of measurement data from INCA to MATLAB, a dedicated ring buffer is used for each measurement raster (signal group). During an INCA online visualization or recording the acquired measurement data is automatically saved in the ring buffer.

The command `IncaGetRecords` can be used to read the time stamps and data from the ring buffer into MATLAB:



The ring buffer is limited to 1 MByte per signal and can hold up to approximately 8 seconds of measurement data depending on the measurement rate. After this time old measurement data will get overwritten. To avoid losing data `IncaGetRecords` periodically has to be executed. Typically this will be done about once each second.

With the command `IncaResetRecords` the time stamps and data in the ring buffer for all measurement rasters can be cleared. All data already saved in the ring buffer will get lost:



Signal

A signal is an element whose value is measured in INCA. Each signal is characterized by its data type (Boolean, integer, float), length (1, 2, 4 or 8 bytes) and conversion formula. The conversion from the physical measure value on the implementation level is specified in the conversion formula.

Signal Group

A signal group consists of several individual signals. It is characterized by its measurement raster, which is the same for all signals in the signal group. Each signal group has a unique name.

2 Installing the INCA-MIP Add-on

INCA-MIP is a functional extension of INCA.

MATLAB uses dynamically linked function calls, so-called MEX files, to communicate with other applications. The INCA-MIP API consists of a collection of MEX files that are copied into the associated sub-directories of your MATLAB program directory during installation.

INCA-MIP is packaged in two versions. The INCA-MIP Base API set is readily available after the installation. In order to use the INCA-MIP Extended API set a software license key is required. A list of APIs and the respective API set can be found in "API Functions" on page 15.

2.1 System Requirements

To use the INCA-MIP add-on, INCA must be installed on your computer. For further information on INCA system requirements, refer to the INCA Installation Guide.

If you would like to develop MATLAB scripts yourself for accessing INCA, you also need a full MATLAB license.

INCA-MIP for INCA V7.3 requires the following program releases:

- INCA V7.3 SPx

  Note

  INCA V7.3 is required for the installation of this INCA-MIP version.

  Make sure that the INCA release number of the INCA installation is compatible with the release number of the INCA-MIP add-on package.

  After installation you can use this INCA-MIP version to work with any INCA V7.x version "Open INCA" on page 26.

- MATLAB 64 bit version 2015b or later (for MATLAB integrated installation)

For further information on supported MATLAB releases, contact your INCA support.

2.2 Installing INCA-MIP

Before installing the add-on it is necessary to determine the type of installation. The following types are possible:

- **MATLAB integrated installation**
  Select this option if you use one MATLAB version for developing MATLAB scripts.

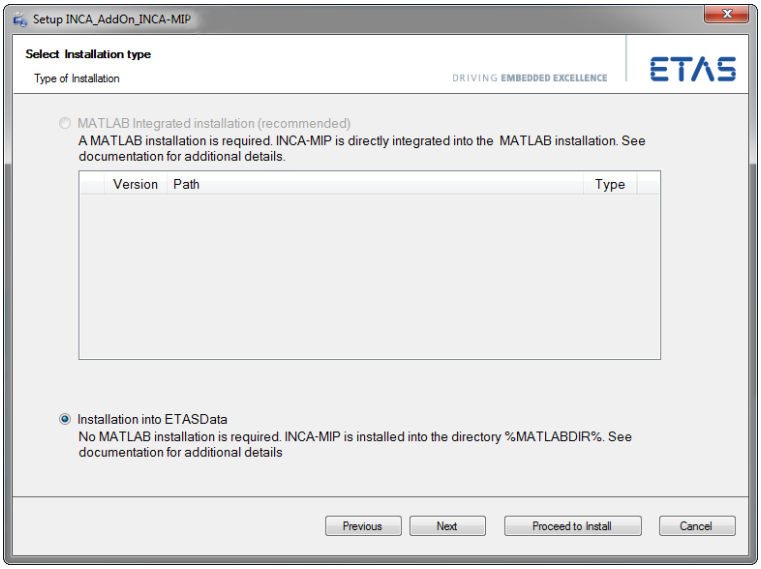
- **Installation into ETASData**
  Select this option if you would just like to run compiled MATLAB scripts or if you would like to use INCA-MIP with different MATLAB versions on your PC. For a more detailed description see below.

To install INCA-MIP

Make sure that INCA is installed on your computer and that the release number of the INCA installation is compatible with the release number of the INCA-MIP add-on package.

If you would like to develop your own MATLAB scripts for accessing INCA, make sure that MATLAB is installed on your computer and that the release number of the MATLAB installation is compatible with the release number of the INCA-MIP add-on package.

1. Close all active programs.
2. Depending on your company-specific regulations, the installation files are provided on a network drive or on a DVD. By using the DVD, the installation routine starts automatically. If this is not the case, execute the `Autostart.exe` file on the DVD manually. If you install the program from a network drive, also execute the `Autostart.exe` file.
3. Click **Main**.
4. Select the INCA-MIP installation.
5. Follow the instructions in the installation routine to install INCA-MIP on your computer.
6. In the installation routine, you are asked to indicate the desired type of installation:



7. If you like to develop MATLAB scripts with exactly one MATLAB

version installed on your PC select the option **MATLAB integrated installation**.

or

Select the option **Installation into ETASData** if one of the following cases applies:

- You want to use INCA-MIP with different MATLAB versions.

  In this case you must add the INCA-MIP subdirectory to the MATLAB toolbox directory of each MATLAB installation on your PC before you can use INCA-MIP commands. See your MATLAB user documentation on how to add directories to the MATLAB path.

- You only want to run readily available MATLAB stand-alone executables created with MATLAB R13.

  Note

  You need the installation into ETASData if you use executable files that contain MATLAB commands for controlling INCA. In this case you do not need a MATLAB license. The executable files have to be provided by developers with a MATLAB installation (see "Creation and Distribution of Stand-alone Executable Files using the MATLAB R13 Compiler" on page 67).

8. Continue with the installation routine.

To license INCA-MIP

Should you plan to use the extended set of API functions, a software license file will be required.

For further information on licensing, refer to "Licensing the Software" on the next page.

2.3 Updating the Cache for MATLAB Toolbox directories

After installing the INCA-MIP API, you should first update the cache for the MATLAB toolbox directories, in case this cache is enabled during your MATLAB installation. This is true for MATLAB V6 and higher if you are using the default settings; the cache was disabled in earlier releases. The cache needs to be updated so that the files used in INCA-MIP API are registered in MATLAB.

See your MATLAB user documentation to update the cache for the MATLAB toolbox directories.

2.4 Disabling the Cache for MATLAB Toolbox Directories

When working with the INCA-MIP API, it is recommended that you disable caching for the MATLAB toolbox directories. Otherwise, malfunctions may occur because either the INCA-MIP API or individual, newly added script files may not be found.

As an alternative to disabling the caching, you can force the cache to update as described above while the cache is enabled. However, to avoid any faulty operation, it is recommended that you disable the cache while working with the INCA-MIP API.

See your MATLAB user documentation for enabling or disabling the cache for the MATLAB toolbox directories.

2.5 Licensing the Software

A valid license is required for using INCA. You can obtain the license file required for licensing either from your tool coordinator or through a self service portal on the ETAS Internet Site under https://www.etas.com/support/licensing. To request the license file you have to enter the activation number which you received from ETAS during the ordering process.

In the Windows Start menu, select

E → **ETAS** → **ETAS License Manager**.

Follow the instructions given in the dialog. For further information about, for example, the ETAS license models and borrowing a license, press **F1** in the ETAS License Manager.

3 API Functions

INCA-MIP provides a number of API functions for automating INCA processes. Some functions are available in the INCA-MIP Base package, others can be used only if you have purchased the INCA-MIP Extended package.

Note

Commands which are available only through INCA-MIP Extended are protected by a software license key. Should you use an Extended API function without a valid software license, MATLAB script execution will throw an exception.
As a development guideline we recommend that prior to using INCA-MIP Extended commands, you verify the validity of the license by means of the `IncaIsLicenseValid` command.

The following table lists all INCA-MIP API functions which are available in the add-on for [[[Undefined variable FM_import.Product_Version]]]. It indicates

- whether the function is also available in the INCA-MIP Base package or only in the INCA-MIP Extended package;
- whether the function is used for initialization, measuring, calibrating, memory page management or whether it is a more general function;
- where in this document you can find more information about the corresponding function.

Function	Base[a]	Ext.[b]	Category	refer to
IncaAddCalibrationElement	x	x	Calibration	on page 54
IncaAddMeasureElement	x	x	Measuring	on page 34
IncaBrowseCalibrationElements		x	Calibration	on page 53
IncaBrowseItemsInFolder		x	Initialization	on page 28
IncaBrowseMeasureElements		x	Measuring	on page 33
IncaClose		x	Initialization	on page 26
IncaCopyPageFromTo	x	x	MemoryPage-Manager	on page 64

IncaDatabaseImport		x	Initialization	on page 27
IncaDownloadDifferences	x	x	MemoryPage-Manager	on page 64
IncaDownloadPage	x	x	MemoryPage-Manager	on page 63
IncaExecuteManualTrigger		x	Measuring	on page 51
IncaGetCalibrationValue	x	x	Calibration	on page 55
IncaGetCurrentPage		x	MemoryPage-Manager	on page 63
IncaGetDatasetsForDevice		x	Calibration	on page 60
IncaGetDeviceProperties		x	Initialization	on page 31
IncaGetDevices		x	Initialization	on page 31
IncaGetHardwareStatus		x	Measuring	on page 48
IncaGetInstalledAddOnInfo	x	x	General	on page 24
IncaGetInstalledProductInfo	x	x	General	on page 23
IncaGetMeasureRatesForDevice		x	Measuring	on page 33
IncaGetProperties		x	General	on page 25
IncaGetRecordingMode		x	Measuring	on page 41
IncaGetRecordingProperties		x	Measuring	on page 36
IncaGetRecordingState		x	Measuring	on page 52
IncaGetRecords	x	x	Measuring	on page 45
IncaGetRecordStruct		x	Measuring	on page 52

IncaGetVersion	x	x	General	on page 24
IncaGroupDevices		x	Calibration	on page 61
IncaIsLicenseValid		x	General	on page 23
IncaMessageIds	x	x	General	on page 20
IncaIsPageWriteProtected	x	x	Memory-PageManager	on page 63
IncaOpen	x	x	Initialization	on page 26
IncaOpenDatabase	x	x	Initialization	on page 27
IncaOpenExperiment	x	x	Initialization	on page 30
IncaResetExperiment	x	x	Initialization	on page 31
IncaResetRecords	x	x	Measuring	on page 48
IncaSetCalibrationMode		x	Calibration	on page 60
IncaSetCalibrationValue	x	x	Calibration	on page 57
IncaSetDatasetInDevice		x	Calibration	on page 59
IncaSetMeasureReadMode	x	x	Measuring	on page 44
IncaSetProjectAndDatasetInDevice		x	Initialization	on page 29
IncaSetRecordingMode		x	Measuring	on page 42
IncaSetRecordingProperties		x	Measuring	on page 38
IncaSetTrigger		x	Measuring	on page 49
IncaShowMessages	x	x	General	on page 22

IncaStartMeasurement	x	x	Measuring	on page 35
IncaStartRecording	x	x	Measuring	on page 43
IncaStopMeasurement	x	x	Measuring	on page 36
IncaStopRecording	x	x	Measuring	on page 44
IncaSwitchPage	x	x	MemoryPage-Manager	on page 62
IncaUploadPages		x	MemoryPage-Manager	on page 64
IncaWriteToFile		x	Calibration	on page 61

a Function is supported in INCA-MIP Base Package
b Function is supported in INCA-MIP Extended Package

In this manual, the function descriptions are ordered according to their application area:

- "General Functions" on page 20
- "Initialization" on page 25
- "Measuring and Recording" on page 32
- "Calibrating" on page 53
- "Memory Page Manager" on page 62

Moreover, a number of sample files are provided. These are described under "Getting to know the INCA-MIP API through Sample Files" below.

Application examples are given under "Application Examples" on page 65.

Note

The INCA-MIP Interface always operates on the global settings of the INCA User Options. For further information on INCA User Options please see the INCA Documentation.

3.1 Getting to know the INCA-MIP API through Sample Files

INCA-MIP comes with a group of examples. These sample files are automatically installed on your computer in addition to the MEX files. The sample files use several examples to demonstrate the use of the INCA-MIP API.

The examples include a number of M files that access the INCA-MIP API, as well as an INCA database where the elements used in the sample scripts have already been created.

The sample files are copied into the following directories during the installation (see "Installing INCA-MIP" on page 11):

- For installation into MATLAB:

  M files: `%MatlabDir%\toolbox\matlab\demos`
- For installation into ETASData:

  M files: `%EtasDataDir%\INCA-MIPx64`

  INCA demo database: `%EtasDataDir%\Database\db_matlabtest`

To use the sample files, you must first start INCA and open the sample database. No hardware is required.

The function of the M files is described below.

- `tOpen.m` – establishes a connection between INCA and MATLAB. This function must be used at the start of each MATLAB session before using any other function of the INCA-MIP API.
- `tDummy.m` – opens an empty INCA experiment using a hardware configuration with the VADI test device. The script creates several measurement variables in the INCA experiment.
- `tEtkDummy.m` – opens an empty INCA experiment using a hardware configuration with the ETK test device. This script creates several measurement and calibration variables in the INCA experiment. It also downloads the working and reference pages, reads the measurement and calibration variables, and modifies the values for individual calibration variables.
- `tGetRecords(aGroupName).m` – gathers the measured data for the `aGroupName` group for 20 seconds and then passes the data to MATLAB. This function can be used both in connection with the VADI and ETK example (for “measure rasters,” see "INCA Glossary" on page 8.)
- `tPrintDB ({aFolder{, aFileId}}).m` – Writes the complete contents of the database beginning with directory `aFolder` to the file `aFileId`. If the function is executed without parameters, the complete database hierarchy is printed to the standard output.
- `tHWStatus.m` – Example for using the API function `IncaGetHardwareStatus`. MATLAB attaches to an already opened experiment and chooses the first measurement element found in the first measurement device found. It continues with a measurement of 5 minutes. If there is a warning or error during the measurement the measurement cycle gets aborted and restarted after a delay of 5 seconds.

3.2 General Functions

The following general API functions are available:

3.2.1 List INCA-MIP Interface Message IDs

INCA-MIP Interface commands may return with an error.

When using try/catch blocks, detailed error information can be returned.

Example

```
try,
      <command_1>
      ...
      <command_n>
   catch,
      [msgstr,msgid] = lasterr
      ...
   end
```

whereas:

`msgstr` a descriptional string

`msgid` the message id. The following message ids are available:

- `INCA:ParameterError`
- `INCA:ReturnParameterError`
- `INCA:WrongParameterValue`
- `INCA:WrongParameterType`
- `INCA:NaN`
- `INCA:ExecutionError`
- `INCA:ResourceError`
- `INCA:RasterFull`
- `INCA:ObjectIsWriteProtected`
- `INCA:CallSeqenceError`
- `INCA:LicenseError`
- `INCA:RecordingInProcess`
- `INCA:NotInstalled`
- `INCA:WrongVersion`

Hints for reaction on error ids:

`INCA:ParameterError`	Wrong number of input arguments (right hand side parameters)
`INCA:ReturnParameterError`	Wrong number of output arguments (left hand side parameters)
`INCA:WrongParameterValue`	Any of the values of the input arguments is out of its valid range or specification
`INCA:WrongParameterType`	Any of the input arguments has a wrong data type
`INCA:NaN`	Any of the parameters contains a 'not a number' value
`INCA:ExecutionError`	During the command execution an error occurred for some reason. Trying to execute functionality on the INCA user interface could give more information about the reason. Restarting INCA or a reboot could also help.
`INCA:ResourceError`	Unable to get operating system resources. Restarting INCA or a reboot could help.
`INCA:RasterFull`	The acquisition list is full for the requested measure raster when trying to add a measurement
`INCA:ObjectIsWriteProtected`	Unable to calibrate because of a write protected object
`INCA:CallSeqenceError`	Before executing the requested command other commands have to be executed first. E.g. `IncaOpenExperiment` is necessary before `IncaAddMeasureElement`.
`INCA:LicenseError`	To execute the command with the given parameters a license is needed

`INCA:RecordingInProcess`	It is not possible to execute the requested command (e.g. enable or disable signals for recording using `IncaSetRecordingMode`) because a recording is currently running.
`INCA:NotInstalled`	It is not possible to open the specified INCA version using the `IncaOpen` command because the corresponding INCA version is not installed.
`INCA:WrongVersion`	It is not possible to open the specified INCA version using the `IncaOpen` command because of one of the following reasons: • INCA is already started and an `IncaOpen` command is executed with a 'version' parameter different from the already opened INCA version • An `IncaOpen` command is executed from within INCA-MIP for INCA Vx.y with a 'version' parameter with major version != x.

3.2.2 Show Messages During Script Execution

Name `IncaShowMessages`

Description Enables/disables the information display in the MATLAB window during script execution

Syntax `IncaShowMessages(trueOrFalse)`

Output Arguments

Input Arguments

`trueOrFalse`	Numerical parameter whose value either equals or does not equal zero. If the parameter is zero, the information display is disabled, otherwise it is enabled (default).

Examples

3.2.3 Show Whether Valid INCA-MIP License exist (INCA-MIP Extended)

`Name`	`IncaIsLicenseValid`
`Description`	Returns a status indicating whether a valid INCA-MIP license is available or not
`Syntax`	`s = IncaIsLicenseValid`
`Output Arguments`	License status: • `0`: no valid license • `1`: valid license
`Input Arguments`	
`Examples`	`status = IncaIsLicenseValid`

3.2.4 Read Information on All Installed INCA Versions

Name	`IncaGetInstalledProductInfo`	
Description	Provides information on all installed INCA versions. This command can be executed before `IncaOpen`.	
Syntax	`info = IncaGetInstalledProductInfo`	
Output Arguments	`info`	Information on installed INCA versions as MATLAB struct for each installation, consisting of the following entries:
	`info.name`	the product name
	`info.version`	the product version string
	`info.hotfixVersion`	the installed hotfix as string, or an empty string if no hotfix is installed
Input Arguments		
Examples	`i = IncaGetInstalledProductInfo;`	

Note

This command was introduced with INCA-MIP V16.0.

Note

This command only works reliably if INCA V7.1 or higher is installed.

3.2.5 Read Information on All Installed Product Add-ons

Name IncaGetInstalledAddOnInfo

Description Provides information on all installed add-ons for a given product. This command can be executed before `IncaOpen`.

Syntax `info = IncaGetInstalledAddOnInfo(productName, productVersion)`

Output Arguments

`info`	Information on installed add-ons as MATLAB struct for each installation, consisting of the following entries:
`info.name`	the name of the installed add-on
`info.version`	the version string of the installed add-on

Input Arguments

`productName`	the name of the product
`productVersion`	the product version as string. The whole version string is relevant.

Examples

```
i = IncaGetInstalledAddOnInfo('INCA', 'V7.3.0');
i = IncaGetInstalledAddOnInfo('INCA', 'V7.3.0 Beta 42');
```

Note

Make sure that you use for the input arguments `productName` and `productVersion` exactly the name and version of a product as returned by IncaGetInstalledProductInfo.

Note

This command was introduced with INCA-MIP V16.0.

Note

This command only works reliably if INCA V7.1 or higher is installed.

3.2.6 Read INCA Version

Name `IncaGetVersion`

Description Returns the INCA version

Syntax `IncaGetVersion`

Output Arguments

Input Arguments

Examples

3.2.7 Read INCA Properties (INCA-MIP Extended)

Name `IncaGetProperties`

Description Reads properties of INCA

Syntax `p = IncaGetProperties`

Output Arguments `p` Properties of INCA as MATLAB struct, consisting of the following entries:

- `p.databasePath` - Pathname of the open INCA database. If no INCA database is open an empty string is returned.
- `p.dataPath` - Pathname of the INCA data directory.
- `p.installationPath` - Pathname of the INCA installation directory.
- `p.tempPath` - Pathname of the directory which is used by ETAS applications for temporary files.

Input Arguments

Examples

```
p = INCAGetProperties;
```

3.3 Initialization

All measurement and calibration operations in INCA will be performed within the context of an experiment. Before opening an experiment, a workspace with a valid hardware configuration must first be created and assigned.

To work with the INCA-MIP API, there must an empty experiment in the INCA database which is linked to a valid workspace and hardware configuration. The experiment can be opened from within MATLAB.

The following API functions are available for initializing:

3.3.1 Open INCA

Name `IncaOpen`

Description Opens INCA and initialize the connection between MATLAB and INCA.

Syntax
```
IncaOpen
IncaOpen( version )
```

Output Arguments

Input Arguments

`version`	INCA version to be opened (optional). Syntax: `<MajorVersion>.<MinorVersion>`. INCA-MIP for INCA x.y can only connect to INCA installations with the same major version x.

Examples
```
IncaOpen;
IncaOpen('7.3');
```

> **Note**
>
> The optional parameter 'version' was introduced with INCA-MIP V16.0.

3.3.2 Close INCA (INCA-MIP Extended)

Name `IncaClose`

Description Disconnects from INCA and optionally closes it after having successfully connected to INCA with `IncaOpen`.

Syntax
```
IncaClose
IncaClose( isDisconnectOnly )
```

Output Arguments

Input Arguments

`isDisconnectOnly`	Specifies if MATLAB only disconnects from INCA or if it also closes INCA (optional). Possible values are: • `0`: Disconnect from INCA and close it (default). • `1`: Disconnect from INCA and leave it open.

Examples
```
INCAClose;
INCAClose(1);
```

Note

This command was introduced with INCA-MIP V16.0.

With INCA-MIP V16.1, the input argument `isDisconnectOnly` has been added. In previous versions, no input argument was available, and `IncaClose` both disconnected from INCA and closed it.

3.3.3 Open a Database

Name `IncaOpenDatabase`

Description Opens the database in the specified directory

Syntax `IncaOpenDatabase({pathName})`

Output Arguments

Input Arguments

`pathName`	The directory in which the database to be opened is stored. If you do not specify a directory, the current database is opened.

Examples

```
IncaOpenDatabase; % open current database
IncaOpenDatabase('c:\etasdata\mydatabase');
```

3.3.4 Import a Database (INCA-MIP Extended)

Name `IncaDatabaseImport`

Description Imports a database export file (*.exp) into INCA. Existing database items will always be overwritten.

Syntax

```
IncaDatabaseImport(path)
name = IncaDatabaseImport(path)
[name, type] = IncaDatabaseImport(path)
```

Output Arguments

`name`	Array of the full path names of all imported database items use `deblank()` when accessing an array element: `name2 = deblank(name(2,:))`

	`type`	Array of types of all imported database items Possible values are: • `Folder`: a database folder • `Experiment`: an Experiment Environment • `Workspace`: a Workspace • `Asap2Project`: an ASAP2 Project • `MeasurementCatalog`: an ASAP2 Measurement Catalog • `CanDB`: an ASAP2 CAN DB use `deblank()` when accessing an array element: `type2 = deblank(type(2,:))`
Input Arguments	`path`	The full path of the *.exp file to be imported
Examples	`names = IncaDatabaseImport('D:\ETASData\[[[Undefined variable FM_import.INCA_Version_Code]]]\-export\Project0815.exp')`	

3.3.5 Read Database Items (INCA-MIP Extended)

Name	`IncaBrowseItemsInFolder`
Description	Reads database items in the given database folder with a search pattern
Syntax	`[name, type] = IncaBrowseItemsInFolder(pattern, folderName)`

Output Arguments	`name`	List of names of the database items
	`type`	List of types of the database items: • `Folder`: Database folder • `Experiment`: Experiment • `Workspace`: Workspace • `Asap2Project`: ASAP2 Project

Input Arguments	`pattern`	Search pattern for database entries to look for. A '*' matches zero or any number of additional characters. A '#' matches exactly one character. All other characters have to match with the database items. There is no difference between lower and upper case.
	`folderName`	Database folder in which the database items are read. Folder hierarchies are separated by a '\'. An empty string is used for the upper most hierarchy level.

Examples

```
[n,t]=IncaBrowseItemsInFolder('*',
'DEFAULT\MyProject');
[name,type]=IncaBrowseItemsInFolder('Prj*_
##', '');
```

3.3.6 Assign Project and Dataset in Device (INCA-MIP Extended)

Name `IncaSetProjectAndDatasetInDevice`

Description Assigns a project and dataset to a device in a given workspace. This can only be done if no experiment is currently opened.

Syntax

```
IncaSetProjectAndDatasetInDevice(workspace,
device,project,dataset)
```

Output Arguments

Input Arguments	`workspace`	database path of workspace
	`device`	name of device
	`project`	database path of project
	`dataset`	database path of dataset

Examples

```
IncaSetProjectAndDatasetInDevice('DEFAULT\ work-
space','ETK:1','DEFAULT\Prj0815','Ds4711\Ds471-
1_3')
```

3.3.7 Open an Experiment

Name `IncaOpenExperiment`

Description Opens the specified experiment (Experiment Environment). After an experiment has been opened, you can use the INCA-MIP API to add your measurement and calibration variables as desired.

Syntax

```
IncaOpenExperiment({closeAllViewsFlag})
```

or

```
IncaOpenExperiment(expFolderName, experimentName, workspaceFolderName, workspaceName{, closeAllViewsFlag})
```

Output Arguments

Input Arguments

`expFolderName`	directory in which the experiment is stored
`experimentName`	name of the experiment
`workspaceFolderName`	directory in which the workspace is stored
`workspaceName`	name of the workspace
`closeAllViewsFlag`	Closes all measure and calibration windows in the selected experiment. Possible settings: • `1`: closes all windows (default) • `0`: leaves the windows unchanged.

Examples

```
IncaOpenExperiment('ExpFolder', 'MyExperiment', 'WorkspaceFolder', 'MyWorkspace');
```

Note

This function has changed since INCA V3.0, because two input arguments have been added to the re-worked workspace.

If the experiment is already open when the function `IncaOpenExperiment` is called, the input arguments specifying the environment are optional. If the experiment is not yet open, you need to call `IncaOpenDatabase` before `IncaOpenExperiment`.

3.3.8 Reset an Experiment

Name `IncaResetExperiment`

Description Resets and closes the current experiment. You can use this function to remove all variables from an experiment. Removing individual variables is currently not supported.

Syntax `IncaResetExperiment`

Output Arguments

Input Arguments

Examples

> **Note**
>
> If the experiment has been opened manually and not by means of a MATLAB command, `IncaResetExperiment` releases the experiment, but does not close the window. You need to execute `IncaOpenExperiment` before you can access the experiment once again.

3.3.9 Read Devices (INCA-MIP Extended)

Name `IncaGetDevices`

Description Reads all devices in the experiment

Syntax `[name, type] = IncaGetDevices`

Output Arguments

`name`	List of names of the devices
`type`	List of device types: • `WorkbaseDevice`: Device with datasets • `MeasurementDevice`: Measurement device

Input Arguments

Examples [name,type]=IncaGetDevices;

3.3.10 Read Device Properties (INCA-MIP Extended)

Name `IncaGetDeviceProperties`

Description Reads properties of a device

Syntax `p = IncaGetDeviceProperties(deviceName)`

Output Arguments	`p`	Device properties as MATLAB struct, consisting of the following entries:
	`p.name`	device name
	`p.descriptionFile`	Pathname of the description file of the project assigned to the device. An empty string is returned if there is no project assigned to the device.
	`p.binaryFile`	Pathname of the binary file of the project assigned to the device. An empty string is returned if there is no project assigned to the device.
	`p.projectDBPath`	Pathname within the INCA database of the project assigned to the device. An empty string is returned if there is no project assigned to the device.
	`p.isWriteProtected`	• `0`: Device has no memory pages or current page is not write protected • `1`: Current page is write protected
	`p.isActive`	• `0`: Device is not connected or not active • `1`: Device is connected and active
	`p.isWorkbaseDevice`	• `0`: Device has no datasets • `1`: Device has datasets
Input Arguments	`deviceName`	name of the device
Examples	`p = IncaGetDeviceProperties('Device');`	

3.4 Measuring and Recording

A signal or measurement variable is always captured as part of a measure raster for that particular measuring device. Each measurement variable may appear in one measure raster only. To configure an experiment, first assign the measurement variables to the individual measure rasters.

Note

Note that the names of elements, devices, signals, and measure rasters are case-sensitive.

3.4.1 Read Measurement Elements (INCA-MIP Extended)

Name `IncaBrowseMeasureElements`

Description Gets measurement elements of the experiment with search pattern and optional device

Syntax
```
[name, type] = IncaBrowseMeasureElements (pattern, {deviceName})
[name] = IncaBrowseMeasureElements(pattern, {deviceName})
```

Output Arguments

`name`	List of names of measurement elements
`type`	List of types of the measurement elements: • `Scalar`: Scalar • `Array`: Vector • `Matrix`: Matrix

Input Arguments

`pattern`	Search pattern for the measurement elements to look for. A '*' matches zero or any number of additional characters. A '#' matches exactly one character. All other characters have to match with the measurement element. There is no difference between lower and upper case.
deviceName	Name of the device

Examples
```
[n,t]=IncaBrowseMeasureElements('ign*', 'Device');
[name,type]=IncaBrowseMeasureElements('*');
```

3.4.2 Read Measure Rasters (INCA-MIP Extended)

Name `IncaGetMeasureRatesForDevice`

Description Gets all measure rasters of a device

Syntax
```
[name] = IncaGetMeasureRatesForDevice (deviceName)
```

Output Arguments

`name`	List of names of the measure rasters

Input Arguments

`deviceName`	Name of the device

Examples
```
n=IncaGetMeasureRatesForDevice('Device');
name=IncaGetMeasureRatesForDevice('Dev');
```

3.4.3 Add Measurement Variable to Experiment

Name `IncaAddMeasureElement`

Description Adds a measurement variable with or without given measure raster to an experiment.

Syntax

```
IncaAddMeasureElement(deviceName, groupName,
signalName {, displayMode})
groupName = IncaAddMeasureElement(
deviceName, [], signalName{, displayMode})
```

Output Arguments

Input Arguments

`deviceName`	name of the device
`groupName`	name of the measure raster It is possible to use multiple rasters by simply combining raster names by means of a '+' character, e.g. '`10ms+100ms`'. When using such a multi-raster, a new virtual raster is created. Each signal can only be measured in exactly one raster or multi-raster. The group name may be [] (see note below).
`signalName`	name of the measurement signal. For scalars, just the name is sufficient; for vectors and matrices, the index in the format `[n]` or `[n,m]` has to be appended to the name. The first element has the index of "zero."
`displayMode`	display mode for the element: • `1`: measurement variable is displayed (default) • `0`: no display

Examples

```
IncaAddMeasureElement('MyDevice', '10ms',
'Channel01', 0);
IncaAddMeasureElement('ETK:1', '1.0ms', 'Mat-
rix[2,1]');
group = IncaAddMeasureElement('CalcDev', [],
'MyCalcSig1');
```

Note

If the measure raster is full, the measurement variable does not get added to the raster.

Note

If the input argument `groupName` is [] (i.e. empty), the signal group will be determined in the following way:
- If the signal is already part of the experiment, its existing signal group name is used.
- If the signal is not part of the experiment, any available signal group is used arbitrarily. In the case of the Calculated device (CalcDev, used for calculated signals), or CAN Monitoring, the signal group that is defined for that signal is used.

As the name of the signal group is needed for `IncaGetRecords`, `IncaGetRecordStruct` or `IncaGetRecordCount`, it is returned as optional left hand side parameter.

Examples:

```
groupName = IncaAddMeasureElement( 'CalcDev', [],
'MyCalcSig')
groupName = IncaAddMeasureElement( 'CAN-Monitoring:1',
[], 'nmot', 1)
```

Note

Using an empty groupName is only possible since INCA-MIP V16.0.

Note

The total number of signals that can be added is device-specific as well as protocol-specific. The number of signals is limited by the amount of free buffer memory allocated by the Target Server Process. The total size of buffer memory depends on the used sample rate.

Example:
A signal with 0.1 ms sample rate requires > 3 Megabyte of data. Therefore the total number of signals that can be added is between 400 and 600 signals. Slower sample rates allow to add more signals.

3.4.4 Start Measurement

Name `IncaStartMeasurement`

Description Starts a measurement in INCA

Syntax `IncaStartMeasurement`

Output Arguments

Input Arguments

Examples

3.4.5 Stop Measurement

Name `IncaStopMeasurement`

Description Stops the current measurement and recording (if enabled) in INCA

Syntax `IncaStopMeasurement{(mdfFileName)}`

Output Arguments

Input Arguments

`mdfFileName`	name of the MDF file in which the recorded data are saved if recording is stopped together with the current measurement. Always specify the full access path to the file (e.g., `'c:\mydata\store1.dat'`).

Examples IncaStopMeasurement('c:\mydata\store1.dat');

> **Note**
>
> To avoid data loss due to ongoing measurements, be sure to stop the current recording with the `IncaStopMeasurement(mdfFileName)` command if the data volume is high. Follow this up with the `IncaGetRecords` command to transfer the remaining data to MATLAB.

3.4.6 Read Recording Properties (INCA-MIP Extended)

Name `IncaGetRecordingProperties`

Description Reads the properties of the default recorder's primary output file and the file extension for the selected primary recording format.

Syntax `properties = IncaGetRecordingProperties`

Output Arguments

`properties`	Recording properties as MATLAB struct, consisting of the following entries:
`properties.fileName`	the recording file name including the primary output file format
`properties.directory`	the directory for the recording file

	`properties.fileFormat`	the file format of the recording file; the following format strings are valid: • `ETASBinary` • `#DiademATF` • `ETASAscii` • `ETASGroupAscii` • `ETASMATLABMFILE` • `ETASGroupMatlabM` • `FamosRecord` • `ETASMDF` • `ETASMDF4`
	`properties.autoIncrement`	Automatically increment the recording file name • `0`: don't auto increment • `1`: do auto increment
	`properties.comment`	The comment in the recording file header. It must not exceed 1024 characters minus number of characters used for the default comment.
	`properties.defaultComment`	The default comment in the recording file header generated by INCA
	`properties.company`	The company in the recording file header
	`properties.project`	The project in the recording file header
	`properties.user`	The user in the recording file header
	`properties.vehicle`	The vehicle in the recording file header
Input Arguments		
Examples	`properties = IncaGetRecordingProperties;`	

Note

With INCA-MIP V16.0, the format IDs for the output argument `properties.fileFormat` have changed:

Before INCA-MIP V16.0:

- ETAS binary data file format
- Diadem ATF measure format (write only)
- ASCII
- ASCII (Multirate/Write only)
- MATLAB M-File
- MATLAB M-File (Multirate/Write only)
- FAMOS (imc)
- MDF (Measure Data Format)
- MDF4 (Measure Data Format for VSG using 4byte floats only)

From INCA-MIP V16.0:

- ETASBinary
- #DiademATF
- ETASAscii
- ETASGroupAscii
- ETASMATLABMFILE
- ETASGroupMatlabM
- FamosRecord
- ETASMDF
- ETASMDF4

3.4.7 Set Recording Properties (INCA-MIP Extended)

Name	`IncaSetRecordingProperties`	
Description	Sets properties for the next measurement data recording	
Syntax	`IncaSetRecordingProperties(properties)`	
Output Arguments		
Input Arguments	`properties`	recording properties as matlab struct containing any combination of the following field designators:
	`properties.fileName`	the recording file name
	`properties.directory`	the directory for the recording file

`properties.fileFormat`	the file format of the recording file; the following format strings are valid: • `ETASBinary` • `#DiademATF` • `ETASAscii` • `ETASGroupAscii` • `ETASMATLABMFILE` • `ETASGroupMatlabM` • `FamosRecord` • `ETASMDF` • `ETASMDF4`
`properties.autoIncrement`	Automatically increment the recording file name • `0`: don't auto increment • `1`: do auto increment
`properties.comment`	the recording file header comment must contain max. 1024 characters minus the character count of the default comment
`properties.company`	the company in the recording file header company
`properties.project`	the project in the recording file header
`properties.user`	the user in the recording file header
`properties.vehicle`	the vehicle in the recording file header

Examples

```
properties.user = 'Michael';
properties.project = 'K70';
IncaSetRecordingProperties(properties);
```

Note

With INCA-MIP V16.0, the format IDs for the output argument `properties.fileFormat` have changed:

Before INCA-MIP V16.0:

- ETAS binary data file format
- Diadem ATF measure format (write only)
- ASCII
- ASCII (Multirate/Write only)
- MATLAB M-File
- MATLAB M-File (Multirate/Write only)
- FAMOS (imc)
- MDF (Measure Data Format)
- MDF4 (Measure Data Format for VSG using 4byte floats only)

From INCA-MIP V16.0:

- ETASBinary
- #DiademATF
- ETASAscii
- ETASGroupAscii
- ETASMATLABMFILE
- ETASGroupMatlabM
- FamosRecord
- ETASMDF
- ETASMDF4

Note

When setting the recording properties with `IncaSetRecordingProperties`, you should not finish a recording with `IncaStopRecording`. Instead `IncaSetTrigger` can be used. Any trigger condition can be used to stop the recording.

Examples:

Stop recording after a constant duration:

```
TIMEDURATION_SECONDS = 25;
IncaSetTrigger('none', 'none', 'none', 'none',
TIMEDURATION_SECONDS);
IncaStartRecording;
% Recording automatically stops after TIMEDURATION_
SECONDS seconds
```

Stop recording after a manual trigger condition:

```
IncaSetTrigger('none', 'manual');
IncaStartRecording;
% Do anything until the stop trigger condition is met
...
IncaExecuteManualTrigger('stop');
```

3.4.8 Read Recording Mode (INCA-MIP Extended)

Name	`IncaGetRecordingMode`
Description	Indicates whether a signal is recorded in the default recorder or not.
Syntax	`IncaGetRecordingMode(deviceName, signalName)`
Output Arguments	recording mode for the default recorder: • `0`: the signal is not recorded in the default recorder • `1`: the signal is recorded in the default recorder

Input Arguments	`deviceName`	name of the device
	`signalName`	name of the measurement signal. For scalars, just the name is sufficient; for vectors and matrices, the index in the format `[n]` or `[n,m]` has to be appended to the name. The first element has the index of "zero."

Examples

```
m = IncaGetRecordingMode('ETK:1', 'hfm');
mode = IncaGetRecordingMode('CalcDev', 'MyCalcSig1');
```

Note

This command was introduced with INCA-MIP V16.0.

Note

Before using `IncaGetRecordingMode`, the signal has to be added with `IncaAddMeasureElement`.

3.4.9 Set Recording Mode (INCA-MIP Extended)

Name `IncaSetRecordingMode`

Description Enables or disables the recording of a signal in the default recorder. The recording can only be disabled for signals which are displayed in the INCA experiment.
Before executing this command the signal has to be added to the experiment with `IncaAddMeasureSignal`.

Syntax

```
IncaSetRecordingMode(deviceName, signalName, recordingMode)
```

Output Arguments

Input Arguments	`deviceName`	name of the device

`signalName`	name of the measurement signal. For scalars, just the name is sufficient; for vectors and matrices, the index in the format `[n]` or `[n,m]` has to be appended to the name. The first element has the index of “zero.”
`recordingMode`	recording mode for the default recorder: • `0`: the measurement variable is removed from the default recorder • `1`: the measurement signal is added to the default recorder

Examples

```
IncaSetRecordingMode(’ETK:1’, ’hfm’, 1);
IncaSetRecordingMode('CalcDev', 'MyCalcSig1', 0);
```

Note

This command was introduced with INCA-MIP V16.0.

Note

Before using `IncaSetRecordingMode`, the signal has to be added with `IncaAddMeasureElement`.

3.4.10 Start Recording

Name `IncaStartRecording`

Description Starts recording in INCA. This function can be used after or instead of `IncaStartMeasurement`.

After a measurement or recording has been started, the measured data are also available in MATLAB.

Syntax `IncaStartRecording`

Output Arguments

Input Arguments

Examples

3.4.11 Stop Recording

Name `IncaStopRecording`

Description Stops the current recording in INCA. The measurement continues to run and must be stopped explicitely with `IncaStopMeasurement`. It is possible to toggle the recording on/off several times during a running measurement.

Syntax `IncaStopRecording(mdfFileName)`

Output Arguments

Input Arguments

`mdfFileName`	The name of the MDF file where the recorded data are saved. Always specify the complete access path to the file (e.g., `'c:\mydata\store1.dat'`).

Examples IncaStopRecording('c:\mydata\store1.dat');

3.4.12 Set Data Reading Mode (Online/Offline Data)

Name `IncaSetMeasureReadMode`

Description Determines from which source the measured data are transmitted to MATLAB. The data are either first prepared in INCA and then transferred to MATLAB (offline data), or they are read directly from the device buffer (online data).

For some devices, such as the ES1303 card and the ES6xx series devices, there are no offline data available during the display of measure data. During the measure data display, it is recommended to use only online data.

When recording measure data, both online and offline data can be used. In each case, the optimized transmission features produce special display characteristics of the results: online data may be incomplete at high loads, while offline data are always complete when measured data are recorded. However, offline data can only be transmitted with a certain time delay at high loads. It is recommended that you use only offline data while measured data are being recorded.

Syntax `IncaSetMeasureReadMode(measureReadMode)`

Output Arguments

Input Arguments

`measureReadMode`	numerical parameter whose value specifies the data source. Possible settings: • `1`: Offline data • `0`: Online data (default)

Examples

```
IncaSetMeasureReadMode(0);
```

3.4.13 Read Measure Data

Name `IncaGetRecords`

Description Transfers measure data to MATLAB. The measure data of each signal group is stored in a dedicated ring buffer which can hold data for up to 30 seconds of measuring time. The measure data is retrieved from MATLAB in groups. You should therefore stop your script execution in MATLAB after you have retrieved the measure data. The greater the amount of data being transferred at each time, the more efficient the data transfer is.

This function transmits a specified number of records for the specified measure raster.

For further information on the ring buffer see the corresponding entry in the "INCA Glossary" on page 8.

Syntax

```
[time, data {,state}] = IncaGetRecords
(deviceName, groupName, maxRecords {{,latest
{, exact}})
```

Output Arguments

`time`	A vector containing the time stamps of the transferred records. This variable contains a maximum number of `m` values, whereas `m <= maxRecords`.
`data`	A 2-dimensional matrix containing the data values for each measure variable in the order in which it was added to the experiment by `IncaAddMeasureElement`. In this matrix, the dimension m reflects the number of transferred records, whereas n indicates the number of measure rasters.
state	Optional return parameter: • 0: Success. Records received • 1: Acquisition not running. No records received • 2: Not enough records. No records received. This can only be returned if exact = 1

Input Arguments

`deviceName`	Name of the device

`groupName`	Name of the measure raster It is possible to use multiple rasters by simply combining raster names by means of a '+' character, e.g. '`10ms+100ms`'. When using such a multi-raster, a new virtual raster is created. Each signal can only be measured in exactly one raster or multi-raster.
`maxRecords`	(Maximum) number of records to be received. See also parameter `exact`. The number you enter here is the dimension `m` for the time or data variable above. If this dimension reaches the value of `maxRecords`, not all existing records are read so that the ring buffer may overflow.
latest	Defines if the oldest or latest n records will be received
exact	It defines to receive records also if n < maxRecords are available in the ring buffer, or if the ring buffer should be unchanged.

Examples

```
[t, d] = IncaGetRecords('ETK:1', '100ms',
500);
data = [data; d];
time = [time; t];
[t, d, s] = IncaGetRecords('ETK:1', '100ms',
25, 1, 1);
```

For a larger context using this piece of code please see .

Note

The optional input arguments 'latest' and 'exact' and the optional output argument 'state' were introduced with INCA-MIP V16.0.

Note

The raster used in `IncaGetRecords` directly corresponds to the raster used in `IncaAddMeasureElement`, i.e. you have to use the same raster or multi-raster. Example:

```
IncaAddMeasureElement('ETK test device:1','RASTER_
A+RASTER_B', 'N')
IncaAddMeasureElement('ETK test device:1','RASTER_
A+RASTER_B', 'n')
[t,d]= IncaGetRecords('ETK test device:1','RASTER_
A+RASTER_B',15)
```

To check the raster assignment of signals, you can use the command `IncaGetRecordStruct`. Example:

```
l=IncaGetRecordStruct('ETK test device:1', 'RASTER_
A+RASTER_B')
```

Note

The following parameter combinations execute as follows:

- `latest = 0, exact = 0: (default)`
  Returns the oldest up to `maxRecords` records from the ring buffer. Any newer records remain unchanged.
- `latest = 1, exact = 0:`
  Returns the latest up to `maxRecords` records from the ring buffer. Any older records intentionally are rejected.
- `latest = 0, exact = 1:`
  Returns the oldest `maxRecords` records from the ring buffer. Any newer records remain unchanged. If only n < `maxRecords` records are availabe in the ring buffer, nothing is received.
- `latest = 1, exact = 1:`
  Returns the latest `maxRecords` records from the ring buffer. Any older records are rejected intentionally. If only n < `maxRecords` records are availabe in the ring buffer, nothing is received.

3.4.14 Reset Ring Buffer

Name `IncaResetRecords`

Description Resets the ring buffer for all signal groups. This function can even be used during a running measurement to reset all ring buffers. They are reset automatically when starting a measurement or recording; it is not necessary to issue this command explicitely.

For further information on the ring buffer see the corresponding entry in the "INCA Glossary" on page 8.

Syntax `IncaResetRecords`

Output Arguments

Input Arguments

Examples

3.4.15 Read Hardware Status (INCA-MIP Extended)

Name `IncaGetHardwareStatus`

Description Gets the current Hardware Status during a measurement or recording

Syntax `[status, message] = IncaGetHardwareStatus`

Output Arguments

`status` current Hardware Status
- `0`: Status ok
- `1`: Warning
- `2`: Error

`message` If `status` returns `1` or `2`, `message` returns a text describing the warning or error.

If `status` returns `1` or `2`, the measurement or recording has to be finished before `IncaGetHardwareStatus` can be called once again.

See also example script `tHWStatus.m` in "Getting to know the INCA-MIP API through Sample Files" on page 18

Input Arguments

Examples

3.4.16 Set Trigger (INCA-MIP Extended)

Name `IncaSetTrigger`

Description Sets the trigger condition before starting a measurement or recording with `IncaStartMeasurement` or `IncaStartRecording`

Syntax
```
IncaSetTrigger(startTrigger{, stopTrigger{, preTriggerTime{, postTriggerTime{, duration}}}})
```

Output Arguments

Input Arguments

`startTrigger`	Start trigger condition. • `manual` for a manual start trigger • `none` if no trigger is to be used
`stopTrigger`	Stop trigger condition. • `manual` for a manual stop trigger • `none` if no trigger is to be used (default)
`preTriggerTime`	The pre trigger time in seconds • `none` if unspecified (default)
`postTriggerTime`	The post trigger time in seconds • `none` if unspecified (default)
`duration`	Duration of measurement or recording in seconds • `none` if unspecified (default); in this case the duration is infinite.

Examples
```
IncaSetTrigger('nmot\ETK:1 > 2000', 'none', 2.0, 3.0)
IncaSetTrigger('none', 'none', 'none', 'none', 360)
```

Note

The following table lists all combinations of input parameters that are supported (other combinations lead to an exception):

Trigger functionality	startTrigger	stopTrigger	preTriggerTime	postTriggerTime	duration
Recording with time duration	'none'	'none'	'none'	'none'	value
Recording with manual start trigger, pre-trigger time and manual stop trigger condition	'manual'	'manual'	value	'none'	'none'
Recording with manual start trigger, pre- and post-trigger time	'manual'	'none'	value	value	'none'
Recording with manual start trigger, pre-trigger time and stop trigger condition	'manual'	value	value	'none'	'none'
Recording with manual stop trigger condition	'none'	'manual'	'none'	'none'	'none'
Recording with start trigger condition and recording duration	value	'none'	'none'	'none'	value

Trigger functionality	startTrigger	stopTrigger	preTriggerTime	postTriggerTime	duration
Recording with start trigger condition and manual stop trigger	value	'manual'	'none'	'none'	'none'
Recording with start trigger condition, pre-trigger time and manual stop trigger	value	'manual'	value	'none'	'none'
Recording with start trigger condition, pre- and post-trigger time	value	'none'	value	value	'none'
Recording with start trigger condition, pre-trigger time and stop trigger condition	value	value	value	'none'	'none'

3.4.17 Execute Manual Trigger (INCA-MIP Extended)

Name `IncaExecuteManualTrigger`

Description Executes a manual start or stop trigger. This only has an effect if a `IncaSetTrigger` command has been set before with the `startTrigger` or `stopTrigger` parameter set to `manual` .

Syntax `IncaExecuteManualTrigger(type)`

Output Arguments		
Input Arguments	`type`	Trigger type • `start` for executing a manual start trigger • `stop` for executing a manual stop trigger
Examples	`IncaExecuteManualTrigger('start')`	

3.4.18 Read Recording State (INCA-MIP Extended)

Name	`IncaGetRecordingState`	
Description	Gets the current recording status.	
Syntax	`result = IncaGetRecordingState`	
Output Arguments	result	recording status • `0`: switched off • `1`: waiting for trigger or recording in progress
Input Arguments		
Examples	`s = IncaGetRecordingState`	

3.4.19 Read List of Measurement Variables (INCA-MIP Extended)

Name	`IncaGetRecordStruct`	
Description	Gets list of measurement variables which have been assigned for measurement or recording. The list returns the measurement names in the same order as they have been assigned with `IncaAddMeasureElement`.	
Syntax	`list = IncaGetRecordStruct(device,groupName)`	
Output Arguments		
Input Arguments	`device`	name of device
	`groupName`	name of measure raster It is possible to use multiple rasters by simply combining raster names by means of a '+' character, e.g. '`10ms+100ms`'.
Examples	`l = IncaGetRecordStruct('ETK:1', '10ms');` `list = IncaGetRecordStruct('device1', 'Syncro');`	

3.5 Calibrating

Calibrations can be performed with scalars, characteristic curves and maps, including the associated break point distributions. In each experiment, it is possible to define any number of calibration variables.

> **Note**
>
> Note that the names of calibration variables are case-sensitive.

3.5.1 Read Calibration Elements (INCA-MIP Extended)

Name `IncaBrowseCalibrationElements`

Description Gets calibration elements of the experiment with search pattern and optional device.

Syntax

```
[name, type] = IncaBrowseCalibrationElements (pattern, {deviceName})
name = IncaBrowseCalibrationElements (pattern, {deviceName})
```

Output Arguments

`name`	List of names of calibration elements
`type`	List of types of the calibration elements: • `Distribution`: Axis distribution • `OneDTable`: Curve • `TwoDTable`: Map • `Scalar`: Scalar • `Array`: Vector • `Matrix`: Matrix

Input Arguments

`pattern`	Search pattern for the calibration elements to look for. A '*' matches zero or any number of additional characters. A '#' matches exactly one character. All other characters have to match with the calibration element. There is no difference between lower and upper case.
deviceName	Name of the device

Examples

```
[n,t]=IncaBrowseCalibrationElements('MAP*', 'Device');
[name,type]= IncaBrowseCalibrationElements ('*');
```

3.5.2 Add Calibration Element

Name `IncaAddCalibrationElement`

Description Adds a calibration variable to the current experiment. Calibrations can be performed with scalars, characteristic curves and maps including the associated axis point distributions. In each experiment it is possible to define any number of calibration variables. Also axis point distributions and group axis point distributions are supported with this command.

Syntax
```
IncaAddCalibrationElement(deviceName, calibrationName {, displayMode})
```

Output Arguments

Input Arguments

`deviceName`	name of the device
`calibrationName`	name of the calibration element
`displayMode`	display mode for the element: • `2`: the calibration variable is displayed and constantly updated (default) • `1`: it is displayed but not updated • `0`: no display Selecting `1` (display only) may considerably improve the performance at high data volumes.

Examples
```
IncaAddCalibrationElement('anEtk', 'Scalar');

IncaAddCalibrationElement('anEtk', 'Curve');
IncaAddCalibrationElement('anEtk', 'Map');
```

Note

Calibration elements of the types 'axis' and 'group axis' are supported since INCA-MIP V16.0. For group axes no interpolation of the dependent curves and maps is executed.

3.5.3 Read Calibration Value

Name `IncaGetCalibrationValue`

Description Reads the current value of a calibration variable or the associated break point distribution

Syntax
```
value = IncaGetCalibrationValue(deviceName,
calibrationName {, start, size} {,
valueType})
```

Output Arguments

`value`	the current value of the calibration variable; it must match the data types specified below: • Scalars: a (1,1) matrix • Curves: an (x,1) matrix • Maps: an (x,y) matrix • Break point distributions: an (x,1)-matrix

Input Arguments

`deviceName`	name of the device
`calibrationName`	the name of the calibration element

`start`	Startindex. Supported datatypes:
	• For curves and break point distributions a startindex `x` has to be specified.
	`x >= 1`
	• For maps a startindex `[x,y]` has to be specified.
	`x,y >= 1`

`size`	Number of values to read. Supported datatypes:
	• For curves and break point distributions a count `n` has to be specified.
	`n >= 1`
	• For maps a count `[n,m]` has to be specified.
	`n,m >= 1`

`valueType`	selection of the output argument (string). The function either returns the value of the calibration variable (default) or the X- and Y-break point distribution. Possible settings: • `v`: value • `x`: x break point (curves and maps • `y`: y break point (maps)

Examples

```
aValue = IncaGetCalibrationValue('anEtk',
'Scalar');
aCurve = IncaGetCalibrationValue('anEtk',
'Curve');
aMap = IncaGetCalibrationValue('anEtk',
'Map');
xMap = IncaGetCalibrationValue('anEtk',
'Map', 'x');
yMap = IncaGetCalibrationValue('anEtk',
'Map', 'y');
aCurveRange = IncaGetCalibrationValue
('anEtk', 'Curve', 2, 3);
aMapRange = IncaGetCalibrationValue ('anEtk',
'Map', [2,3], [3,4]);
xMapRange = IncaGetCalibrationValue ('anEtk',
'Map', 2, 3, 'x');
```

3.5.4 Change Calibration Value

Name `IncaSetCalibrationValue`

Description Assigns a value to a calibration variable or associated break point

Syntax

```
IncaSetCalibrationValue( deviceName, calibrationName, value )
IncaSetCalibrationValue( deviceName, calibrationName, value, valueType )
IncaSetCalibrationValue( deviceName, calibrationName, value, start )
IncaSetCalibrationValue( deviceName, calibrationName, value, start, valueType )
result = IncaSetCalibrationValue( deviceName, calibrationName, value )
result = IncaSetCalibrationValue( deviceName, calibrationName, value, valueType )
result = IncaSetCalibrationValue( deviceName, calibrationName, value, start )
result = IncaSetCalibrationValue( deviceName, calibrationName, value, start, valueType )
```

Output Arguments	`result`	result of calibration (optional, only in case of failures) If no result bit is set, then the calibration value has been successfully modified. This is also the case if one of the bits 5 to 8 is set, which provide additional information. If, however, one of the bits 0 to 4 is set, then the calibration has failed. • Bit 0 set: calibration not done • Bit 1 set: lower weak bound violated • Bit 2 set: upper weak bound violated • Bit 3 set: lower hard bound violated • Bit 4 set: upper hard bound violated • Bit 5 set: limited to lower weak bound • Bit 6 set: limited to upper weak bound • Bit 7 set: limited to lower hard bound • Bit 8 set: limited to upper hard bound There may be different causes for a calibration not to be executed. For instance, depending on the active calibration mode, any of the bounds would have been violated. In this case more detailed information is returned by bit 1 to 4. Another cause might be that the calibration element or active page is write protected or a x- or y-distribution would be violating the monotony. In all these cases only bit 0 is set.
Input Arguments	`deviceName`	name of the device
	`calibrationName`	name of the calibration element
	`value`	value of the calibration element. Acceptable data types: • Scalars: a (1, 1) matrix • Curves: an (x, 1) matrix • Maps: an (x, y) matrix • x and y break point distributions: an (x, 1) matrix

	`start`	Start index. Supported datatypes:
		• For curves and break point distributions a startindex `x` has to be specified.
		`x >= 1`
		For maps a startindex `[x,y]` has to be specified.
		`x,y >= 1`

	`valueType`	Selection of value (string). The function modifies either the value of the calibration variable (default) or the X/Y break point distribution. Possible settings: • `v`: value (default) • `x`: x break point (curves and maps) • `y`: y break point (maps)

Examples

```
IncaSetCalibrationValue('anEtk', 'Scalar',
aValue);
IncaSetCalibrationValue('anEtk', 'Curve',
aCurve);
IncaSetCalibrationValue('anEtk', 'Map',
aMap);
IncaSetCalibrationValue('anEtk', 'Map', xMap,
'x');
IncaSetCalibrationValue('anEtk', 'Map', yMap,
'y');
IncaSetCalibrationValue('anEtk', 'Curve',
aCurveRange, 2);
IncaSetCalibrationValue('anEtk', 'Map',
aMapRange,[2,3]);
IncaSetCalibrationValue('anEtk', 'Map',
xMapRange, 2, 'x');
```

3.5.5 Assign Dataset to Device (INCA-MIP Extended)

Name `IncaSetDatasetInDevice`

Description Assigns a dataset to a device in an open experiment

Syntax `IncaSetDatasetInDevice(device,dataset)`

Output Arguments

Input Arguments	`device`	name of device
	`dataset`	database path of dataset

Examples

```
IncaSetDatasetInDevice
('ETK:1','Ds4711\Ds4711_3')
```

3.5.6 List Datasets of a Device (INCA-MIP Extended)

Name `IncaGetDatasetsForDevice`

Description Gets a list of all dataset names for a given device

Syntax
```
name = IncaGetDatasetsForDevice(device)
[name, properties] = IncaGetDatasetsForDevice(device)
```

Output Arguments

`name`	A string list with the full path of all datasets found
`properties`	A string list of the dataset properties Possible values are: • '' (empty string): A dataset with read-write access • `r` : A dataset with read-only access • `m` : A master dataset with read-write access • `mr`: A master dataset with read-only access

Input Arguments

`device`	name of device

Examples
```
l = IncaGetDatasetsForDevice('ETK:1')
```

3.5.7 Set Calibration Mode (INCA-MIP Extended)

Name `IncaSetCalibrationMode`

Description Sets the global calibration mode valid for all subsequent calibrations done with `IncaSetCalibrationValue`. The mode remains valid even after closing and reopening an experiment. When starting the MATLAB Interface the default mode for both lower and upper limits is `rejectWeakBoundViolation`.

Syntax
```
IncaSetCalibrationMode(lowerLimitMode, upperLimitMode)
```

Output Arguments

Input Arguments	`lowerLimitMode`	The new Calibration Mode for lower limits
	`upperLimitMode`	The new Calibration Mode for upper limits: • `rejectWeakBoundViolation`: reject complete calibration if weak bound would be violated at least once (default) • `limitToWeakBound`: If min. or max. weak bound limit would be violated use min. or max. weak bound value instead • `rejectHardBoundViolation`: Ignore weak bounds. Reject complete calibration if hard bound would be violated at least once • `limitToHardBound`: Ignore weak bounds. If min. or max. hard bound limit would be violated use min. or max. hard bound value instead
Examples	`IncaSetCalibrationMode ('rejectHardBoundViolation', 'limitToHardBound')`	

3.5.8 Group Devices (INCA-MIP Extended)

Name	`IncaGroupDevices`	
Description	Activates or deactivates Device Grouping	
Syntax	`IncaGroupDevices(onOff)`	
Output Arguments		
Input Arguments	`onOff`	• `0`: Deactivate device grouping • `1`: Activate device grouping
Examples	`IncaGroupDevices(1)`	

3.5.9 Write DCM File (INCA-MIP Extended)

Name	`IncaWriteToFile`
Description	Writes a DCM file within an open experiment
Syntax	`IncaWriteToFile(format,file,device, calibs {,options})`

Output Arguments		
Input Arguments	`format`	File format identifier: • '`DCM`': DCM format
	`file`	Full path of file to be written to
	`device`	Device whose calibration elements will be written
	`calibs`	List of calibration elements to write (as cell array)
	`options`	Options used for writing in specified format

Examples

```
calibs = {'A0_KW', 'BRABEVI_KL', 'KFZW_GKF'};
IncaWriteToFile('DCM','C:\DCMOut1.dcm',
'device1', calibs);
IncaWriteToFile('DCM','C:\DCMOut2.dcm',
'ETK:1','A0_KW');
```

3.6 Memory Page Manager

All previously described API functions are effective for a device's currently active page. In principle, calibration access is possible only from the working page. However, it might occur that write access to the ETK's working page is blocked because the checksums of the working pages in the INCA database and in the ETK do not match.

The following API functions can be used for memory page management.

3.6.1 Activate Memory Page

Name	`IncaSwitchPage`	
Description	Activates the specified memory page.	
Syntax	`IncaSwitchPage(deviceName, pageName)`	
Output Arguments		
Input Arguments	`deviceName`	name of the device
	`pageName`	name of the page: • `wp`: working page • `rp`: reference page

Examples

3.6.2 Get Current Page (INCA-MIP Extended)

Name `IncaGetCurrentPage`

Description Gets the currently active memory page

Syntax `pageName = IncaGetCurrentPage(deviceName)`

Output Arguments

`pageName`	name of the active memory page: • `wp`: working page • `rp`: reference page

Input Arguments

`deviceName`	Name of the device

Examples

3.6.3 Check Write-Protection

Name `IncaIsPageWriteProtected`

Description Checks whether the specified memory page is write-protected

Syntax `isRW = IncaIsPageWriteProtected(deviceName, pageName)`

Output Arguments

`isRw`	• `0`: page is not write-protected • `not 0`: page is write-protected

Input Arguments

`deviceName`	name of the device
`pageName`	name of the page: • `wp`: working page • `rp` : reference page

Examples

3.6.4 Download Memory Page

Name `IncaDownloadPage`

Description Downloads the specified memory page to the control unit

Syntax `IncaDownloadPage(deviceName, pageName)`

Output Arguments

Input Arguments

`deviceName`	name of the device
`pageName`	name of the page to download • `wp`: working page • `rp`: reference page

Examples

3.6.5 Copy Memory Page

Name `IncaCopyPageFromTo`

Description Copies the specified memory page. Currently, it is only possible to copy from the reference page to the working page; other combinations of sources and targets are not supported.

Syntax
```
IncaCopyPageFromTo( deviceName,
sourcePageName, destinationPageName)
```

Output Arguments

Input Arguments

`deviceName`	name of the device
`sourcePageName`	name of the page to be copied: • `wp`: working page • `rp`: reference page
`destinationPageName`	name of the page to copy to: • `wp`: working page • `rp`: reference page

Examples

3.6.6 Download Differences

Name `IncaDownloadDifferences`

Description Loads the differences between the working page and reference page into the control unit. As with the corresponding menu option, this is only updated if the working page and reference page in the target unit match the reference page in INCA.

Syntax
```
IncaDownloadDifferences( deviceName )
```

Output Arguments

Input Arguments

`deviceName`	name of the device

Examples

3.6.7 Upload Pages (INCA-MIP Extended)

Name `IncaUploadPages`

Description Uploads reference and working page to newly created datasets. The new datasets are automatically assigned to the device.

Syntax
```
IncaUploadPages(device{,referencePage, work-
ingPage})
```

Output Arguments		
Input Arguments	`device`	name of device
	`referencePage`	Dataset name for uploaded reference page. If not specified, INCA uses a default name
	`workingPage`	Dataset name for uploaded working page. If not specified, INCA uses a default name
Examples	`IncaUploadPages('ETK:1');`	
	`IncaUploadPages('ETK:1', 'ref_1', 'work_1');`	

3.7 Application Examples

Example 1

```
% Check if working page is write-protected and
% download the page if it is write-protected
if(IncaIsPageWriteProtected (’anEtk’, ’wp’))
 IncaDownloadPage(’anEtk’, ’wp’);
end
% Switch to the working page
IncaSwitchPage( ’anEtk’, ’wp’);
```

Example 2

In the following example, the functions described above are used to read measured values from the device `MyDevice` and measure raster `10ms`. To execute this example, you must first open an experiment in INCA that includes an assigned device named `MyDevice`.

```
      % Measure the following signals
      IncaAddMeasureElement( ’MyDevice’, ’10ms’, ’Chan1’
);
      IncaAddMeasureElement( ’MyDevice’, ’10ms’, ’Chan2’
);
      IncaAddMeasureElement( ’MyDevice’, ’10ms’, ’Chan3’
);
      IncaAddMeasureElement( ’MyDevice’, ’10ms’, ’Chan4’
);
      % Now measure
      data = [];
      time = [];
      IncaShowMessages(0);
      IncaSetMeasureReadMode(0)
      IncaStartMeasurement;
      deltaT = 0;
```

```
      % Measure for 20 seconds
      while( deltaT < 20 )
            % Pause for 0.1 seconds to have more than
one
            % record -- saves processor time.
            pause(0.1)
            % Get up to 500 records for group 10ms
            [ t, d ]=IncaGetRecords( 'MyDevice', '10ms',
500 );
            % Append t and d to time and data
            data = [data; d];
            time = [time; t];
            if( length(time) )
               % Calculate time measured
               deltaT = time( length(time)) - time(1);
            end
      end
      IncaStopMeasurement;
      IncaShowMessages(1);
      % Plot the results
      plot(time, data);
```

This example uses only one measure raster. However, you can use several groups and request the data for each group independent of MATLAB.

4 Creation and Distribution of Stand-alone Executable Files

With INCA-MIP, you can create and compile m-files containing MATLAB API functions including INCA-MIP functions. The resulting stand-alone files can be executed also in environments without a MATLAB installation.

The procedure differs in some details for the different MATLAB compiler versions.

4.1 Creation and Distribution of Stand-alone Executable Files using the MATLAB R13 Compiler

Creating stand-alone executable files requires a MATLAB installation. The resulting executable, together with copies of some MATLAB and ETAS DLLs, can be used without requiring a MATLAB installation on the target system.

4.1.1 Compilation of m-Files

To compile m-Files using the MATLAB R13 compiler:

1. Execute the following command:

   ```
   mcc -m <m-file-script>
   ```

   Example:

   With the following command a stand-alone executable file is created from the file `testCase1.m`:

   ```
   mcc -m testCase1
   ```

   Result is the file `testCase1.exe`.

See your MATLAB user documentation under the keyword *MATLAB Compiler* or *`mcc`* for further settings of theMATLAB compiler.

Note

All Inca*.dll files that are used by the script as well as the incaRci2Matlab.dll must be copied to the target system where the compiled script will be executed (see "Distribution of Stand-alone Executable Files" on the next page).

Note

INCA can be controlled by only one MATLAB session at a time. Trying to control INCA simultaneously from different instances of MATLAB or stand-alone executables will be aborted with an error message.

4.1.2 Distribution of Stand-alone Executable Files

Stand-alone executable files created as described above need runtime libraries both from MATLAB and from ETAS. A MATLAB installation is not necessary.

To distribute stand-alone executable files compiled with the MATLAB R13 compiler:

1. Install the required MATLAB runtime libraries. See your MATLAB user documentation under *Distributing Stand-Alone Applications* for information on how to install the MATLAB runtime libraries.

   To install the ETAS runtime libraries, install the INCA-MIP Add-On and select **Installation into ETASData** in the installation routine (see "Installing INCA-MIP" on page 11).

or

   Copy the required files from the following locations within your MATLAB installation on your development machine :

   ```
   %MATLABDir%\bin\win32\
   incaRci2Matlab.dll
   ```

   ```
   %MatlabDir%\toolbox\matlab\
   general\Inca*.dll
   ```

2. Copy these files together with the stand-alone executable files into the same directory.

4.2 Creation and Distribution of Stand-alone Executable Files using the MATLAB R14 Compiler or Higher

Creating stand-alone executable files requires a MATLAB installation. The resulting executable can be used on the target system without a MATLAB installation or copies of additional MATLAB and ETAS DLLs.

4.2.1 Compilation of m-Files

To compile m-Files using the MATLAB R14 compiler:

1. Copy all Inca*.dll files into the current working directory.
2. Execute the following command:

   ```
   mcc -m <m-file-script> -a incaRci2Matlab.dll
   ```

   Example:

   With the following command a stand-alone executable file is created from the file `testCase2.m`:

   ```
   mcc -m testCase2 -a incaRci2Matlab.dll
   ```

   Result is the file `testCase2.exe`.

The MATLAB R14 compiler creates a container with all MEX function DLLs and dependent DLLs which are needed to execute the compiled MATLAB script. All Inca*.dll files that are used by the script as well as the incaRci2Matlab.dll have to be part of this container.

When the compiled script is executed, the DLLs do not need to be present on the system.

See your MATLAB user documentation under the keyword *MATLAB Compiler* or `mcc` for further settings of the MATLAB compiler.

Note

INCA can be controlled by only one MATLAB session at a time. Trying to control INCA simultaneously from different instances of MATLAB or stand-alone executables will be aborted with an error message.

Note

With MATLAB R14 SP3 (Version 7.1) or higher, the INCA MEX function DLLs have the extension *.mexw32.

4.2.2 Distribution of Stand-alone Executable Files

Executing standalone executable files that were compiled using the MATLAB compiler R14 only require the executable itself. A MATLAB installation or copies of MATLAB libraries are not required.

To distribute stand-alone executable files compiled with the MATLAB R14 compiler:

- Simply copy the stand-alone executable files to the target system.

Afterwards you can simply execute them; no further steps are required.

5 ETAS Contact Addresses

ETAS Headquarter

ETAS GmbH

Borsigstraße 24	Phone:	+49 711 3423-0
70469 Stuttgart	FAX:	+49 711 3423-2106
Germany	Internet:	www.etas.com

ETAS Subsidiaries and Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

ETAS subsidiaries	Internet:	www.etas.com/en/contact.php
ETAS technical support:	Internet:	www.etas.com/en/hotlines.php

Index