

|  |  |
| --- | --- |
|  | **Update** |
|  |  |
|  | Architecture Proposal |
|  |  |
|  |  |
|  |  |
|  | Implementation Document |
|  |  |
|  | Version: 0.3 |
|  | State: Draft |
|  | Classification: Internal use only |
|  | Author: WES |
|  | Creation date: 2013-07-24 |
|  | Repository: $/Gorba/Main/Motion/Update/Documents/ID\_Update\_Architecture\_Proposal.docx |
|  | Gorba AG  Sandackerstrasse  9245 Oberbüren  Switzerland |

**Table of contents**

[1 Introduction 4](#_Toc362599655)

[1.1 Wording 4](#_Toc362599656)

[2 Architecture Overview 5](#_Toc362599657)

[3 Use cases 6](#_Toc362599658)

[3.1 User creates an update with icenter.update 6](#_Toc362599659)

[3.2 Background System sends Update Command to an Update Client 6](#_Toc362599660)

[3.3 User exports an update with icenter.update to a USB stick 7](#_Toc362599661)

[3.4 User imports update results with icenter.update from a USB stick 7](#_Toc362599662)

[3.5 Update Client receives an Update Command 7](#_Toc362599663)

[3.6 Slave imotion.update connects through Medi to a master 8](#_Toc362599664)

[3.7 Update is received on a Unit and feedback is sent back 8](#_Toc362599665)

[3.8 Update is installed on a Unit and feedback is sent back 8](#_Toc362599666)

[3.9 Update Provider checks for new Update State Infos and log files 10](#_Toc362599667)

[4 Exchanged Objects 11](#_Toc362599668)

[4.1 Update Definition 11](#_Toc362599669)

[4.2 Update Command 11](#_Toc362599670)

[4.3 Update State Info 11](#_Toc362599671)

[5 Remarks and Reminders 12](#_Toc362599672)

[6 Glossary 13](#_Toc362599673)

**Modification management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Modifications** | **State** |
| 0.1 | 2013-07-24 | WES | SW | Initial version | draft |
| 0.2 | 2013-07-25 | WES | SW | Added chapters after meeting with LEF and RAN | draft |
| 0.3 | 2013-07-26 | WES | SW | Updated a few things after final meeting with LEF and RAN | draft |

**Review**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Remarks** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Release**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Dept.** | **Remarks** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Introduction

This document describes the initial architecture proposal for the Gorba Update.

## Wording

Words written with a capital initial letter are specific terms used by Gorba to explain the system. A glossary for these terms is available in chapter 6.

# Architecture Overview



**Unit**(Slave)

**Unit**(Master)

**WAN**

**WAN**



**LAN**

**USB**

**Operator PC**

**Server**

**Protocols**

**WCF (net.tcp)**

**FTP**

**File System**

**Medi**

**Main**

**Software**

**icenter.update**

**icenter**

**BackgroundSystem**

**imotion.update**

**imotion.update**

**Main Comm. SW**

**Copmponents**

Update Service Proxy

Update Service

File / FTP Update Provider

File Update Provider

File / FTP Update Client

Medi Update Provider

Medi Update Client

Create Update 🡪

Upload Resources 🡪

🡨 List Updates (+State)

Update State Info 🡪  
Log files 🡪  
 🡨 Download Resources

Update Command 🡪

🡨 Update State Info

🡨 Log files

Resources 🡪

Update Command 🡪

🡨 Update State Info

🡨 Log files

Resources 🡪

🡨 Update Registration

Update Command 🡪

🡨 Update State Info

🡨 Log files

Resources 🡪

\*

**icenter.update**

**Push**

**polling**

**„polling“**

**Push**

**(after registration)**

\* = to be able to

handle USB updates

# Use cases

This chapter explains very briefly some of the most important use cases of the Update system.

## User creates an update with icenter.update

1. User starts icenter.update (from his desktop or the “ClickOnce” webpage)
2. User logs in with his user name and password
3. User selects the tenant
4. User creates an update
   1. User selects the files to be included in the update (and where to put them on the target system)
   2. User chooses if the update should be immediate or timed
   3. User selects to which units the update will apply
   4. User commits the update
5. icenter.update uploads all required resources to the Background System
   1. icenter.update creates hashes for all included files and stores them locally   
      (persistent over multiple sessions)
   2. icenter.update verifies with the Background System which resources are unknown to it
   3. icenter.update uploads the yet unknown resources to the Background System
6. icenter.update creates an Update Definition and sends it to the Background System
7. Background System stores the Update Definition in the database
8. Background System creates one Update Command for each affected Unit
9. Background System sends all Update Commands to all Update Providers that are registered  
   (the providers can then filter if they are responsible for the given Unit)

## Background System sends Update Command to an Update Client

This use case describes the path of a single Update Command “over the air” from a File or FTP Update Provider to a target repository.

The term “target repository” is used to describe either an FTP or local server directory where the Update Provider puts an update.

1. Update Provider in the Background System receives the given Update Command (see step 9 above in chapter 3.1)
2. Update Provider makes sure that the repository.xml in the target repository is valid; if not it creates the file with the current default repository structure (t.b.d.).
3. Update Provider uploads all necessary Resources to the target repository
   1. Update Provider checks if the Resource already exists in the target repository (check by file name only)
   2. Update Provider uploads the Resource with a temporary file name to the target repository
   3. Update Provider downloads the temporary file and verifies it was not corrupted during upload (optional, what about big files?)
   4. Update Provider renames the temporary file to the right Resource name (<hash>.rx)
4. Update Provider uploads the Update Command to the target repository
   1. Update Provider uploads the Update Command with a temporary file name to the target repository
   2. Update Provider renames the temporary file to the right name

## User exports an update with icenter.update to a USB stick

This use case describes how a USB stick for delivering an update is created in icenter.update.

The term “target repository” is used to describe the directory (structure) on the USB stick where the update will be copied to.

1. User selects “Export updates…” in icenter.update (menu item or button)
2. icenter.update opens the Export Wizard
   1. User selects “USB stick” as target (and the path to the USB stick)
   2. User selects which updates should be exported to the USB stick
   3. User can select which units should be exported (only units touched by the updates previously selected are shown; by default all units are selected)
3. icenter.update gets all relevant Update Commands from the Background System
4. icenter.update checks what relevant resources (i.e. used by any of the downloaded Update Commands) are already available on the USB stick and verifies their hash
5. icenter.update downloads all resources that are currently unavailable on the local system and the USB stick to the local Resource Repository
6. icenter.update uses a File Update Provider to create the repository (see step 2 and following in chapter 3.2)

## User imports update results with icenter.update from a USB stick

This use case describes how log files and Update State Infos are imported from a USB stick into the Background System.

1. User selects “Import update results…” in icenter.update (menu item or button)
2. User selects “USB stick” as source (and the path to the USB stick)
3. icenter.update downloads all log files and Update State Infos from the stick and sends them to the Background System
4. icenter.update deletes all files from the USB stick that have been uploaded successfully to the server

## Update Client receives an Update Command

This use case describes the way an Update Client downloads an Update Command from an FTP server or local directory (e.g. USB stick) – see chapters 3.2 and 0.

The term “source repository” is used to describe the FTP server directory or the local (USB) directory where the Update Client gets an Update Command from.

An Update Client can be responsible for more than one Unit (e.g. master-slave setup)

1. Update Client regularly polls the source repository to:

* check if repository.xml has changed
* check if a new Update Command has arrived for any of the registered Units
* upload archived log files that have not yet been uploaded with this Update Client

1. Update Client downloads a new Update Command from the source repository
2. Update Client verifies if the Update Command was already handled before (check Background System ID and Sequence Number), if so, it doesn’t continue in this sequence
3. Update Client downloads all necessary resources
   1. Update Client checks its Resource Repository if a given Resource is already available
   2. Update Client downloads the Resource with a temporary file name to a temporary location
   3. Update Client verifies it was not corrupted during download
   4. Update Client moves the Resource to the local Resource Repository
4. Update Client notifies local components about the received Update Command  
    Local components can be:

* the local Update Agent that handles updates for this Unit
* another Update Provider that forwards the update to another Unit

## Slave imotion.update connects through Medi to a master

This use case describes the way a slave Unit connects to a master Unit and gets its updates.

1. Medi Update Client (on the slave Unit) sends via Medi an Update Registration to the Medi Update Provider (on the master Unit). The registration contains:

* All Unit names the client is registering for

1. Medi Update Provider answers with an Update Registration Acknowledge
2. Medi Update Provider registers itself to all available local Update Clients (e.g. FTP, USB, …)
3. When a local Update Client on the master gets an Update Command, it notifies the Medi Update Provider (see chapter 3.5 step 5)
4. Medi Update Provider sends all resources for the Update Command (through Medi) to the mentioned Unit (Medi will automatically only transfer the resources that are not available in the slave)
5. Medi Update Provider sends the Update Command to the Medi Update Client on the mentioned Unit
6. Medi Update Client notifies local components about the received Update Command   
   (see chapter 3.5 step 5)

## Update is received on a Unit and feedback is sent back

This use case describes what happens when the Update Agent receives a new Update Command.

1. Update Agent receives an Update Command and all related resources from a local Update Client
2. Update Agent sends an Update State Info [State=Downloaded] through all local Update Clients to the Background System
3. If the Update Command has an Activation Time in the future, the Update Command is stored locally and a timer is set for the Update to happen when configured (this timer is restarted every time imotion.update launches). It doesn’t continue in this sequence.
4. Otherwise the update is installed on the Unit (see chapter 3.8 below)

## Update is installed on a Unit and feedback is sent back

This use case describes the way an update (received with an Update Command) is installed on a Unit.

In this chapter, “Update.exe” refers to the update application (which is also called imotion.update). Update.exe contains the Update Agent.

1. Update Agent checks that the Sequence Number is greater than the last executed Sequence Number (or from a different Background System ID), if not it sends Update State Info [State=Ignored] (if the update was never installed on this Unit) or [State=Installed] (if the update was already installed) through all local Update Clients to the Background System
2. Update Agent verifies that all Resources required for the Update Command are available locally
3. Update Agent sends an Update State Info [State=Installing] through all local Update Clients to the Background System
4. Update Agent computes the difference between the current file system structure and the one required by the Update Command (this set of differences is called “update set” below)
5. If there is a pre-update executable defined, Update Agent runs it
6. If the update set contains an update of Update.exe, it is installed first
   1. A temporary application directory “${root}\Progs\Update\_${unique\_id}” is created   
      ($unique\_id t.b.d.)
   2. A temporary configuration directory “${root}\Config\Update\_${unique\_id}” is created
   3. All files destined for \Progs\Update (as defined in the Update Command) are copied to the temporary application directory
   4. All files destined for \Config\Update (as defined in the Update Command) are copied to the temporary configuration directory
   5. If there was only a change in config for Update.exe, it requests System Manager to relaunch Update.exe; otherwise:
   6. Update.exe is started from the temporary application directory with the parameters “/install ${update\_command\_id} /target ${target\_directory}”
      1. Update.exe waits until the parent process has exited
      2. Update.exe copies itself and all relevant files (config, …) to the given target directory (if necessary it can access the local Resource Repository to get additional files)
      3. Update.exe launches Update.exe in the target directory and exits
      4. Update.exe (in the target directory) deletes the temporary application and configuration directories  
         🡪 after this the procedure starts again at the very first step (1.)
7. Update Agent installs all applications except for System Manager (and Update) as follows:
   1. All relevant Resources are copied from the Resource Repository to temporary directories inside “${root}\Update\Data\Install” representing the same structure as in ${root}
   2. All copied files are verified that they have the same hash as the original Resource
   3. All relevant application are exited through System Manager
   4. All relevant original directories are moved to “${root}\Update\Data\Backup” representing the same structure as in ${root}
   5. All directories from “${root}\Update\Data\Install” are moved to their respective location
   6. The contents of all folders is verified against the current Update Command (names only)
   7. If there is no update for System Manager:

* all applications are launched again through System Manager (otherwise: see below)
* If there is a post-update executable defined, Update Agent runs it
* Update Agent sends an Update State Info [State=Installed] through all local Update Clients to the Background System (this is done even if nothing was updated at all)

1. Update Agent installs System Manager
   1. All relevant Resources are copied from the Resource Repository to temporary directories inside “${root}\Update\Data\Install” representing the same structure as in ${root}
   2. All copied files are verified that they have the same hash as the original Resource
   3. All applications (except Update.exe) are exited through System Manager
   4. System Manager is exited
   5. All relevant original directories are moved to “${root}\Update\Data\Backup” representing the same structure as in ${root}
   6. All directories from “${root}\Update\Data\Install” are moved to their respective location
   7. The contents of all folders is verified against the current Update Command (names only)
   8. If there is a post-update executable defined, Update Agent runs it
   9. System Manager is started with the argument “/waitforexit ${update\_process\_id}”
   10. Update.exe exits 🡪 this will tell System Manager to continue starting up
   11. When Update.exe starts again, Update Agent sends an Update State Info [State=Installed] through all local Update Clients to the Background System

## Update Provider checks for new Update State Infos and log files

Every update provider checks its repository (either by polling or through notifications) for new Update State Infos and log files and forwards it to all registered components (e.g. Update Clients for forwarding or the Background System to update the database).

🡪 There is no workflow available for this use case since it depends on the kind of Update Provider.

# Exchanged Objects

## Update Definition

* Unique ID for the Background System
* Sequence number (by time of creation)
* Activation date/time
* All directories to be updated by this update (always entire directories are updated, never single files)

## Update Command

* Unique ID for the Background System
* Sequence number of the Update Definition
* Activation date/time
* Tenant ID
* Unit name
* Entire directory structure including all files expected on the Unit after the update

## Update State Info

* Unique ID for the Background System
* Sequence number of the Update Definition
* Tenant ID
* Unit name
* Timestamp of the creation of the state info on the Unit, including milliseconds  
  (allows ordering of feedback by time)
* State:
  + Downloaded: update is “parked”
  + Installing: update is being installed
  + Installed: update was successfully installed
  + Ignored: update was ignored because it had a lower sequence number than the last installed update
  + PartiallyInstalled: there were errors while installing the update, some parts of the update were installed
  + Failed: there were unrecoverable errors while installing the update, nothing was installed
* Update Source: human readable text saying where the update was received from
* ErrorReason: human readable explanation for the state (only available if State is “Ignored”, “PartiallyInstalled” or “Failed”)
* Entire directory structure including all “managed” files found on the Unit after the update (only available if State is “Installed”, “Ignored”, “PartiallyInstalled” or “Failed”)
  + For each directory also a flag if the update was successful (required for “PartiallyInstalled”

# Remarks and Reminders

* Update Command always contains the entire structure of all directories managed by imotion.update; there are no partial updates
* If a Unit receives an update (from the same Background System ID) with a Sequence Number less than the last received Sequence Number, the Update Command is ignored
* An Update State Info reports the state of the entire update as well as the state for each directory inside the update (each file with its hash)
* A unit deletes local log and state info files once they have been transmitted by all Update Clients
* TFTStartup.bat will remain the starting point of the Unit (Fallback scenario)

# Glossary

This chapter contains all specific terms that are created in the context of the Update system and explains their meaning.

|  |  |
| --- | --- |
| **Update** | The entire update system, containing icenter.update, Background System and imotion.update |
| **icenter.update** | GUI application running under Windows to create and manage updates for units |
| **Background System** | Server system managing everything in the “Gorba world” using Web Services and a database |
| **imotion.update** | Invisible (console) application running on a Unit that manages the update of software and configuration on that Unit |
| **Unit** | A mobile or stationary system for passenger information running Gorba Software, this can be an iqube, TFT/Topbox or another embedded system |
|  |  |
| **Update Definition** | Object describing the folders and files to be updated by a single update. This object is independent of a Unit. |
| **Update Command** | Object describing an update for a given Unit. It contains the entire folder and file structure expected on the Unit after the update has occurred and is specific to a Unit. |
| **Update Provider** | Class responsible for sending Update Commands to a Unit and receiving Update State Infos as well as log files from a Unit. An Update Provider always communicates with an Update Client. There are different kinds of Update Providers for different kinds of transfer technologies (e.g. File, FTP, Medi, …) |
| **Update Client** | Class responsible for receiving Update Commands for one or more Units and returning Update State Infos as well as log files from the Unit(s). An Update Client always communicates with an Update Provider. There are different kinds of Update Clients for different kinds of transfer technologies (e.g. File, FTP, Medi, …) |
| **Update Agent** | Class responsible for executing an update on the local Unit. |
|  |  |
| **repository.xml** | An XML file describing the repository structure provided by Update Provider and used by Update Client. |
| **Resource** | A file identified with an MD5 hash. The file name is: <hash>.rx |
| **Resource Repository** | A directory where all resources on a Unit are stored. This directory can be shared by multiple applications (resources are unique). |
|  |  |