The New Version

EASY-ROB™ V7.6

December 2018

Version 1.6

EASY-ROB™

Table of content

EASY-ROB™ V7.6 Update

Dear EASY-ROB™ Community!

We are happy to introduce the new EASY-ROB™ version 7.6 and, as always, you will find the highlights here in the quick overview:

- **Robot Library- Top-subject**
  The EASY-ROB™ robot library is growing continuously -With version 7.6 more than 20 new models will be published. Not only single models, but also entire series were integrated by our employees for example from KUKA the Cybertech series, from Stäubli the TX-2 series with CS9 controller and from Universal Robots the E series. Of course, many models of other manufacturers have been integrated, too.
- **Modifying of 3D geometries during simulation execution**
  Imported geometries can be modified using the API of the Robotics Framework (DLL) and Robotics Simulator Professional (Exe) during the simulation runtime. This makes it possible to create even more process-oriented simulations.
- **EASY-ROB™ AutoPath™**
  Collision-free path planning
  EASY-ROB™ AutoPath™ creates collision-free paths. The automatic calculation greatly simplifies the work of the operator. Let your software benefit from the new way of programming.
- **EASY-ROB™ Itself Collision**
  The Itself Collision allows to check the collision of the robot with itself, which was otherwise only secured by the settings of the travel ranges.
- **Secure * .cel and * .rob files by the EASY-ROB ™ encryption and OwnerKey**
  From now on, you can protect your created * .cel and * .rob files from stranger eyes. You either use the EASY-ROB™ encryption or even combine it with your own private "top secret"-usr_ownerkey.
  One thing is certain: this will keep your data safe!
- **ERK News: ToolPath, ToolBox and API-Post-Process**
  The EASY-ROB™ Robotics Simulation Kernel has been massively expanded. ToolPath offers another method for processing robot target positions. Next there is the new powerful Option ToolBox and a new interface API-Post-Process.

Effective immediately, all customers with a valid v7.6 license or software maintenance agreement will receive the new EASY-ROB™ Version 7.6 free of charge.

For customers with older versions it is possible to purchase an update. Please contact our sales department under +49 6192 921 70 79 or sales@easy-rob.com.

We would like to thank you in advance for your suggestions.

Many Thanks!

Your EASY-ROB Team

Changing of 3D geometries during simulation execution

Previously, 3D geometries (such as STL and IGP files) could only be manipulated during import. After that, it was no longer possible to change these geometries in shape (e.g. scaling).

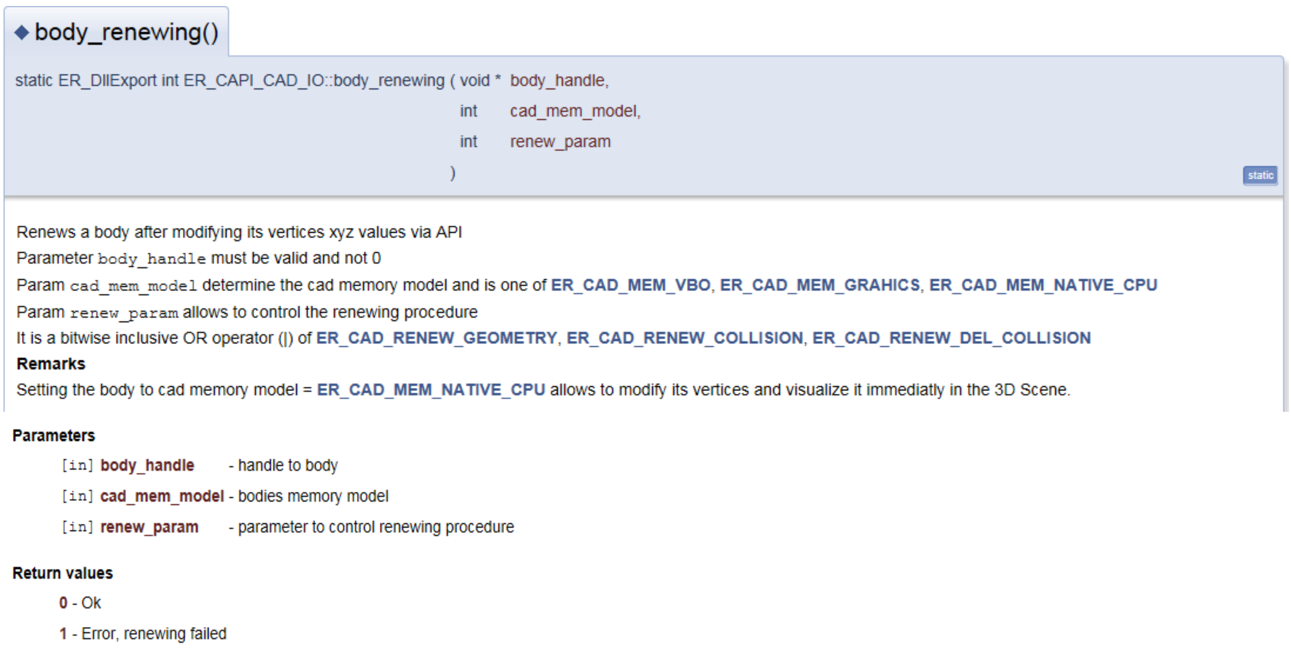
With a new API method, geometries can now be changed during the simulation runtime, in which the points/vertices are "moved". This can e.g. make sense to represent a bending process, so that the robot in each process step moves along on a changing work piece, or simply to scale a geometry just.

For each geometry a memory model is stored:

- CAD_MEM_VBO
  Geometry has been optimized and loaded into the graphics card. (Vertex Buffer Objects - VBO)
- CAD_MEM_GRAPHICS
  Geometry has been optimized and implemented internally by vertex arrays, not VBO
- CAD_MEM_NATIVE_CPU
  Geometry not optimized remains in original "native" state.

New API-method "body_renewing()"

The method "*ER_CAPI_CAD_IO :: body_renewing ()*" allows geometries to be renewed and transferred to the three previously mentioned memory models. It can also be decided whether the geometry should be included in the collision list, depending on the performance and application. This is done by means of the parameter "*renew_param*".



◆ body_renewing()

```
static ER_DllExport int ER_CAPI_CAD_IO::body_renewing ( void *  body_handle,
                                                        int     cad_mem_model,
                                                        int     renew_param
                                                      )
```

static

Renews a body after modifying its vertices xyz values via API
Parameter `body_handle` must be valid and not 0
Param `cad_mem_model` determine the cad memory model and is one of **ER_CAD_MEM_VBO**, **ER_CAD_MEM_GRAHICS**, **ER_CAD_MEM_NATIVE_CPU**
Param `renew_param` allows to control the renewing procedure
It is a bitwise inclusive OR operator (|) of **ER_CAD_RENEW_GEOMETRY**, **ER_CAD_RENEW_COLLISION**, **ER_CAD_RENEW_DEL_COLLISION**

Remarks
Setting the body to cad memory model = **ER_CAD_MEM_NATIVE_CPU** allows to modify its vertices and visualize it immediatly in the 3D Scene.

Parameters

[in]	**body_handle**	- handle to body
[in]	**cad_mem_model**	- bodies memory model
[in]	**renew_param**	- parameter to control renewing procedure

Return values

0 - Ok
1 - Error, renewing failed

In the storage model CAD_MEM_NATIVE_CPU, the vertices can now be changed "directly" via the API, without a "*body_renewing ()*" being called again. In this case, no collisions etc. should be checked.

This novelty is now available for the Robotics Framework (DLL) and the Robotics Simulator Professional (Exe).

Geometry-Import: Optimization in three steps

1. **Step**
   During import of geometries, these are stored in the first step 1:1 and can directly, i.e. without scaling, be visualized. Bounding boxes are also calculated here and the geometries are located in the memory model **CAD_MEM_NATIVE_CPU**.

2. **Step**
   In the second step the geometries or their objects are optimized. Here vertices e.g. are reduced and generated for the OpenGL rendering arrays. The geometries are now available in the memory model **CAD_MEM_GRAFICS**.

3. **Step**
   In the final step, these arrays are loaded into the graphics card (Vertex Buffer Objects - VBO), if possible. Here operates the memory model **CAD_MEM_VBO**.

Programming example for "body_renewing()"

In the following simple example for a current geometry the z-values of all vertices are first doubled and halved in the next call. Parameter "*renew_param*" becomes = *ER_CAD_RENEW_GEOMETRY* | *ER_CAD_RENEW_COLLISION* will be set so that collision can also be tested for the changed geometry. The graphical update visualizes the result.

```
// Example:
void *body_hnd = er_cad_io.inq_body_handle_current(); // get body handle of current group
int n_obj = er_cad_io.inq_body_n_obj(body_hnd); // get number of objects within body
er_user_io_dialog_info_line_msg(0,"cBody with n_obj = %d",n_obj);
int z_sign=-1;
z_sign = -z_sign;
for (int i_obj=0;i_obj<n_obj;++i_obj)   // go for all objects
{
        void *obj_hnd = er_cad_io.inq_body_obj_handle(body_hnd,i_obj); // get object handle
        int n_pnt = *er_cad_io.inq_body_obj_n_point(obj_hnd);   // get number of vertices of object
        for (int i_pnt=0;i_pnt<n_pnt;++i_pnt)   // go for all vertices
        {
                float *p = er_cad_io.inq_body_obj_points(obj_hnd,i_pnt);
                p[2] *= z_sign>0 ? 2.0f : 0.5f; // change z-value, just double or halve
        }
}

// Renew to see the effect, refresh complete geometry and the collision model as well
int cad_mem_model = er_cad_io.inq_body_cad_mem_model(body_hnd); // get current cad memory model
cad_mem_model = ER_CAD_MEM_NATIVE_CPU;  // change cad memory model to ER_CAD_MEM_NATIVE_CPU

int     renew_param = ER_CAD_RENEW_GEOMETRY | ER_CAD_RENEW_COLLISION;

int res = er_cad_io.body_renewing(body_hnd,cad_mem_model,renew_param);
if (res)
        _info_line_msg(0,"ERROR:  body_renewing() failed");

// Visualize here
er_sys_view.grf_update_export();
...
```

Innovations for EASY-ROB™ Robotics Simulation Kernel (ERK)

In this release, special attention was paid to the Robotics Simulation Kernel (ERK) and its simplified integration into technology-based software applications.

ToolPath - Easily generates trajectories

The ToolPath is a trajectory for robots or devices and consists of target positions (targets) and can be generated in the ERK (method class: ERK_CAPI_TOOLPATH). All targets include attributes such as name, ID, index, speed, acceleration, external axis values, tool data and -names, reference coordinate systems, synchronization flags, motion types (PTP, SLEW, LIN, CIRC), instructions, etc.

The number of ToolPaths and the associated targets is unlimited. ToolPaths are uniquely assigned to the robot, can be created, deleted and moved in order.
Target positions can be completely manipulated using extensive methods, i.e. you can delete, move, ignore, create new ones, make changes in the order and access all data.

The implicit use of ToolPath templates makes it much easier to add targets, so you only need to specify data such as position and speed.

PTP-Move:	*erSetTargetLocation_Move_Joint (tarloc_hnd, CartPosvec)*
JointAbs-Move:	*erSetTargetLocation_Move_JointAbs (tarloc_hnd, JointPos, speed_pERKent)*
LIN-Move:	*erSetTargetLocation_Move_CIRC (tarloc_hnd, CartPosVec)*
CIRC-Move:	*erSetTargetLocation_Move_LIN (tarloc_hnd, CartPosVecVia, CartPosVec)*

If the ToolPath is completely defined by the host application, a single call of the method *erGET_NEXT_TOOLPATH_STEP()* is sufficient to start the interpolation along the whole ToolPath. In each interpolation step, all states of the robot with its synchronized devices (positioner, tracking axis) can be accessed and displayed, for example, in a 3D scene.

The advantages are apparent.

Previously, the target positions in the ERK were transferred individually by the host application via the methods *SET_NEXT_TARGET* and *GET_NEXT_STEP* and processed individually by the ERK.The administrative effort, especially for transitions to the next target ("need more data", "target reached") was expensive. This resulted in increased kernel integration and increased implementation complexity.

On the other hand, a certain robotics know-how was assumed, since different attributes had to be selected for the target positions for the correct calculation by the ERK.

With the new ToolPath functionality not only a single robot target position but an entire trajectory (= ToolPath) can be transferred to the ERK. The axis angles and other results (axis speeds, positions of the TCP, TCP speed, path length to the destination, etc.) are now calculated for the interpolation with a single call of the method *erGET_NEXT_TOOLPATH_STEP()* for all contained target positions in one run and returned closed. Thus, on the customer side, the integration of the ERK into the host application results in a noticeably reduced effort.

Additional benefit

Furthermore, in each individual interpolation step, the robot configuration and the axis values of the robot are calculated and stored in the target. In particular, the new function "API-Post-Process" benefits from this. It can convert the content of the entire ToolPaths into a robot program. The ToolPath thus forms a neutral level from which a robot program can be generated in a suitable syntax for each robot controller. The new function "**ERK-ToolBox**" also accesses the ToolPath and thus has the possibility to calculate external axes according to given constraint rules.

It is also possible to transfer immediately multiple trajectories and the ERK calculates all required values at once. The path order in turn can be manipulated. This way individual paths can be de- and reactivated.

Backward compatibility

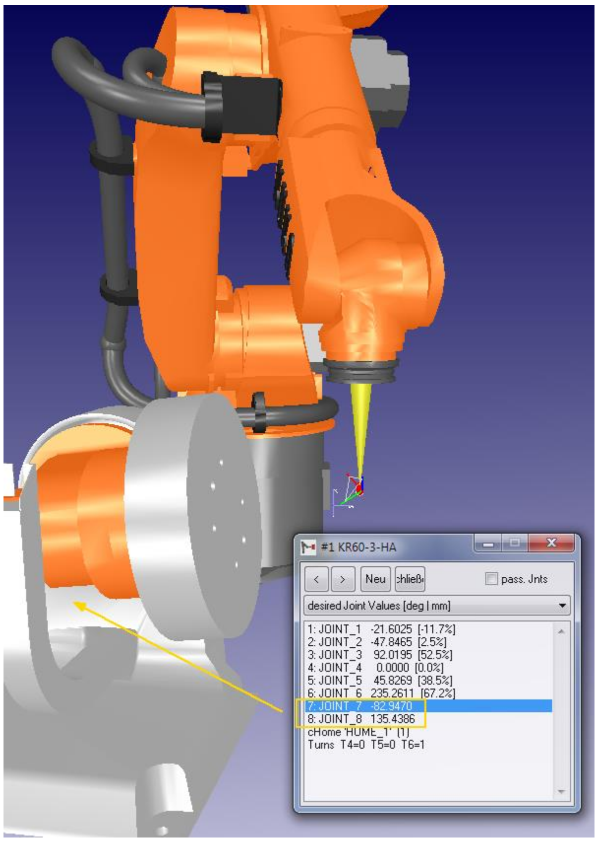
Of course, all implementations that do not use the ToolPath will continue to be 100% supported and maintained in the future. ToolPath offers only a simplification and creates new possibilities and functions, which should support the slightly less experienced robotics user.

ToolBox – a powerful extension

With the new "ERK ToolBox" functionality a powerful extension will be offered to the EASY-ROB™ Robotics Simulation Kernel.

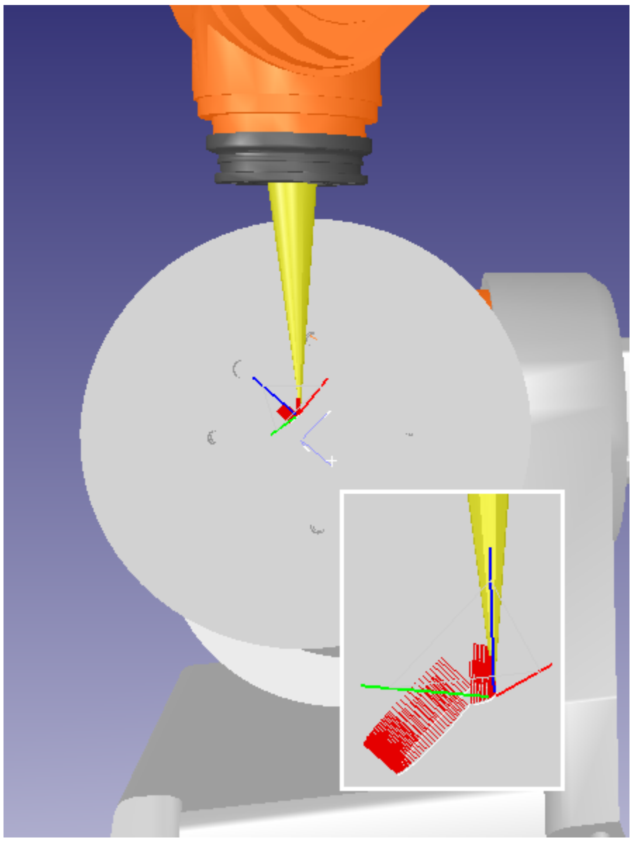
Based on a defined ToolPath, various calculations can be carried out using the ERK ToolBox.

In the current functionality, the external axis values for a single- or two-axis positioner are calculated as a function of specified constraints. A typical constraint is the specification of the tool orientation with regard to the work piece surface, so it will always be vertical, for example, see picture.



```
1: JOINT_1   -21.6025  [-11.7%]
2: JOINT_2   -47.8465  [2.5%]
3: JOINT_3    92.0195  [52.5%]
4: JOINT_4     0.0000  [0.0%]
5: JOINT_5    45.8269  [38.5%]
6: JOINT_6   235.2611  [67.2%]
7: JOINT_7   -82.9470
8: JOINT_8   135.4386
cHome 'HOME_1' (1)
Turns  T4=0  T5=0  T6=1
```

Robot with a two-axis positioner

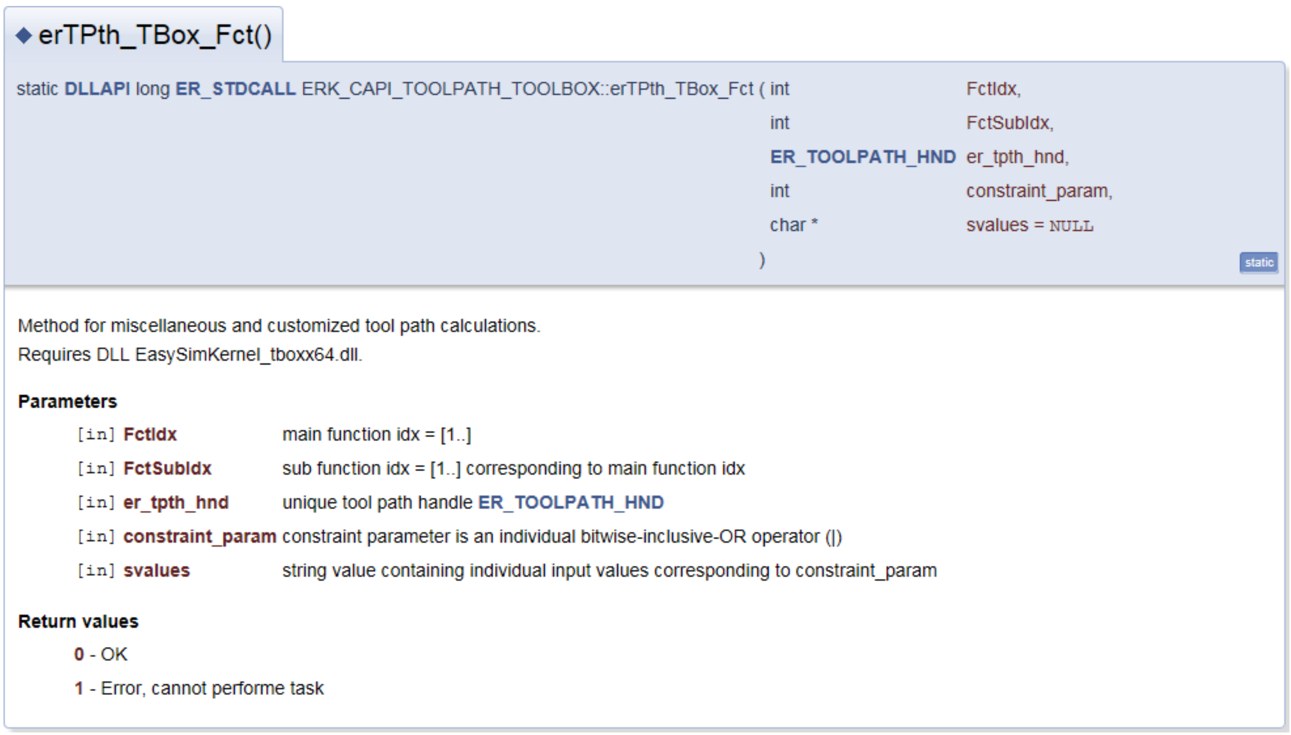

Constraint: Perpendicular to the work piece surface

During the calculation, the current system configuration (position and linkage of industrial robots, linear unit and external positioners) is automatically accessed.

Advantages for the user

- Complex calculations are done by Robotics Simulation Kernel respectively ToolBox.
- ToolBox provides a powerful solution instead of costly in-house development.
- OEM customers can respond flexibly to a wide variety of user requirements (customization).
- The ToolBox functionality is outsourced in a DLL "*EasySimKernel_tboxx64.dll*" so that end customers can be supplied individually.
- ToolBox is scalable
- Your EASY-ROB team is available for your support and individual requirements

The ToolBox calculation is called from the method class: ERK_CAPI_TOOLPATH_TOOLBOX with *erTPth_TBoxFct()*. The desired constraints are transferred with the ToolPath handle (ER_TOOLPATH_HND).



◆ erTPth_TBox_Fct()

```
static DLLAPI long ER_STDCALL ERK_CAPI_TOOLPATH_TOOLBOX::erTPth_TBox_Fct ( int              FctIdx,
                                                                            int              FctSubIdx,
                                                                            ER_TOOLPATH_HND  er_tpth_hnd,
                                                                            int              constraint_param,
                                                                            char *           svalues = NULL
                                                                          )
```
static

Method for miscellaneous and customized tool path calculations.
Requires DLL EasySimKernel_tboxx64.dll.

Parameters

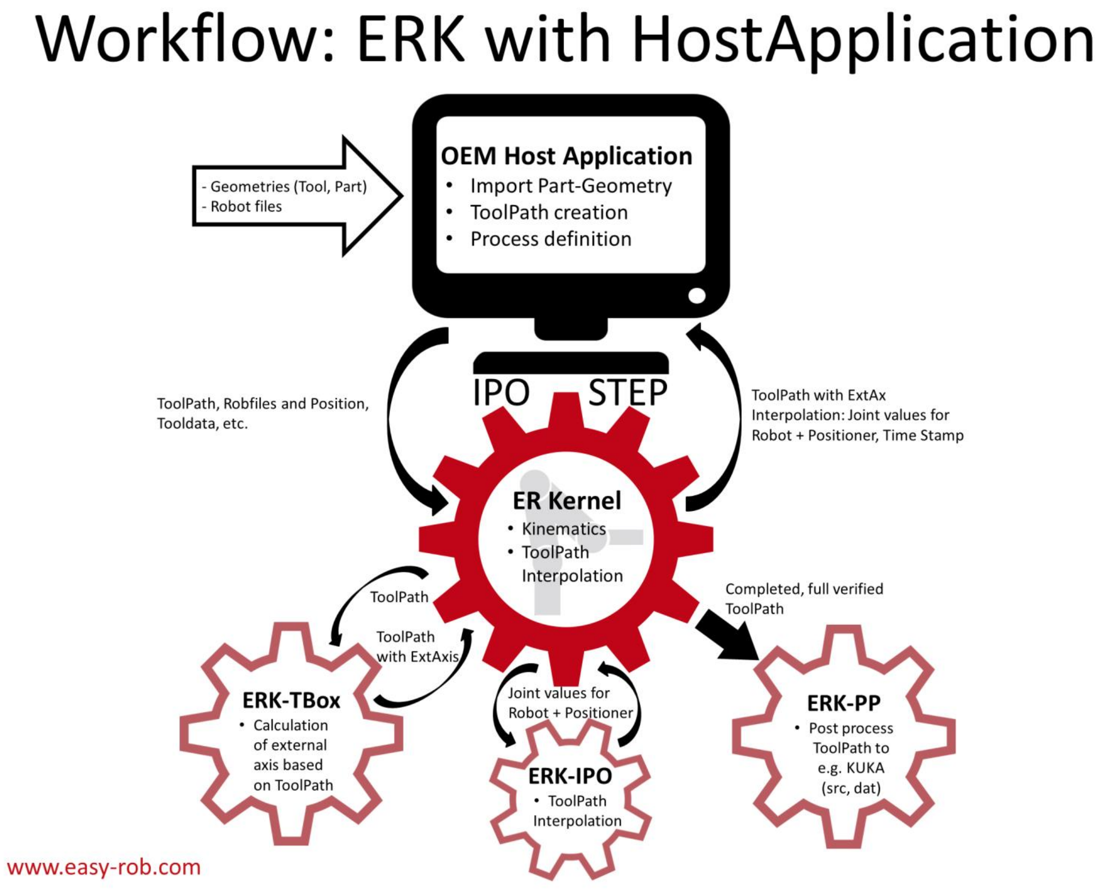
[in]	**FctIdx**	main function idx = [1..]
[in]	**FctSubIdx**	sub function idx = [1..] corresponding to main function idx
[in]	**er_tpth_hnd**	unique tool path handle **ER_TOOLPATH_HND**
[in]	**constraint_param**	constraint parameter is an individual bitwise-inclusive-OR operator (|)
[in]	**svalues**	string value containing individual input values corresponding to constraint_param

Return values

0 - OK
1 - Error, cannot performe task

Schematic diagramm: ERK workflow with additional APIs and options

For a better understanding and overview of the interplay of the modules, the following workflow should contribute.

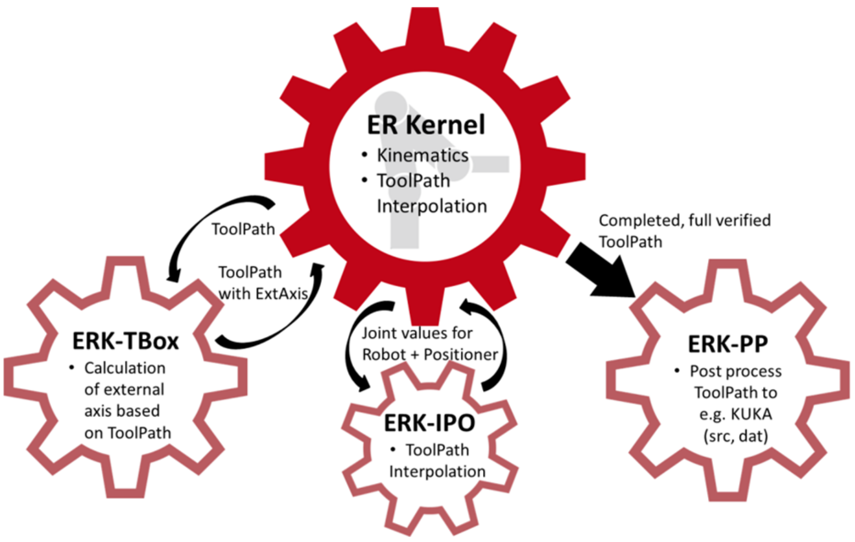

Workflow: ERK with HostApplication
- Geometries (Tool, Part)
- Robot files
OEM Host Application
• Import Part-Geometry
• ToolPath creation
• Process definition
ToolPath, Robfiles and Position, Tooldata, etc.
IPO
STEP
ToolPath with ExtAx Interpolation: Joint values for Robot + Positioner, Time Stamp
ER Kernel
• Kinematics
• ToolPath Interpolation
ToolPath
ToolPath with ExtAxis
Completed, full verified ToolPath
ERK-TBox
• Calculation of external axis based on ToolPath
Joint values for Robot + Positioner
ERK-IPO
• ToolPath Interpolation
ERK-PP
• Post process ToolPath to e.g. KUKA (src, dat)
www.easy-rob.com

Typical procedure out of the host application

1. The host application loads the robot files, positions and links them
2. The host application creates/renews a ToolPath including process definitions
3. Calling **ERK-ToolBox** to calculate external axes *(erTPth_TBoxFct ())*.
4. Starting interpolation with **ERK-IPO** *(erGET_NEXT_TOOLPATH_STEP ())*
   Parallel visualization in each interpolation step, with simultaneous collision detection.
5. Verification of the robot movement; Goal: Error-free ToolPath, possibly adaptation, optimization → 2
6. Creation of robot program, call **ERK-PP** (API-Post-Process - *erTPth_PostProc ())*

API-Post-Process – create robot programs

Another powerful addition to the EASY-ROB™ Robotics Simulation Kernel is the API-Post-Process (**ERK-PP**) to create robot programs. The basic functionality was taken over by the EASY-ROB ™ Robotics Simulator Professional and the EASY-ROB™ Robotics Framework.


ER Kernel
• Kinematics
• ToolPath Interpolation
ToolPath
ToolPath with ExtAxis
Completed, full verified ToolPath
ERK-TBox
• Calculation of external axis based on ToolPath
Joint values for Robot + Positioner
ERK-IPO
• ToolPath Interpolation
ERK-PP
• Post process ToolPath to e.g. KUKA (src, dat)

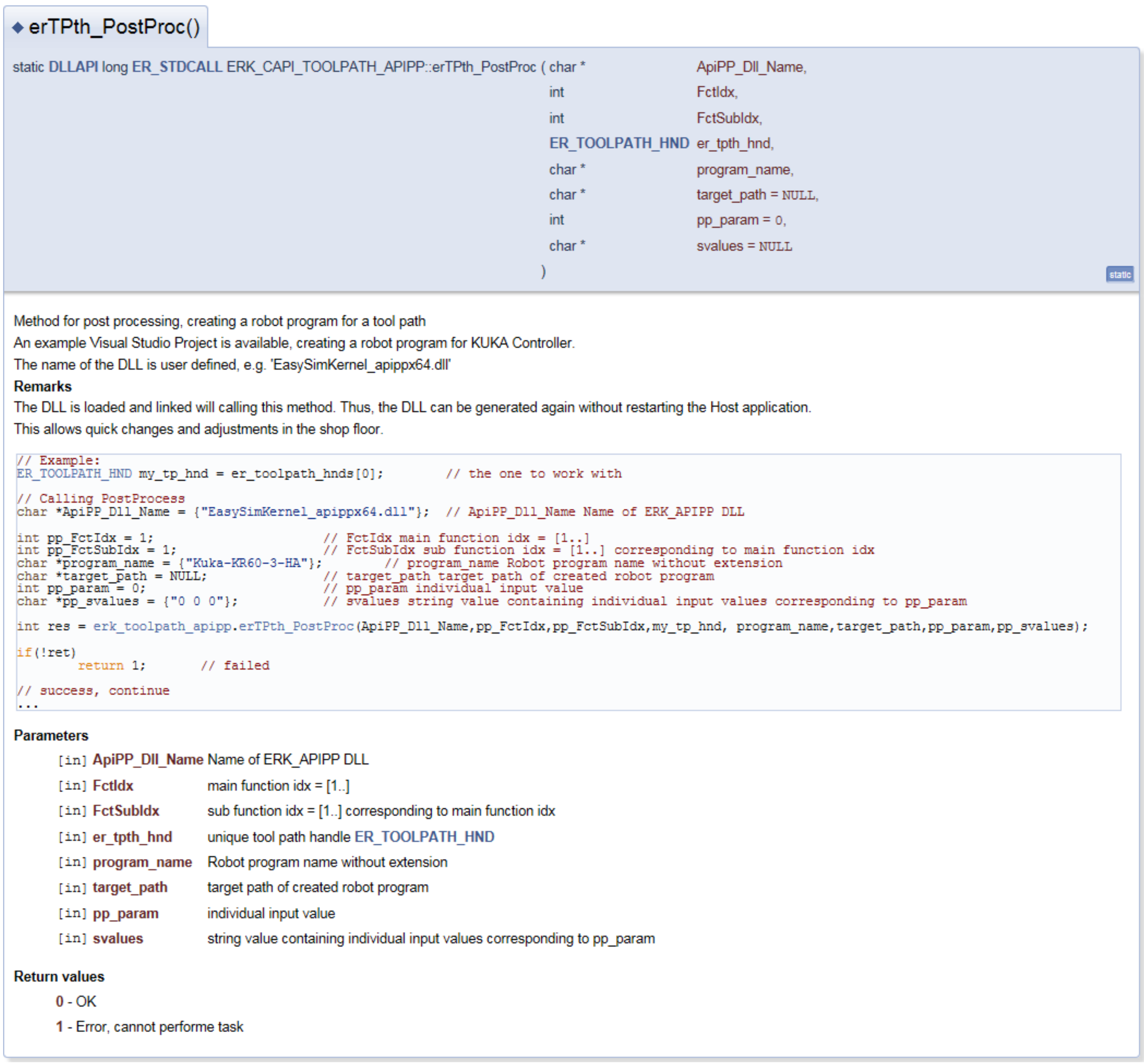
In contrast to this, the ERK-PP accesses the ToolPath and converts the individual instructions into a robot program. It should be noted at this point that the ToolPath contains all necessary data after a successful interpolation (**ERK-IPO**) to generate a target program. These include, for example, the configuration and axis angles of the robot in each target position (target) which are needed to properly write out the "signs" and "turns". Furthermore, each target also contains individual instructions, which for example can be native process-dependent statements.

With the successful concept of an **API**, the OEM partner (manufacturer of the host application) can individually adapt the generated robot program to the process and robot control, which offers maximum flexibility. The process know-how thus always remains with the manufacturer.

Advantages:

- Create, adapt and expand robot programs individually
- The number of possible post-processors is unlimited
- A manageable post-processor example for KUKA KRL makes customization easy for other languages (**Fanuc / LS, ABB / Rapid, Universal Robots / URScript, Yaskawa / InformII, Comau / PDL2**).
- Process know-how is retained by the manufacturer
- Programming examples avoid costly and time-consuming new developments
- Your EASY-ROB team is available for your support and individual requirements

The post processor is called from the method class: ERK_CAPI_TOOLPATH_APIPP with *erTPth_PostProc()*. In this case, further settings are transferred with the ToolPath Handle (ER_TOOLPATH_HND), such as Name of the generated robot program file and destination directory. The parameters *FctIdx* and *FctSubIdx* can be used to call the various postprocessors in the DLL.



◆ erTPth_PostProc()

```
static DLLAPI long ER_STDCALL ERK_CAPI_TOOLPATH_APIPP::erTPth_PostProc ( char *          ApiPP_Dll_Name,
                                                                          int             FctIdx,
                                                                          int             FctSubIdx,
                                                                          ER_TOOLPATH_HND er_tpth_hnd,
                                                                          char *          program_name,
                                                                          char *          target_path = NULL,
                                                                          int             pp_param = 0,
                                                                          char *          svalues = NULL
                                                                        )
```

static

Method for post processing, creating a robot program for a tool path

An example Visual Studio Project is available, creating a robot program for KUKA Controller.

The name of the DLL is user defined, e.g. 'EasySimKernel_apippx64.dll'

Remarks

The DLL is loaded and linked will calling this method. Thus, the DLL can be generated again without restarting the Host application.

This allows quick changes and adjustments in the shop floor.

```
// Example:
ER_TOOLPATH_HND my_tp_hnd = er_toolpath_hnds[0];        // the one to work with

// Calling PostProcess
char *ApiPP_Dll_Name = {"EasySimKernel_apippx64.dll"};  // ApiPP_Dll_Name Name of ERK_APIPP DLL

int pp_FctIdx = 1;                          // FctIdx main function idx = [1..]
int pp_FctSubIdx = 1;                       // FctSubIdx sub function idx = [1..] corresponding to main function idx
char *program_name = {"Kuka-KR60-3-HA"};          // program_name Robot program name without extension
char *target_path = NULL;                   // target_path target path of created robot program
int pp_param = 0;                           // pp_param individual input value
char *pp_svalues = {"0 0 0"};               // svalues string value containing individual input values corresponding to pp_param

int res = erk_toolpath_apipp.erTPth_PostProc(ApiPP_Dll_Name,pp_FctIdx,pp_FctSubIdx,my_tp_hnd, program_name,target_path,pp_param,pp_svalues);

if(!ret)
        return 1;       // failed

// success, continue
...
```

Parameters

[in]	**ApiPP_Dll_Name**	Name of ERK_APIPP DLL
[in]	**FctIdx**	main function idx = [1..]
[in]	**FctSubIdx**	sub function idx = [1..] corresponding to main function idx
[in]	**er_tpth_hnd**	unique tool path handle ER_TOOLPATH_HND
[in]	**program_name**	Robot program name without extension
[in]	**target_path**	target path of created robot program
[in]	**pp_param**	individual input value
[in]	**svalues**	string value containing individual input values corresponding to pp_param

Return values

0	OK
1	Error, cannot performe task

Last but not least - the ERK-PP is loaded dynamically

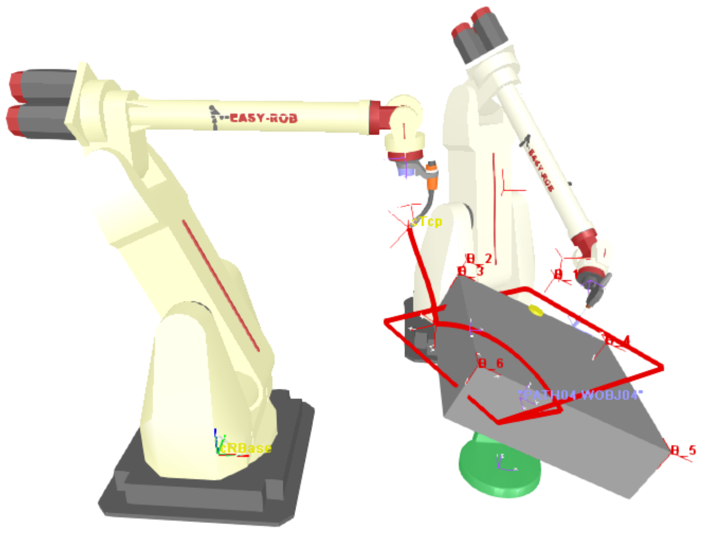
When calling the post-processor, the DLL "*EasySimKernel_apippx64.dll*" is loaded dynamically and released at the end. Thus, while working in the host application, the ERK-PP can be adapted without restarting the host application, loading the scene and making all other settings. This saves considerable time during commissioning on site.

AutoPath™ - now a stand-alone-module

AutoPath™ collision-free path planning functionality is now available as a stand-alone module in addition to EASY-ROB™ Robotics Simulator Professional (Single- and Multi-Robot) and the Robotics Simulation Framework, as well as the EASY-ROB™ Robotics Simulation Kernel (ERK).

In this way, EASY-ROB Software GmbH continues to pursue its goal of being able to offer a large, always fitting family of harmonized software modules for robot simulation. For all EASY-ROB™ customers, this means being well-prepared for all future requirements.

As a target group, the AutoPath™ module clearly aims all customers of the ERK who, in addition to reliable robotics simulation, want to simplify the work of the operator considerably. Or all interested companies who already use their own motion planning and execution and want to make a big step towards *Next Level Robot Programming*, but have so far spared such a costly in-house development.



Example screenshot of a Multi-Robot application

Callback function

After the successful separation of algorithm and GUI additional function were donated to the AutoPath™. Callback features allow AutoPath™ to be controlled intelligently by the host application (your software application):
start and end positions are specified and the calculated result are the respective intermediate positions (WayPoints) in order to move free of collision from the start to the end position.

The calculation of the intermediate positions can now be influenced by callback function. Thus, for example a Cartesian work space can be specified, in which the TCP position of the robot should move. After these geometric conditions et al by axis configurations are fulfilled via callback function, then the collision check is carried out, if desired also via EASY-ROB™ modules.

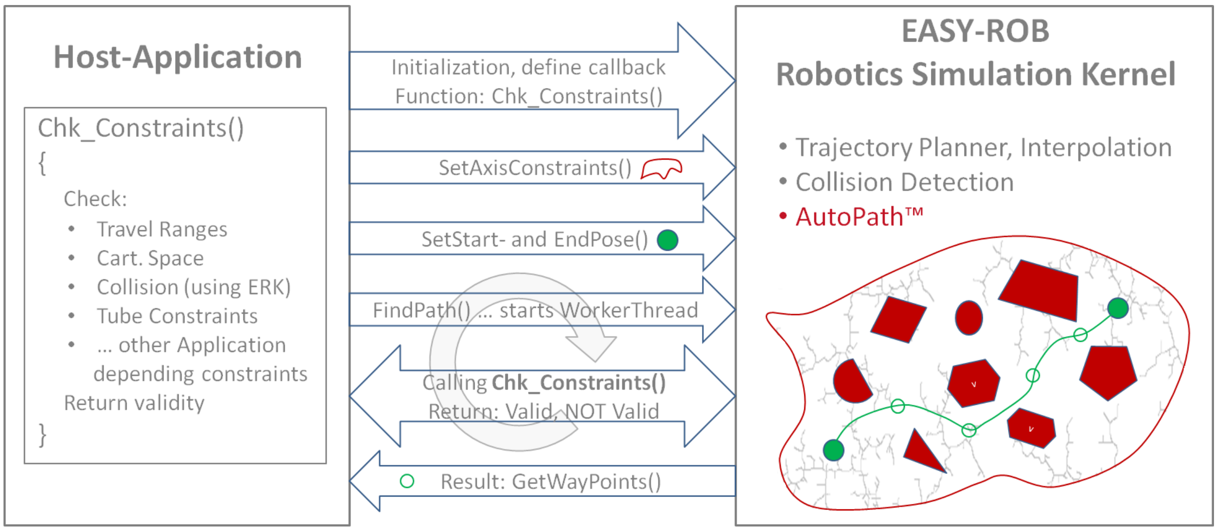
Direct reference to your own application

The callback function can be used to directly relate to robotics. The intermediate positions can be calculated in the context of your own application and quickly integrate into your software.

On the next page, find out what they are and what they do. Of course, AutoPath™ is driven by a meaningful API, described by the Doxygen documentation.

For further questions or further information please contact our sales department at sales@easy-rob.com

Communication: Host Application - ERK with AutoPath™



Sequence of events

- Initializing:
  Definition of the callback function and the travel ranges of the robots / kinematics.
  The callback function, which is part of the host application, defines the constraint conditions, which are checked according to the technical requirement and priority.
- Set the start and end position, which must meet the constraint conditions.
- FindPath:
  Start of the AutoPath™ algorithm in the worker thread, call the callback function cyclically to check the constraint conditions. → Return: Valid or NOT Valid
- Result → WayPoints
  If a collision-free path was found that meets all constraint conditions from the callback function, a list of WayPoints can be retrieved. WayPoints are axis-specific intermediate points that have to be interpolated with fully synchronous PTP.

Advantages and Application Possibilities

Advantages

- Automatic calculation of TagPoints and axis configurations
- Callback functions
- Easy integration into technology driven software solutions
- API available

Application Possibilities

- Industrial robots
- Service robots
- Animation and simulation
- Motion planning
- Assembly tests
- Offline programming
- Measurement protocols
- Autonomous driving
- Industry-independent

Feature Overview

Automatic calculation of TagPoints

- Due to the automatic calculation of TagPoints obstacles are going to be avoided
- with a sophisticated RRT search algorithm

Callback Functions

- Cartesian Space
- Collision
- Tube Constraints
- Travel Ranges
- Let your individual constraints be taken into account

Calculation of axis configurations

- also supplies the axis configuration for each TagPoint

Specification of travel ranges

- Axis constraints for the travel ranges can be defined via API

API

- C/C++ and C# Method class ERAuto_CAPI

Integration

- Detailed doxygen documentation
- Programming examples for MS Visual Studio® C/C++ and C#

Set private user key (OwnerKey)

Work cell files (* .cel), robot assemblies (* .ras), and kinematics (* .rob) can also be saved as ASCII files for better and faster editing. Thus, various information for example about coordinate systems or axis lengths are contained in plain text. For tight-loop iteration steps in a hot project phase, the ability to view these files externally via text editors, without EASY-ROB™, to change, very helpful.

Perform encryption

But what if sensitive data or trade secrets are included in the end?

To protect your work cells as well as your created kinematic structures from "strangers eyes", you can now encrypt your created files with the new "*usr_ownerkey*" encryption called at EASY-ROB™.

With the "*usr_ownerkey*" a created *.cel, *.ras or *.rob file can be encrypted and saved as a binary file, making any information about your project or kinematics unreadable to third parties.

The "*usr_ownerkey*" can be set individually. This can controll within a company, for example which user should see appropriate information and, above all, a vulnerability in the exchange of files is closed, if those files should get to other companies or even the competition.

Decryption- Undo the Encryption

Of course, previously encrypted *.cel, *.ras or *.rob files can be decrypted again. You just need your personal "*usr_ownerkey*" and immediately your files can be saved out of EASY-ROB™ as ASCII files.

Where to find the usr_ownerkey?

The *"usr_ownerkey"* is stored encrypted in an *.enc file, for example "*MyCompany-ownerkey.enc*". The location and name of the file can be specified in the configuration file "config.dat" and loaded immediately when starting EASY-ROB™. By default, it is the user folder:

- „USR_DIR=“, bzw. "USR_OWNERKEY_DIR="

The folder is created and set by default in the EASY-ROB ™ installation directory as a user folder, but can then be modified or changed.

For more information about customizing the user folders and paths, see the update description of version 7.3, page 6, filename "Update-ER_v7305-2017_EN.pdf".

OwnerKey only in EASY-ROB™ directly changeable

So that third parties from the outside cannot change the *"usr_ownerkey"*, set up or changes must be done via the EASY-ROB™ user interface:

- Menu Aux → Settings → Encryption → OwnerKey Definition

Change or create a new OwnerKey

When changing an existing OwnerKey, the set OwnerKey must always be entered first for verification.

The first time it is used after a successful installation is:

- 4711

and accesses the OwnerKey-file:

- usr_ownerkey_4711.enc

Subsequently, a new OwnerKey can be entered and stored in a new OwnerKey -file for example "*MyCompany-ownerkey.enc*".

Note: Please do NOT write your OwnerKey in the OwnerKey -file name.

Encryption *without* a OwnerKey

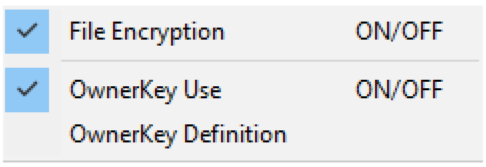
If you want to encrypt your *.cel, *.ras or *.rob files without a OwnerKey, go through the EASY-ROB™ interface and enable encryption as follows:

- Menu Aux → Settings → Encryption → File Encryption

Note: Although your file is now encrypted, anyone with an EASY-ROB™ full version can decrypt your file.

Encryption *with* a OwnerKey

If you like to encrypt your *.cel, * .ras or *.rob files with OwnerKey, both check boxes must be set:



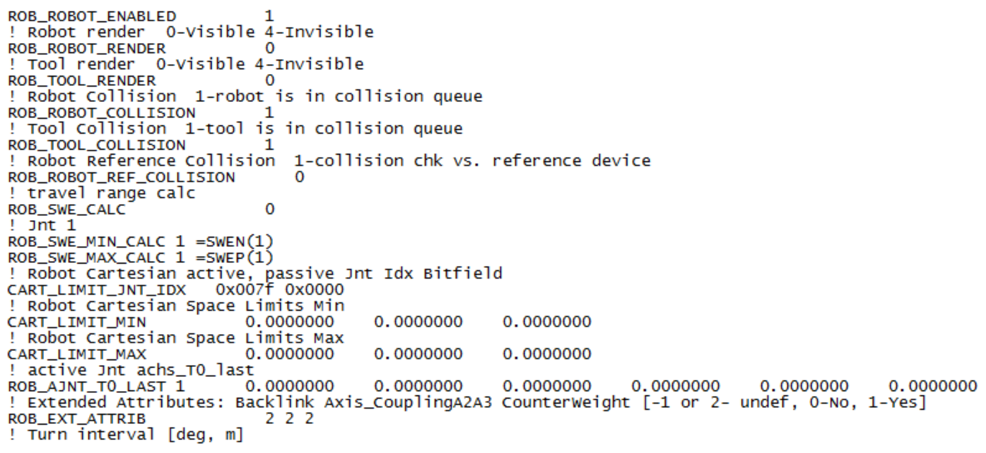
- Menu Aux → Settings → Encryption → File Encryption

and

- Menu Aux → Settings → Encryption → OwnerKey Use

Example of an encryption

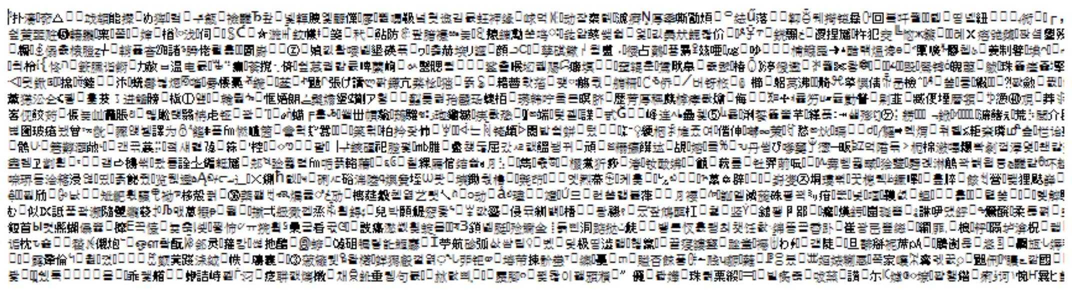
Without encryption



```
ROB_ROBOT_ENABLED           1
! Robot render   0-Visible 4-Invisible
ROB_ROBOT_RENDER            0
! Tool render   0-Visible 4-Invisible
ROB_TOOL_RENDER             0
! Robot Collision   1-robot is in collision queue
ROB_ROBOT_COLLISION         1
! Tool Collision   1-tool is in collision queue
ROB_TOOL_COLLISION          1
! Robot Reference Collision   1-collision chk vs. reference device
ROB_ROBOT_REF_COLLISION        0
! travel range calc
ROB_SWE_CALC                0
! Jnt 1
ROB_SWE_MIN_CALC 1 =SWEN(1)
ROB_SWE_MAX_CALC 1 =SWEP(1)
! Robot Cartesian active, passive Jnt Idx Bitfield
CART_LIMIT_JNT_IDX    0x007f 0x0000
! Robot Cartesian Space Limits Min
CART_LIMIT_MIN          0.0000000    0.0000000    0.0000000
! Robot Cartesian Space Limits Max
CART_LIMIT_MAX          0.0000000    0.0000000    0.0000000
! active Jnt achs_TO_last
ROB_AJNT_TO_LAST 1      0.0000000    0.0000000    0.0000000    0.0000000    0.0000000    0.0000000
! Extended Attributes: Backlink Axis_CouplingA2A3 Counterweight [-1 or 2- undef, 0-No, 1-Yes]
ROB_EXT_ATTRIB            2 2 2
! Turn interval [deg, m]
```

Sample screenshot of an unencrypted *.cel file (ascii)

With encryption



[illegible]

Sample screenshot of an encrypted *.cel file

Safety advice

- Please keep your “*usr_ownerkey”* very carefully. If the “*usr_ownerkey”* is lost, your created data can no longer be read as plain text or be decrypted.
- Please write down your “*usr_ownerkey”* separately and do not rely solely on the “*usr_ownerkey”*-file as it could be deleted if necessary.
- With your “*usr_ownerkey”* you can always create a new “*usr_ownerkey”*-file.

3D-PDF Export with Animation as SDK

Locate a robust 3D PDF export for your software with
Animation and have so far not found?

Then we have exactly the right solution for you:
Expand your product by our successful 3D PDF export!

Applications of 3D-PDF Export SDK

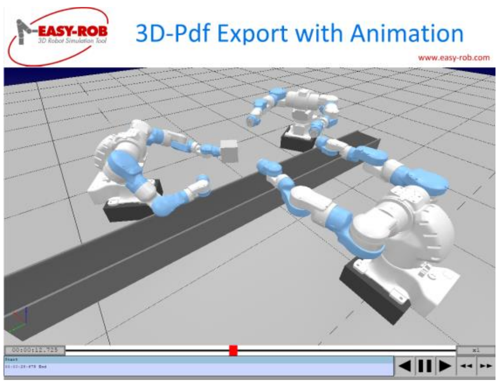
- Quick and easy presentation to third parties
- Disclosure of interactive simulation concepts
- Installation and maintenance instructions
- Documentation of difficult information
- Universal education and training material
- Interactive sales documentation for an improved understanding at customers

The EASY-ROB™ 3D-PDF SDK allows you easily to implement the functions required for the above-mentioned applications in your software.

Save motion sequences with animation in 3D-PDF

In the Adobe® Reader, you can use the navigation bar to start, pause, stop, fast-forward, rewind, and change the speed (x1 / 64x to x64x). The time specifies the real process time.

The layout is freely definable.



Navigation Bar in the Adobe® Reader

Programmierschnittstelle

- A C/C++ and C# method class will be provided for the 3D-PDF SDK:
  ER3DPDF_CAPI

For further requests or information, please contact our sales department under:

sales@easy-rob.com

Self-Collision (Itself Collision)

A powerful collision detection has distinguished EASY-ROB™ for many years. The collision detection allows for a safe simulation and also distinguishes within finely structured assemblies of the CAD geometries.

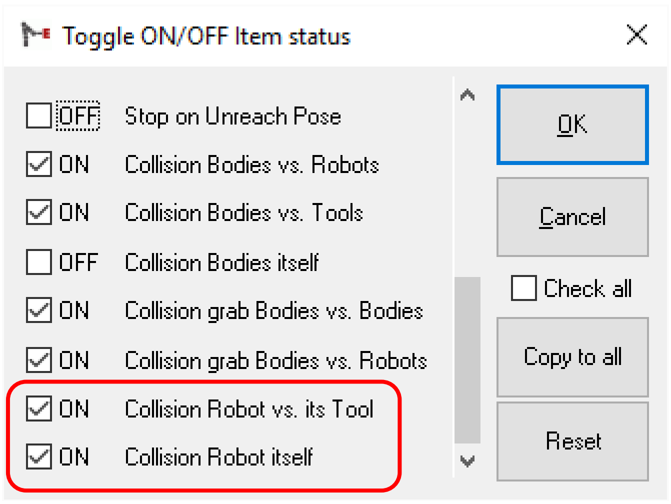
So far, collisions within a robot model have not been detected. If a collision occurred, for example, between axis 5 and axis 2 of a device, this collision could not be detected and displayed. As a rule, this problem was limited by the correct setting of the travel ranges.

Ultimately, the user had to ensure that such "itself collisions" were prevented within the robot kinematics. Here, a lot of care and time was necessary because the entire simulation run had to be checked and manually monitored. The "Stop On" functions, in particular the "Stop on Collision" function, ignored the self-collisions up to now.

Self-Collision global activation

First, the constant readiness in EASY-ROB™ must be created so that the detection of self-collisions takes effect.

To do this, open the "Stop On" menu and set the checkbox "*Collision Robot vs. its Tools*" and "*Collision Robot itself*"


Toggle ON/OFF Item status

- ☐ OFF Stop on Unreach Pose
- ☑ ON Collision Bodies vs. Robots
- ☑ ON Collision Bodies vs. Tools
- ☐ OFF Collision Bodies itself
- ☑ ON Collision grab Bodies vs. Bodies
- ☑ ON Collision grab Bodies vs. Robots
- ☑ ON Collision Robot vs. its Tool
- ☑ ON Collision Robot itself

Stop On Menu

Confirm your selection with OK".

So that the "*Collision Robot Vs. its tools*"- and the "*Collision Robot itself* "-parameter is set permanently for further usage of EASY-ROB™, please save it again briefly in the environment file:

- Menu File → Save → Environment file

If you manually edit the environment file "easy-rob.env", search for the following location:

```
! COLL_ROBOT_TOOL = 1, enables collision between Robot vs. its Tool
COLL_ROBOT_TOOL 1
! COLL_ROBOT_ROBOT = 1, enables collision between Robot itself
COLL_ROBOT_ROBOT 1
```

The variable assumes the value 1 for an active Itself Collision, and 0 for a disabled one.

Creation of Collision Exclude List for Robots

For each device/robot collision exclusion lists can be generated automatically in EASY-ROB™. Collision exclusion lists include all geometries of a device that collide in an "inoffensive" position, for example the home position of the robot.

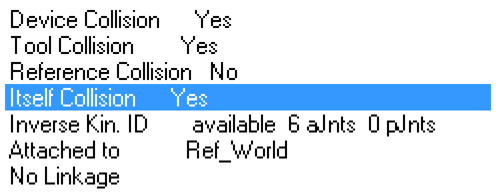
Once these lists have been created and the It Self collision has been activated, EASY-ROB™ will now detect all collisions of the geometries within a robot that are not in the collision exclusion list.

To create the collision exclusion list for the active device, open the Kinematics Window [Ctrl + K] and select:

- Menu Robotics → Itself Collision Settings → Create Collison Exclude List

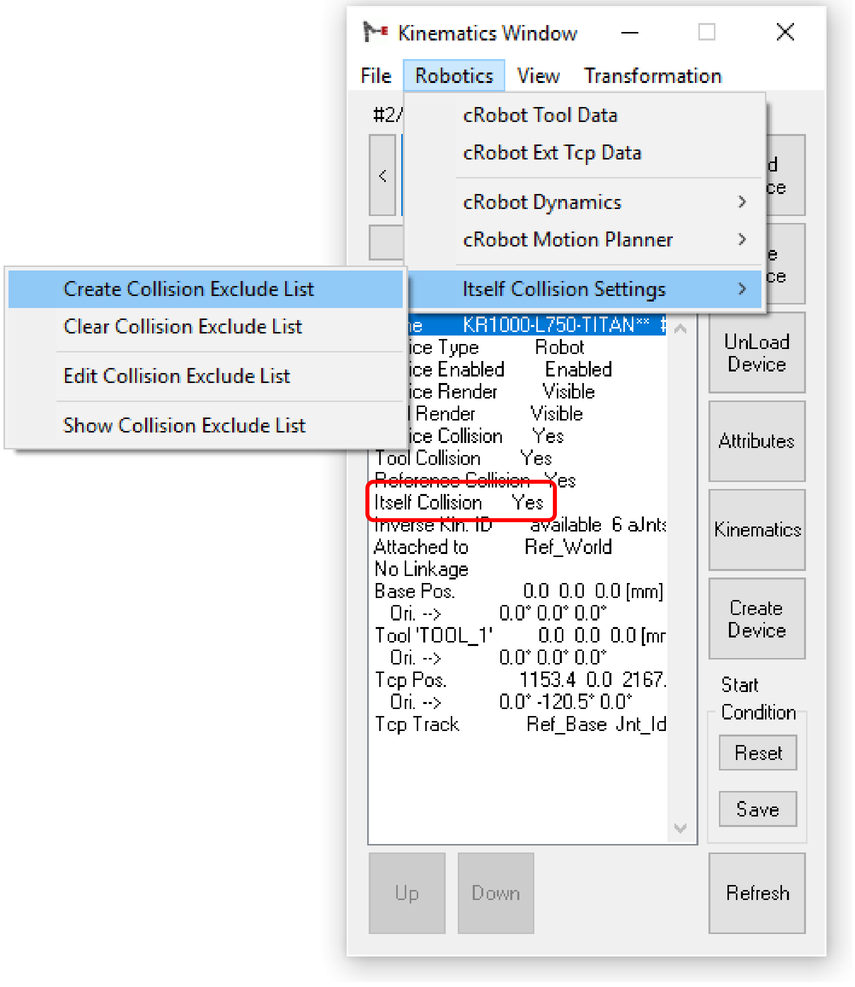
In the next step, the "Itself Collision" in the Kinematics Window must be switched on for the current device

- Double click on "Itself Collision No/Yes"



```
Device Collision     Yes
Tool Collision       Yes
Reference Collision  No
Itself Collision     Yes
Inverse Kin. ID      available  6 aJnts  0 pJnts
Attached to          Ref_World
No Linkage
```

Setting of the Kinematics Window



```
#2/
  e      KR1000-L750-TITAN**
  ice Type          Robot
  ice Enabled       Enabled
  ice Render        Visible
  l Render          Visible
  ice Collision     Yes
Tool Collision      Yes
Reference Collision Yes
Itself Collision    Yes
Inverse Kin. ID     available  6 aJnts
Attached to         Ref_World
No Linkage
Base Pos.           0.0  0.0  0.0 [mm]
  Ori. -->          0.0° 0.0° 0.0°
Tool 'TOOL_1'       0.0  0.0  0.0 [mr
  Ori. -->          0.0° 0.0° 0.0°
Tcp Pos.            1153.4  0.0  2167.
  Ori. -->          0.0° -120.5° 0.0°
Tcp Track           Ref_Base  Jnt_Id
```

Itself Collision settings in the Kinematics Window

Note

The collision exclusion list is created once and stored with the robot model/work cell. If the robot is used again, even in other projects, the list does not have to be recreated.

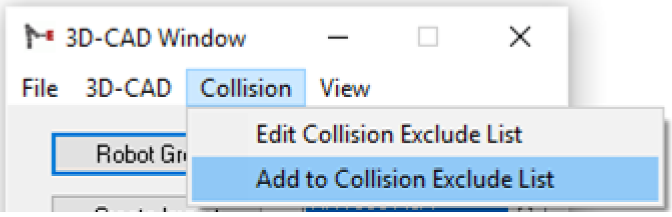
Collision detection works with triangles. These are part of geometries. Even when making adjustments in the Kinematics Window, geometries are always used when considering itself-collisions.

Individual Settings using the 3D-CAD Window

If further geometries should be included in the collision exclusion list, this is done via the 3D-CAD Window.

In the 3D-CAD Window, select geometry pairs or multiple geometries to be added to the collision exclusion list. Then select please:

- Menu Collision → Add to Collision Exclude List.



Individual adding

If the robot model consists of a large number of geometry files, and browsing in the list view is not an option, you can also select the two geometries directly via the graphical user interface.

Click on the "Pick" button and keep [Ctrl] pressed while selecting the geometry files. These are then "highlighted" in the list view. Just repeat the previously explained step "Add to Collision Exclude List".

Of course, the collision exclude lists can also be edited. Please select here:

- 3D-CAD Window → Collision → Edit Collision Exclude List

Note:

- The geometry indices start with 1 to n, with n - number of geometries in the group
- Geometry indices are separated by spaces
- Geometry indexes do not have to occur in a specific order

Improved Reference Collision

A powerful collision detection has distinguished EASY-ROB™ for many years. The collision detection allows for a safe simulation and also distinguishes within finely structured assemblies of the CAD geometries.

So far, collisions have not been detected when for example a tool as a device was directly attached to a robot model (Reference Collision No). Nevertheless, a collision occurred, for example tool with the hand or base axes of the robot, this collision was not detected and also not displayed.

Ultimately, the user had to make sure that such "reference collisions" between tool and robot were prevented.

Similar to the itself collision, an exclusion list is again created, so that geometries that collide anyway after attaching or gripping a device are excluded from the collision test.

Activate „Reference Collision“, exclude geometries

Set the reference collision to "Yes" for the device which is attached or grasped (Child device). To do this, open the "Kinematics Window" and set the value to "Yes" by double-clicking:

- Kinematics Window → Reference Collision → Yes

In the next step you select the geometries of the parent device and the Child device, which collide in the basic position and should therefore be excluded in the collision test.

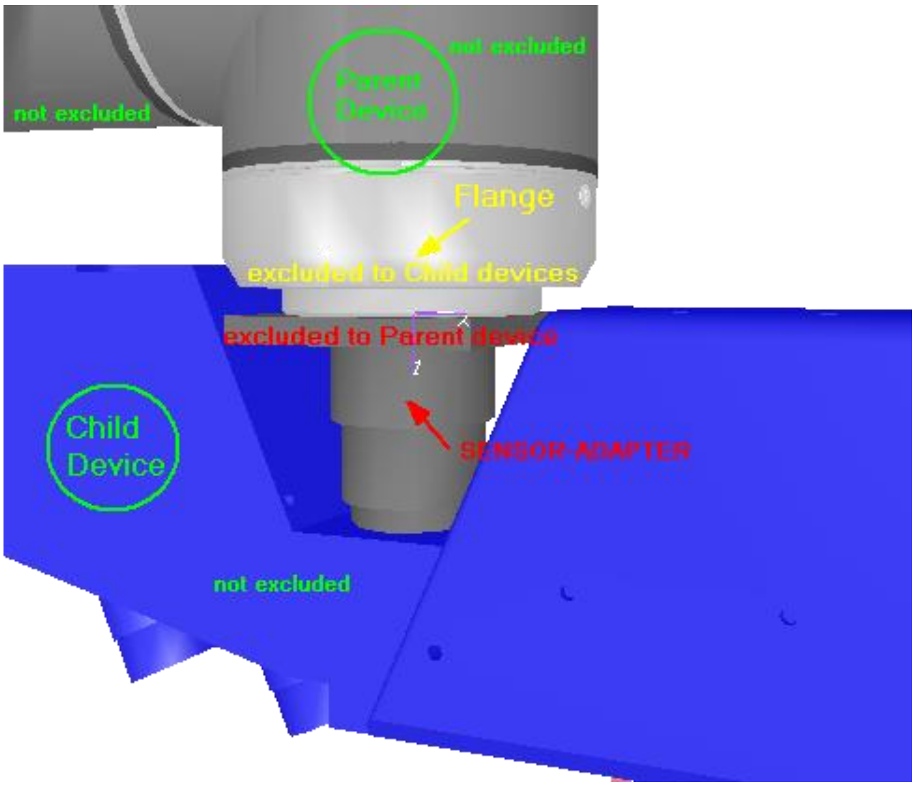
Ultimately, in the case of collision control in general and in particular geometries are monitored, the aforementioned devices actually only use them. Therefore, in the final step, you must set a dependency for the linked or "attachted" devices at the geometry level.

The one becomes the Parent device (parent) and the other becomes the Child device (child) -

Similar names can be found in the CAD: here there are components "parts" which are derived from a basic geometry and this relationship is also called "parent-child" relationship.

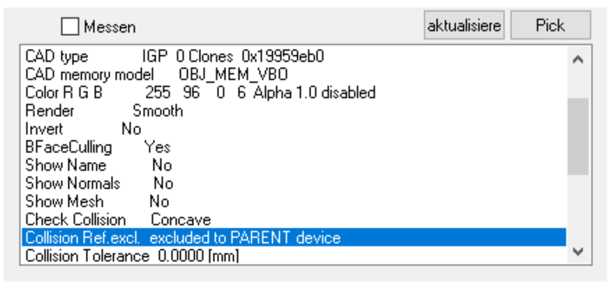
For example, the robot to the Parent device, here the geometry of the axis 6, and the adapter geometry of the sensor, now becomes the Child device, please see next picture:

Parent and Child devices



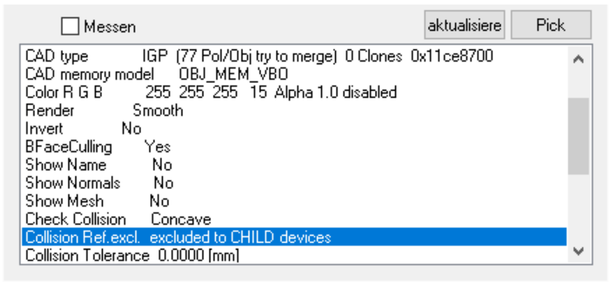
The robot is the Parent device and the sensor is the Child device
In addition to the flange geometry of the Parent device and the adapter geometry of the Child device, all geometries of the Parent and Child device are to be checked against each other for collision.

Example setting in the 3D CAD window for the adapter geometry of the Child device:



```
CAD type            IGP  0 Clones  0x19959eb0
CAD memory model     OBJ_MEM_VBO
Color R G B          255   96    0   6  Alpha 1.0 disabled
Render               Smooth
Invert              No
BFaceCulling         Yes
Show Name            No
Show Normals         No
Show Mesh            No
Check Collision      Concave
Collision Ref.excl.  excluded to PARENT device
Collision Tolerance  0.0000 [mm]
```

Correct settings for the Child device



```
CAD type            IGP  (77 Pol/Obj try to merge)  0 Clones  0x11ce8700
CAD memory model     OBJ_MEM_VBO
Color R G B          255  255  255   15  Alpha 1.0 disabled
Render               Smooth
Invert              No
BFaceCulling         Yes
Show Name            No
Show Normals         No
Show Mesh            No
Check Collision      Concave
Collision Ref.excl.  excluded to CHILD devices
Collision Tolerance  0.0000 [mm]
```

Correct settings for the Parent device

Complete robot libraries

EASY-ROB™ provides complete libraries for the integration of all major types of robots of the market. These include ABB, b + m, Comau, Denso, Eisenmann, Fanuc, Guedel, igm, Kawasaki, KUKA, Mitsubishi, OTC-Daihen, Rice, Stäubli, Tricept, Unimation, Universal Robots and Yaskawa.

The robot libraries of ABB, KUKA, Comau, Fanuc, Stäubli and Yaskawa are almost complete and constantly maintained by us.

Currently more than 1000 robots, positioners and external tracking axes are available of various manufacturers.

Further information:

http://www.easy-rob.com/en/product/extensions/robot-libraries.html

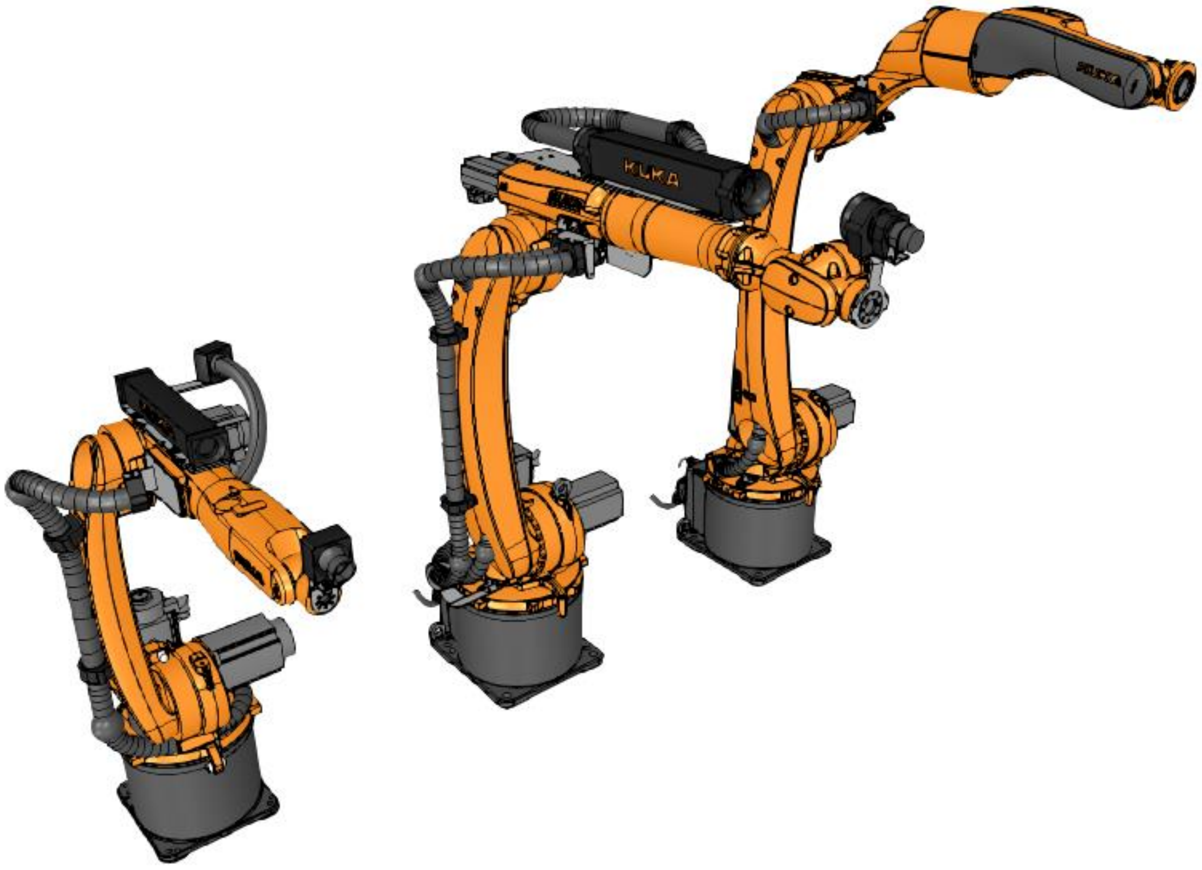
Detail overview of new added Robot Models

As since the last release a lot more robots have been added, you will find a detailed list of the new models at the end of each manufacturer's page.

Important notice:

Non-existent robots, handling systems, machines, tools or even special kinematics can be easily and quickly "reconstructed virtually" in EASY-ROB™

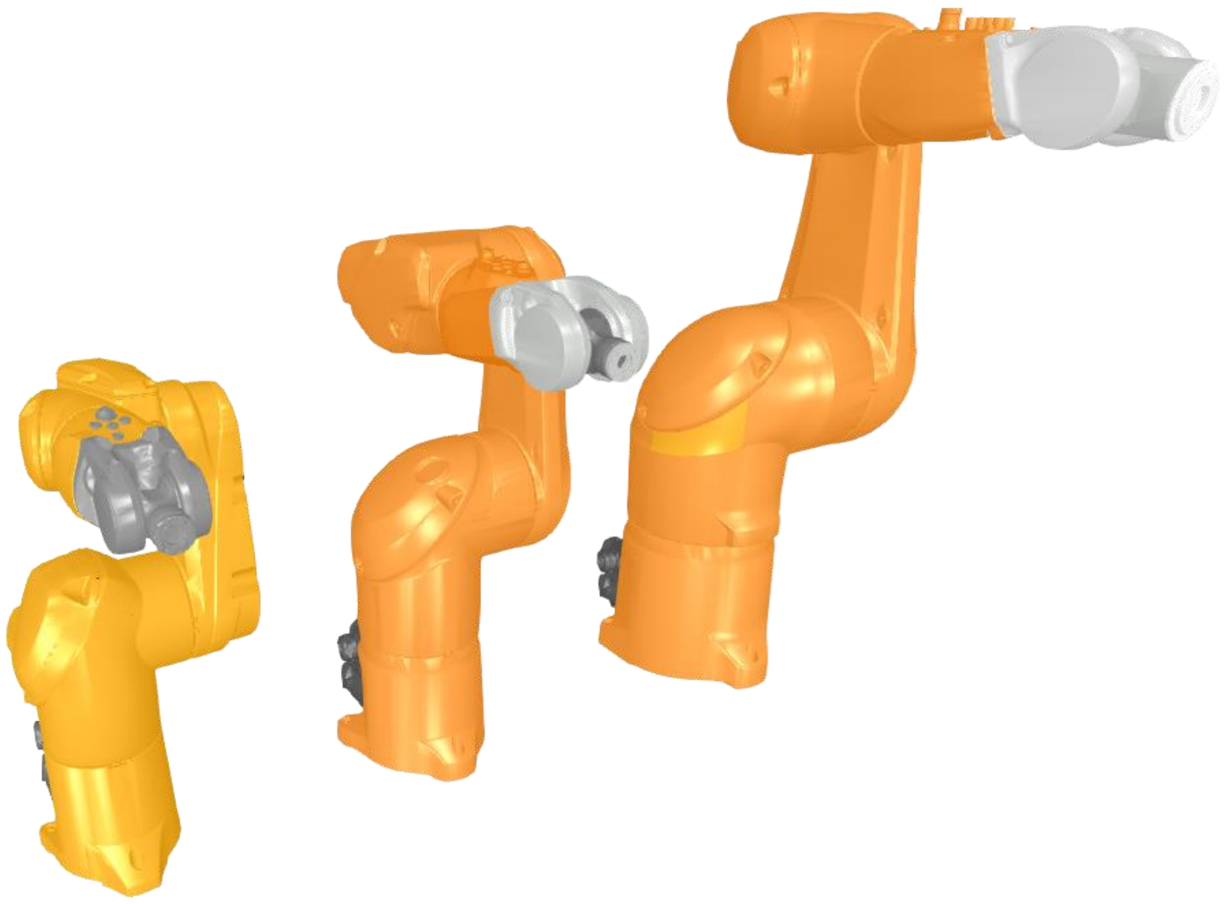
KUKA the new CYBERTECH-Series



KR-10-R1420-HP KR-20-R1810 KR-8-R2100-ARC-HW

KR-10-R1420	KR-30-3-C	KR-8-R1420-arc-HW
KR-10-R1420-HP	KR-30-3-C-F	KR-8-R1620,
KR-12-R1810	KR-3-R540	KR-8-R1620-arc-HW
KR-16-R1610	KR-6-R1820	KR-8-R1620-HP
KR-16-R2010	KR-6-R1820-arc-HW	KR-8-R2010
KR-20-R1810	KR-6-R1820-HP	KR-8-R2100-arc-HW
KR-22-R1610		

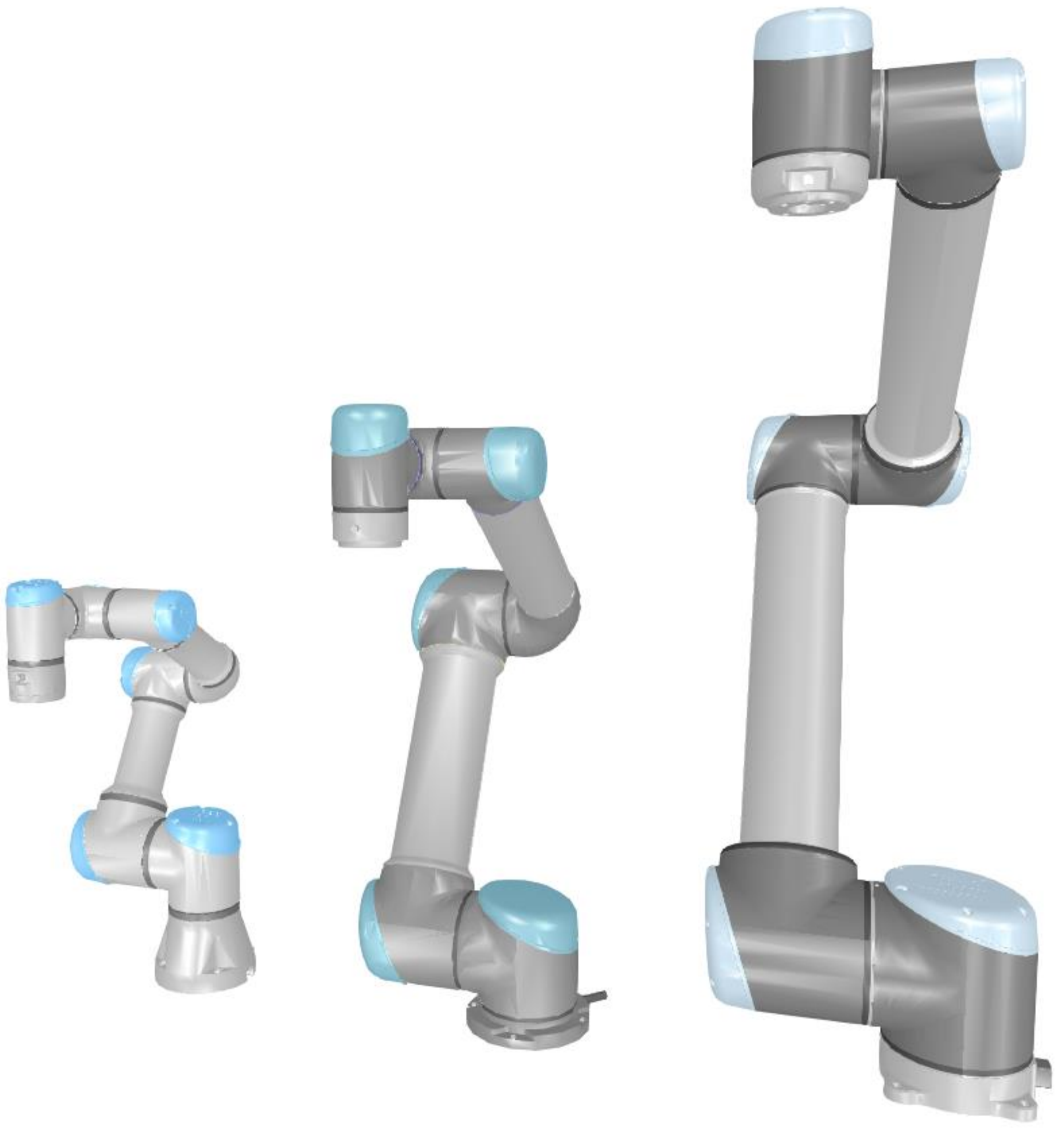
Stäubli the new TX-2 Series with CS9 Controller



TX2-40 TX2-60 TX2-90

TX2-40	TX2-60	TX2-90
	TX2-60-L	TX2-90-L
		TX2-90-XL

Universal Robots the new e Series

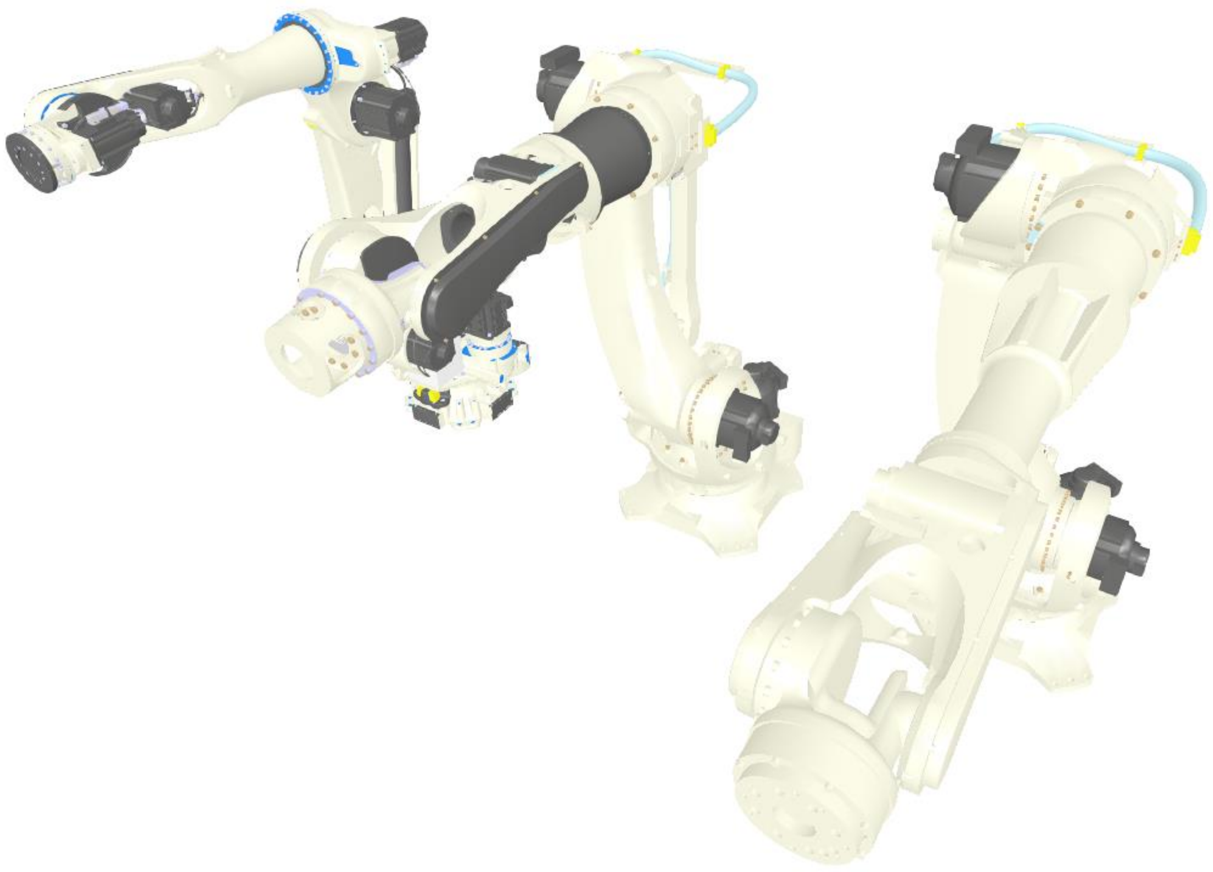


UNIVERSAL ROBOTS

UR3e UR5e UR10e

The existing series have been massively expanded:

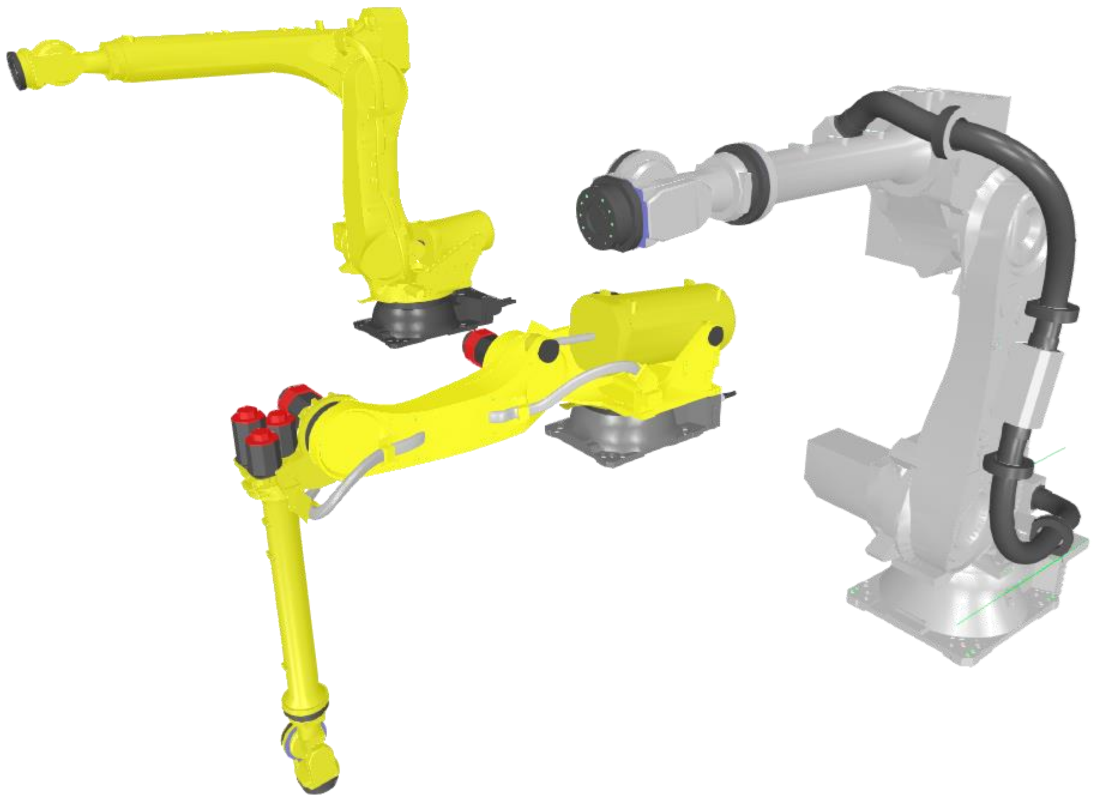
Kawasaki Series CP, CX and BX



CX-110L BX-300-L BX-200X

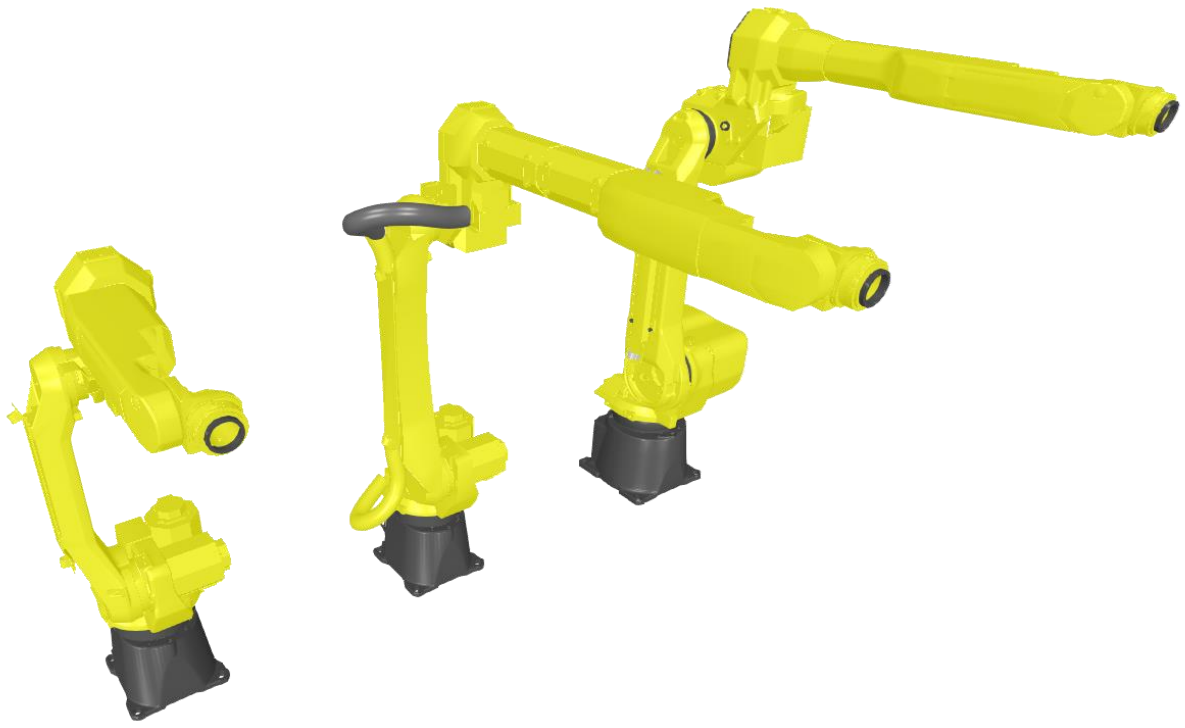
BT-165L	BX-100L-C	CX-110L
BT-200L	BX-100S	CX-165L
CP180L	BX-130X-C	CX-210L
CP300L	BX-165L-C	MC004N
CP500L	BX-165N-C	MC004V
CP700L	BX-200L-C	MS005N
	BX-200X	
	BX-250L	
	BX-300L	

FANUC‘s R-2000 Series



R-2000iC-210L R-2000iC-100P R-2000iC-210WE

FANUC‘s Arc-Mate Serie



Arc-Mate-100-iC-10S Arc-Mate-100-iC-8L Arc-Mate-120-iC-12L

ARC-Mate-100iC-12	M-900IB-280	R-2000iC-100P
ARC-Mate-100iC-12S	M-900IB-280L	R-2000iC-210L
ARC-Mate-100iC-6L	M-900IB-360	R-2000iC-210WE
ARC-Mate-100iC-7L	M-900iB-400L	R-2000iC-220U.rob
ARC-Mate-100iC-8L	M-900IB-700.rob	R-2000IC-270F.rob
ARC-Mate-100iCe-6L		
ARC-Mate-120IC-12L		

Contact data

EASY-ROB Software GmbH

Address:	Hauptstr. 42 65719 Hofheim am Taunus Germany
Contact:	Mr. Stefan Anton, Mr. Patryk Lischka
Phone.:	+49 (0) 6192 921 70-77 / -79
FAX:	+49 (0) 6192 921 70 66
Email:	contact@easy-rob.com sales@easy-rob.com
Web:	www.easy-rob.com
Online Shop:	http://www.easy-rob.com/produkt/shop.html/

EASY-ROB customer area

Content:	Program updates and robot libraries	
Web:	www.easy-rob.com/special/kundenbereich	
Log in details:		
	User name:	customer
	Password:	**********

Own notes