API PRO CMMS

Client Server Management

By API Maintenance Systems A/S
# TABLE OF CONTENTS

**OVERVIEW** ..................................................................................................................4

**INSTALLATION** ...............................................................................................................5

1. Create CSM/client.ini? ......................................................................................................5
2. Create CSM/client pf? .......................................................................................................5
3. Create/modify CSM/batch.ini ..........................................................................................6
4. Create/modify CSM/batch pf? ...........................................................................................7
5. Create/modify CSM/compile.ini .........................................................................................7
6. Create/modify CSM/compile pf? .........................................................................................7
7. AppServer .......................................................................................................................7
8. Make CSM-initiator packages ...........................................................................................8

**CSM ADMINISTRATION** .................................................................................................9

Simple ..................................................................................................................................9
Advanced ..............................................................................................................................10
Initiator .................................................................................................................................11
Other options .......................................................................................................................15

**BEHIND THE SCENE** ....................................................................................................18

Start API PRO at a client: ....................................................................................................18
Command line interface .......................................................................................................21

**CSM WORK-FLOW** ..........................................................................................................25

Load .....................................................................................................................................25
Compile .................................................................................................................................25
Deploy ..................................................................................................................................27

**HOW TO** ........................................................................................................................28

Add/change some files ..........................................................................................................28
Add a program .......................................................................................................................29
Delete some files ..................................................................................................................29
Reject login ...........................................................................................................................29
CSM event .............................................................................................................................30
Overwrite file-type ...............................................................................................................31
Advanced MSI-library ..........................................................................................................31
Add an “interface”-database for compile ............................................................................31
Check AppServer continuously for changes ........................................................................32

**FILE-SYSTEMS** ...............................................................................................................34

Client ....................................................................................................................................34
Appserver ..............................................................................................................................35

**DATA BASE** ....................................................................................................................41

Build-lock .............................................................................................................................41
Change-lock ...........................................................................................................................41

**COMMAND-LINE MSI-INSTALLATION** ..........................................................................42

**ENHANCEMENTS ARE RELATED FIX-PACKAGES** ......................................................44

**“ISSUES”** ........................................................................................................................46
Overview

CSM or Client Server Management handles common tasks for keeping the API PRO application updated – including:

- Install product updates like fix-packages, custom-packages or new service packs.
- Produce packages for clients for easy set-up using MSI-packages.
- Avoid the use of a file-share between the AppServer and Clients.
- Ensure the local image of the application is up-to-date.

To do this CSM needs to be configured at the AppServer (running 7.0SP05 or later).
Installation

In order for CSM to work we need to have access to the file-system of the AppServer and set-up the following files here:

1. Create CSM/client.ini?

    WORK/apipro.ini will be used as a default for CSM/client.ini, but if some changes should be made for the CSM-clients only, this file should be used.
    Considered the use of a DNS-entry for the AppServer as this could make later changes to the network set-up much easier to deploy.

    ...
    [Startup]
    ;DLC=
    nosplash=yes
    ...

    [API PRO]
    AppServer-Name=<name>
    AppServer-Host=<dns> or <ip>
    AppServer-Port=<port>
    ...
    Section Startup: Remove DLC, set nosplash.
    Section API PRO: AppServer settings

2. Create CSM/client.pf?

    WORK/client.pf will be used as a default for CSM/client.pf, but if some changes should be made for the CSM-clients only this file should be used.

    ...
    -mc                # Compression
    -p csm/client.p    # Startup Program
    ...

3. Create/modify CSM/batch.ini

Batch processing uses CSM/batch.ini:

...  
[Startup]  
DLC=<dlc>  
...  
Section *Startup*: set-up DLC.
4. Create/modify CSM/batch.pf

Batch processing uses `csm/batch.pf`:

```
-db C:\APISERVER\APITEST8\DB\api3
...
-b # Batch mode
-p csm/server.p # Startup Program
```

TIP: Use a direct database connection if possible as this will allow the AppServer to be offline while deploying business critical changes to the system.

5. Create/modify CSM/compile.ini

Batch compile (only) processing uses the set-up from `CSM/compile.ini`:

```
...
[Startup]
DLC=<dlc>
PROPATH=.,.,.,\CUSTOM,.,\XCODE
...
```

If this file is missing `CSM/batch.ini` will be used instead.

6. Create/modify CSM/compile.pf

Batch compile (only) processing uses the set-up from `CSM/compile.pf`:

```
-db C:\APISERVER\APITEST8\DB\api3
...
-b # Batch mode
-rx # Enable compile of xcode
-p csm/compile.p # Startup Program
```

NOTE: If this file is missing `CSM/batch.pf` will be used instead REQUIRING a full-progress license for the compile to work. Using the progress option –rx and the special compile program `csm/compile.p` allows a run-time license to do the compile.

7. AppServer

Files monitored by CSM need to be documented. This is done by making a “Recompile & deploy” build using the “Client Server Management” tool.
All programs will be compiled and will together with the other files (help-files, crystal report setups) be recorded so future changes to the AppServer WORK/-area will be found and optional forwarded to the clients at next login.

8. Make CSM-initiator packages

Run the Client Server Management program and produce a zip or msi package for the clients. Alternative the files required can be exported to init/:

```
cd csm
init.cmd
```

This directory can be copied, ZIP’ed or MSI’ed to another location where the WIX-toolset (as mentioned later) is installed:

```
candle.exe client.wxs -ext WixUIExtension -ext WixUtilExtension
light.exe client.wixobj -ext WixUIExtension -ext WixUtilExtension -spdb -out client.msi -sw1076
```
CSM administration

The program “Client Server Management” found in the “System” menu provides the functionality for administration divided into 3 tabs.

Simple

At start-up the status for the latest build are shown. The entire frame is a drop-zone accepting archives, libraries, 4gl-files and root-directories.

If the build is being processed by the server the dialog will be auto-updated every 2 seconds showing the status in the 3 boxes which match the 3 major steps in the CSM-workflow.

Load
- **green**: Box showing the status of the load process (1 of 3 steps):
  - All files have been loaded at the build-area at the server.
- **yellow**: Files are being loaded.
- **red**: The package couldn’t be loaded.

Compile
- **green**: Box showing the status of the compile process (2 of 3 steps):
  - All programs have been compiled in the build-area at the server.
The buttons in the bottom of the dialog provides 3 ways to start the work-flow:

**Load & deploy**
- The standard “open-file-dialog” appears with relevant filters.
- Besides zip-archives, progress-libraries can be used.
- The archive is unpacked to a temporary directory at the client and then “Load“ begins. The same will happen if the archive is dropped in the “drop-zone”.

**Recompile & deploy**
- Create a new empty build and mark it for a total compile.
- This function can be used when changes are made to the translations set-up.
- Used at install-time to make the level “1” – documentation.

**Import & deploy**
- The standard “open-directory-dialog”.
- Only roots of package are accepted, and at least one of the well-known sub-directories: CUSTOM/, XCODE/, WORK/ or special-files listed later needs to be present otherwise the load will be rejected. The root can be dropped in the “drop-zone” to trigger the same functionality.

To start the “Load”-phase the attached description is shown in a “Load files?”-dialog.
Here the user has to accept or reject the process to begin.

If the package is accepted the “Load” will start by creating a new build at the AppServer.
Then the files are uploaded from the client to the server without the need for a file-share.
The different ways of loading packages on this first Simple-tab, will all automatically trigger the 2 next steps Compile and Deploy of the build.

**Advanced**

The browser shows all the known builds and provides a broader set of functions to control the CSM-workflow.
Unlike the “deploy” buttons found at the Simple-tab, the next steps of the CSM-work-flow are not triggered automatically.

**Load package**  The standard “open-file-dialog” appears with relevant filters. Besides zip-archives, progress-libraries can be used. The archive is unpacked to a temporary directory at the client and then *Load* begins.

**Recompile**  Create a new empty build and mark it for a total compile. This function can be used when changes are made to the translations set-up. Used at install-time to make the level “1” – documentation.

**Import directory**  Import a special directory to a new build.

**Description**  Show the description of the build in focus.

**Compile build**  2. step of the work-flow: *Compile* the build in focus. A new batch-process is started at the AppServer. Locked build can’t be compiled. The process is documented in detail in the section: CSM work-flow

A detailed walk-through of the 3 steps of the work-flow – 1: Load, 2: Compile, 3: Deploy.

**Load**

First step of the CSM work-flow is used to set-up a build which is a work-space for changes that should be applied to the application. Builds are located at the AppServer under the CSM/build/-directory, and numbered by a counter.

Changes can be fix-packages, complete custom-packages or customized changes made locally and different functions can be used to load these from any client. Zip-archives or progress-libraries will be unpacked at the client (to a temporary directory). Using the normal connection to the AppServer, the files are uploaded to the build and required sub-directories are created on the way.

**Compile**

3. step of the work-flow: *Deploy* the build in focus. The process is documented in detail in the section: Deploy
**Check**

Do a manual check of the work/ and make a change-log if any changes are found, including deletion of files. This function is part of the *Deploy*-phase.

**Initiator**

This tab provides the capability to configure and make MSI-packages which ease the installation at the clients.

NOTE: “WiX Toolset” found at: [http://wixtoolset.org/](http://wixtoolset.org/) is required for making MSI-packages and should be installed at the client running this set-up.

Initial values are taken from the current environment.

Configuration parameters:

- **System/License**
  - Active license.
- **Title**
  - Used for the desktop shortcut, start-up menu, and control panel entry.
- **AppServer URL**
  - Connection parameter for the AppServer given as an URL.
- **AppServer Host**
  - Name of host, running the “Name-server” which provides the connection to the AppServer.
- **AppServer Port**
  - Port on host.
- **AppServer Name**
  - Name of AppServer.
  - NOTE: This value is case-sensitive.
- **INSTALLDIR**
  - Default installation directory at the clients.
- **Update ini-file**
  - The AppServer values are written to the ini-file.
  - A copy of CSM/client.ini (or WORK/apipro.ini if missing) will be installed.
Language
Which language should be the default when API PRO starts.

Update pf-file
The language value is written to the pf-file.
A copy of CSM/client.pf (or WORK/apipro.pf if missing) will be installed.

Shared installation
Normally the generated package will be installed for the current user only and NOT require local administrator rights. Checking this will change the package to be a shared installation for all users using the same files at each client (including ini and pf files) and will like the “baseline” require local administrator rights to be installed.

Load set-up
Load a set-up previous saved by the Initiator menu option.

Generate
Choose a filename for the MSI-package.
Two utilities: candle.exe and light.exe of the “WiX Toolset” have to be located first the client is used for making packages.
The set-up will be saved if the current .ini-file under the “API PRO”-section:

[API PRO]
AppServer-Host=csm.apipro.com
AppServer-Port=15000
AppServer-Name=APITEST8
Candle-command=C:\Program Files (x86)\WiX Toolset v3.7\bin\candle.exe
Light-command=C:\Program Files (x86)\WiX Toolset v3.7\bin\light.exe

NOTE: If this set-up is wrong – just delete the two lines from the .ini-file and point to the right locations next time the function is used.

The files of the package are:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csm/apilogo.bmp</td>
<td>Image in top of start-up window. Should this file be missing work/image/apilog5.bmp which is used by the API PRO login, will be re-used. Sample:</td>
</tr>
<tr>
<td>csm/apipro.ico</td>
<td>Icon for start-up window. Should this file be missing the standard work/image/apipro.ico will be used.</td>
</tr>
<tr>
<td>csm/banner.bmp</td>
<td>Optional image.</td>
</tr>
<tr>
<td>csm/client.cmd</td>
<td>Client start-up script. Should this file be missing the standard work/client.cmd will be used.</td>
</tr>
<tr>
<td>csm/client.ico</td>
<td>Icon for desktop-shortcut and menu-item in the start-up folder. Should this file be missing the standard work/image/apipro.ico</td>
</tr>
</tbody>
</table>
API PRO Client Server Management (v8) or work/image/api7.ico (v7) will be used.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
<th>Default File</th>
</tr>
</thead>
<tbody>
<tr>
<td>csm/client.ini</td>
<td>Client configuration part 1 including AppServer connection parameters. Should this file be missing work/apipro.ini will be used.</td>
<td>csm/client.ini</td>
</tr>
<tr>
<td>csm/client.pf</td>
<td>Client configuration part 2 including language set-up. Should this file be missing the standard work/client.pf will be used.</td>
<td>csm/client.pf</td>
</tr>
<tr>
<td>csm/dialog.bmp</td>
<td>Optional image. Size should be: 493x312 pixels. Sample: <img src="sample.png" alt="Sample Image" /></td>
<td><img src="sample.png" alt="Sample Image" /></td>
</tr>
<tr>
<td>work/csm/client.r</td>
<td>Start-up program which ensures the local application is up-to-date before API PRO starts.</td>
<td>work/csm/client.r</td>
</tr>
</tbody>
</table>
Other options

The menu provides some additional options:

Load & deploy The same option as the button found on the Simple-tab – see: Simple.
Recompile & deploy The same option as the button found on the Simple-tab – see: Simple.
Import & deploy The same option as the button found on the Simple-tab – see: Simple.

Load package The same option as the button found on the Advanced-tab – see: Advanced.
Recompile The same option as the button found on the Advanced-tab – see: Advanced.
Import directory The same option as the button found on the Advanced-tab – see: Advanced.

Description The same option as the button found on the Advanced-tab – see: Advanced.

Compile build The same option as the button found on the Advanced-tab – see: Advanced.
Program list Show the list of programs to be included in the compile.
Compile log Show the log from the compile.
Missing List programs which did not compile.

Deploy build The same option as the button found on the Advanced-tab – see: Advanced.
Lock build Locks the build for changes and (re?)-compile.
Unlock build Unlocks the build.
Delete build Delete the build and all related files.

Check The same option as the button found on the Advanced-tab – see: Advanced.
Changes Show the list of changes (from the previous change-log).
Delete changes Delete the change-log.

Lock Locks the CSM for changes. New fix-packages can’t be loaded.
Unlock Unlocks the CSM allowing new fix-packages to be loaded.

Export All changes made to the local work/-directory are exported to a directory. All created or modified files are copied and required subdirectories are made to keep the structure intact. A package.txt-file listing the changes is placed in the root of the directory. Progress
source files (.p or .w suffixes) will be placed under a client/-subdirectory and other files under a work/-subdirectory. This function can be used to make a backup if the client-image is to be updated or to upload the changes to the appserver, which then will be deployed every client.

**Reset client**  
Removes the csm/client.log file at the client, resulting in a new complete client-image load at next application start-up.

**Reset server**  
Data-areas at the server are deleted, except for the change-log-value, since we need this sequence to increment continuously.

**Initiator**  
Export files for the client-initiator and some additional set-up to a directory. The directory can be zipped manually or be edited by the Set-up-dialog to create a custom MSI-package.

**init.zip**  
Make a client-initiator package and pack this as a zip-archive. As for the unzip-command, a file-dialog will appear if the zip-command environment-key is missing or doesn’t point to the command and the new value will be stored. Finally a save-file-dialog will ask for the filename of the client-initiator-package.

**init.msi**  
Make a client-initiator package and pack this as a zip-archive. As for the unzip-command, a file-dialog will appear if the zip-command environment-key is missing or doesn’t point to the command and the new value will be stored. Finally a save-file-dialog will ask for the filename of the client-initiator-package.
Behind the scene

Start API PRO at a client:

1. A desktop-icon, and a special folder in the start-up menu provides different access points for the application.
   The `csm/client.r` program is the first program called (set-up in `client.pf`).

2. The directory where the `csm/client.r` resides is considered to be the `work/`-directory on this client.

3. A connection to the app-server is established using the key-values of the “API PRO”-section of the `client.ini` file – just like will do in `apipro.w` later.

4. The `csm/server.r` program is loaded at the AppServer.

5. As for the client side, the server will consider the directory where `csm/server.r` resides to be the “work/”-directory on this server.

6. The client looks for the current state (change-log) in the local `csm/client.log`.

7. The server is asked for complete list of changes, given the installed `change-log` at the client.
   If the server is at the same level (will be the typical case!) the client is considered to be up-to-date and the application will start immediately.

If the server is “newer”, all the “list of changes” in between are merged into a resulting list of files. Should any of the required lists be missing a complete client-
image will be given.
If the server is “older” than the client, the value is rejected by the server and a fresh complete client-image will be returned. This could happen if a system has been restored from a backup.

For each file the list include size and modification time. R-codes will list MD5-values.
The changes can include deletion of files too.

8 The client will skip all r-codes with unchanged MD5-values and calculate the total number/size of files of the update.

9 Files are downloaded - one file at the time, showing the status on the way and creating any required sub-directories.

In version 7 this dialog had two progress-bars:

If an error occurs the bar change color:
A typical error is a CodeJock lock when two clients wants to access the same file:

** Cannot find or open file .../work/icons/cj1521/Vista.cjstyles, errno = 13. (43)

10 The new value for change-log is stored in csm/client.log reducing the next installation to only look for new changes.

11 The file csm/client.err lists the errors or will be deleted if everything went ok.

12 The file csm/client.ok will be deleted if any error occurred, or will be created if the download went ok.

13 The standard apipro.w program is called.

From this login dialog two hidden CSM-events are available through keyboard shortcuts:
ALT+F5 makes a client-refresh and download new files since last refresh.
Use-full when a build have been deploy after the client was started.
ALT+F12 asks the server for a complete image of the required files.
Use-full if some files have been lost at the client.
Command line interface

Most of the functionality in CSM can be accessed from command-line through the \texttt{--PARAM} option.

At the clients the start-up program \texttt{csm/client.p} will accept the following values for \texttt{--PARAM}:

<table>
<thead>
<tr>
<th>-PARAM syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(blank)</td>
<td>Normal client start-up with optional file download and login.</td>
</tr>
<tr>
<td>CheckClient</td>
<td>Used for updating the client installation only. Any new changes will be downloaded but the application will not be launched.</td>
</tr>
<tr>
<td>CompileAll</td>
<td>If the CSM is locked this command is rejected:</td>
</tr>
</tbody>
</table>

A new build will be created and the number will be reported:

![Information Window](image)

Then the compile starts at the app-server.

NOTE: The build will not be deployed to the app-server when the compilation has ended. Use the separate \texttt{DeployBuild}\texttt{-command}.

<table>
<thead>
<tr>
<th>CompileBuild</th>
<th>Compile build \textit{number} at the app-server. Beneath this process is described in detail.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateBuild</td>
<td>Get next build number from \texttt{csm/build.log} and create the \texttt{csm/build/number/\textit{root}}. A database-lock held during this check ensures only one process have access to the files.</td>
</tr>
<tr>
<td>CreateInit, \textit{dir}</td>
<td>The files of the CSM-initiator will be exported to this directory.</td>
</tr>
</tbody>
</table>
DeleteBuild, number

If the build is locked it can’t be deleted:

Unlocked builds will have the \texttt{csm/build/number/} root will be removed and the build can no longer be used.
This can’t be undone.

DeployBuild, number

The build has to be next in line or it will be rejected:

The build will be locked and deployed at the app-server triggering a \texttt{CheckWork}-command

DirToBuild, directory, number

If the build is locked it will be rejected:

The structure of \texttt{directory} will be copied to the build creating any required sub-directories.

EmptyBuild, number

If the build is locked it can’t be emptied:
Unlocked builds will have the `csm/build/number/` root left empty (all sub-directories and files are removed).
This can’t be undone.

**LockBuild, number**  
Locks the builds for changes.

**ResetClient**  
A full client-image will be downloaded.
This command is available from the standard login-dialog using the keyboard-combination ALT-F12.

**RollBackLast**  
The latest build deployed at the server will be rolled back. A new build is created, and loaded with the source from the backup-area.
All programs are then compiled and finally the new build is deployed to the app-server.
At this state a new RollBackLast command will go through the same steps and return the system to the state from before the first RollBackLast.

**StartClient**  
Normal start-up.
Any new changes will be downloaded (like CheckClient) and then the application will be launched.
This command is available from the standard login-dialog using the keyboard-combination ALT-F5 (F5 known for “refresh” in browsers...).

**UnlockBuild, number**  
Unlock a locked build.

Sample script:
```
Cd \APIPRO8\APITEST8 & Client.cmd -param RollBackLast
```

At the server `csm/server.p` will accept the following values for `-PARAM`:

<table>
<thead>
<tr>
<th>-PARAM syntax</th>
<th>Description</th>
</tr>
</thead>
</table>
| CheckWork     | Run through work/ and check the status of each file against the last known status stored in `csm/server.lst`. If any change is found a new change-log will be created and each change will be recorded in the corresponding log-file `csm/change/number.lst`. The new status is stored in `csm/server.lst`.
A database-lock held during this check ensures only one process have access to the files.
Consider using this command triggered from schtasks to ensure
Compile build number. Beneath this process is described in detail.

CreateInit, directory

The files of the CSM-initiator will be exported to this directory

Deploy build number.

As for the command at the client, the build has to be next in line and will be locked.

Sample script (cd in work/):

prowin32.exe -pf ../csm/batch.pf -ininame ../csm/batch.ini -param CheckWork
CSM work-flow

A detailed walk-through of the 3 steps of the work-flow – 1: Load, 2: Compile, 3: Deploy.

Load

First step of the CSM work-flow is used to set-up a build which is a work-space for changes that should be applied to the application. Builds are located at the AppServer under the CSM/build/-directory, and numbered by a counter.

Changes can be fix-packages, complete custom-packages or customized changes made locally and different functions can be used to load these from any client.

Zip-archives or progress-libraries will be unpacked at the client (to a temporary directory).

Using the normal connection to the AppServer, the files are uploaded to the build and required sub-directories are created on the way.

Compile

Second step of the CSM work-flow is to compile the build.

This is done into the separate CSM/build/number/work/ directory, leaving the present application running untouched.

PROPATH of the compile-process needs to include the current server version, and other outstanding builds in correct sequence.

If a new build 13 is to be compiled, and build 12 is not yet deployed it would give a complete compile-path:

1) CSM/build/13/client/
2) CSM/build/12/client/
3) CLIENT/
4) CSM/build/13/custom/
5) CSM/build/12/custom/
6) CUSTOM/
7) CSM/build/13/xcode/
8) CSM/build/12/xcode/
9) XCODE/
10) WORK/

Only directories which actually exists are added to the list and work/ is included to provide other files like images and icons.

The current MD5 value is likewise found looking in:
1) CSM/build/12/work/
2) WORK/

and compared to the new value of the compiled r-code in: CSM/build/13/work/
If the values are equal the r-code are deleted from build 13.
A re-compile of build 12 shouldn’t include 13 in the propath.

Steps in the compile-phase:

1  Ensure CSM/build/number/work/ holds no r-code (leftovers from a previous compile)
   Status files are deleted prior to the compile:
   CSM/build/number/compile.err
   CSM/build/number/compile.log
   CSM/build/number/compile.ok
   CSM/build/number/missing.lst

2  Spawn a batch-process using the set-up from CSM/compile.ini and CSM/compile.pf

3  Trigger the workflow event: compilebegin

4  Configure the compile: PROPATH, LANGUAGES, translation-db…

5  For every deleted source-file (see CSM/build/number/delete.lst) an “illegal placeholder” is created.
   The PROPATH is then searched for ancestors and if found they will be copied to make sure the compile will have the complete source.
   Examples:
   XCODE/file.i is seen as an ancestor for CUSTOM/file.i as CUSTOM/ is placed before XCODE/ in the PROPATH.

6  For every program:
   Find current MD5-value.
   Make required “save-into”-directory

7  Compile the program and catch errors

8  Compare new MD5-value and delete if unchanged.

9  When all programs are compiled, all empty directories are deleted.

10 Make a CSM/build/number/compile.ok or CSM/build/number/compile.err file to signal the result.

11 Trigger the workflow event: compileend

12 If CSM/build/number/deploy.do is found and no errors occurred, then the build will be deployed immediately to the AppServer. This feature should be used with care during normal office hours as the clients will then run with potential different versions of the business-logic at the same time.
   The file CSM/build/number/deploy.tot will trigger deployment even if an error occurred during compile.
   If the compile is made by an restricted progress client (option –rx) the deployment process is spawned to a new batch process with the set-up from CSM/batch.ini
and CSM/batch pf files allowing access to the database.

**Deploy**

When a build is being deployed, files from `build/number` are copied to the active directories of the AppServer. This process can be triggered automatically when the build have been compiled, done manually using the CSM-tool or launched by the CSM-command: “DeployBuild” from a command-line.

It’s not possible to deploy a build if the previous one hasn’t been deployed to the AppServer and builds can’t be re-deployed.

Steps of the deploy-phase:

1. Trigger the workflow event: ‘deploybegin’
2. Lock the build if not already locked.
3. Status files are deleted:
   - `build/number/deploy.ok`
   - `build/number/deploy.err`
4. The associated backup-area: `backup/number` are cleared.
5. Files from: `build/number` are copied to their destination except for special files of the build and files listed in `build/number/delete.lst`
   Before the destination is over-written or deleted the previous contents is backed-up by a copy to the backup-area.
6. Actions listed in: `build/number/deploy.act` are executed
7. The updated file-system is documented by a CSM-“check”.
8. The special file: `build/number/fix-db.do` will file spawn a new batch process with the set-up from CSM/batch.ini and CSM/batch pf files allowing access to the database and all executed programs being updated.
   Here `fix-db/fix-db.p` will load the designs and other relevant data.
9. Trigger the workflow event: ‘deployend’
How to

This section lists some common scenarios working with CSM.

Add/change some files

Some files placed locally should be part of the application.
Example: A new CR-template called fiscal.rpt should be loaded into WORK/template/.

1. Make a directory that will be the root of the package.
   C:\temp\load

2. Create the required subdirectories to hold the file-structure of the changes:
   C:\temp\load\work
   C:\temp\load\work\template

3. Place the changed files:
   C:\temp\load\work\template\fiscal.rpt

4. Make a description of the changes in package.txt:
   Edit C:\temp\load\package.txt

5. Setup compilation. By default all known programs will be compiled but the file compile.lst can be used to specify a smaller list (could be empty).
   Make C:\temp\load\compile.lst
   Note: Make sure the suffix of the file is right and not ends up as “.lst.txt”

6. Load the root-directory into CSM and go through the 3 steps (load, compile, deploy).

7. Quit API PRO and start a new client.

8. Validate the changes were mirrored to the local image.
   Now work\template\fiscal.rpt should be present.

9. Delete the local root-directory.
   C:\temp\load

Local file-system required for this change:

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\temp\load\</td>
<td>Root-directory</td>
</tr>
<tr>
<td>C:\temp\load\work\crprint\template\fiscal.rpt</td>
<td>File to be loaded</td>
</tr>
<tr>
<td>C\temp\load\compile.lst</td>
<td>Empty file</td>
</tr>
<tr>
<td>C\temp\load\package.txt</td>
<td>Description of the package</td>
</tr>
</tbody>
</table>
Add a program
Load a locally made program and make it part of the application on all clients.
Example: A new program hello.p

1. Load the program using “Load & deploy” or the drop-zone.
   The source of locally made programs are placed in CLIENT/ at the AppServer (parallel to WORK/).
2. Quit API PRO and start a new client.
3. Validate the changes were mirrored to the local image.

Delete some files
Fix-pack A13224 adds a special file called delete.lst to the root-directory of the package.
If present this file lists files to be deleted at the AppServer and clients.
Example: Delete hello.p and fiscal.rpt from the CR-templates.

1. Make a directory that will be the root of the package.
   C:\temp\load
2. Setup delete.lst:
3. Edit C:\temp\load\delete.lst
4. client/hello.p
5. work/hello.r
6. work/crprint/template/fiscal.rpt
7. Note: Notice the files are relative to the parent-directory of the application.
8. Make a description of the changes in package.txt:
    Edit C:\temp\load\package.txt
9. Set up compilation. By default all known programs will be compiled but the file compile.lst can be used to specify a smaller list (could be empty).
    Make C:\temp\load\compile.lst
10. Load the root-directory into CSM and go through the 3 steps (load, compile, deploy).
11. Quit API PRO and start a new client.
12. Validate the changes were mirrored to the local image.
    Now work\hello.r and work\template\fiscal.rpt should be deleted.
13. Delete the local root-directory.
    C:\temp\load

Reject login
Fix-pack A13224 (or A12526) adds a way to handle situations where the AppServer doesn’t respond.
This could be when system-work requires no users have access to the system (The AppServer is stopped), during power-failure or even if the system crashes.
If connection to AppServer APILIVE fails during start-up a series of WORK/CSM/noconnect* files are searched from the list below and the first found will be shown/opened:

   a. noconnect-APILIVE.html
   b. noconnect-APILIVE.htm
   c. noconnect-APILIVE.txt
   d. noconnect.html
   e. noconnect.htm
   f. noconnect.txt

WORK/CSM/noconnect.html:
<head><title>No connection</title></head>
<body>
<h1>No connection</h1>
<p>The AppServer you are trying to connect to is currently unavailable.</p>
<iframe src="http://apipro.com/rd/noconnect/apitest8.html" height="400" width="600"></iframe>
</body>
</html>

Tip: Using an html-file with IFRAME pointing to an external webserver allows status updates for most cases.

**CSM event**

Get signals from the CSM work-flow.
Example: Send a mail after every compile with the log-file attached.

CSM/event.cmd:

@ECHO OFF
REM event.cmd
REM Run scripts called from the different events of the CSM workflow
REM Parameters: event build buildhome
REM
IF NOT "%1"="compileend" GOTO END
IF EXIST "%~3\compile.err" C:\APISERVER\tools\blat.exe "%~3\compile.log" -to ppl@apipro.com -subject "Compile build %2 failed" -server 192.168.195.7 -f apitest6.rd.apipro.com@test.com
IF EXIST "%~3\compile.ok" C:\APISERVER\tools\blat.exe "%~3\compile.log" -to ppl@apipro.com -subject "Compile build
%2 finished succesfully" -server 192.168.195.7 -f apitest6.rd.apipro.com@test.com
:END
ENDLOCAL

**Overwrite file-type**

The client-image made of files from WORK/ at the AppServer is based on a series of rules, but some-times a file need to be included or rejected. Should a file be listed in CSM/work.lst the normal rule will not apply and instead by controlled by the attached TAG:

- **CLIENT** Part of the image
- **DELETED** Should be deleted
- **SERVER** Not part of the image

Example: Delete old.lg on clients and make sure new.log is not skipped (as other log-files).

CSM/work.lst (Sample):
new.log CLIENT
old.lg DELETED

**Advanced MSI-library**

It’s possible to have a local library for MSI-packages including the set-up they are based upon.

Follow these steps:

1. Dump the set-up from a system you want to manage locally into a directory using the “Initiator” menu-option. This will take the current files for the initiator package from the appserver you are connected to.
2. The files can now be edited for special set-up or requirements – like “default language”, “jodejock”-version...
3. Using the “Load set-up” button found at the “Initiator”-tab this local directory can then be loaded into any CSM-tool/appserver and then these files will be the source for the package generated by the “Generate”-button.
4. The entered values in the tab will be stored in client.log and client.wxs files found in the root of the local directory.

**Add an “interface”-database for compile**

Some installations have an additional interface-database running, which should be connected during the compile

Follow these steps:

1. Create an empty interface-database in the CSM/ directory
2. Load the provided interface.df
3. Make sure CSM/compile.pf exist. If this file is NOT found then take a copy from CSM/batch.pf

4. Add the database to CSM/compile.pf including option -RO.

CSM/compile.pf (Sample):

```
-db C:\APISERVER\APITEST8\DB\api3
-db C:\APISERVER\APITEST8\CSM\interface -RO
...
-b # Batch mode
-rx # Enable compile of xcode
-p csm/compile.p # Startup Program
```

**Check AppServer continuously for changes**

Using the CSM/check.cmd the “Task Scheduler” can be used to capture any changes made to WORK/ outside CSM on a regular basis:

Set current directory (“Start in”) to the WORK/-directory.
File-systems

This section lists the new files and directories used (Not all files are new).

Client

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( work/apipro.r )</td>
<td>Normal start-up procedure. Could be made “visible” showing the logo faster than today…</td>
</tr>
<tr>
<td>work/client.ini</td>
<td>Normal .ini-file configured for app-server connection including:</td>
</tr>
<tr>
<td></td>
<td>... noSplash=YES</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>work/client.pf</td>
<td>Normal .pf-file configured for app-server connection including:</td>
</tr>
<tr>
<td></td>
<td>-mc</td>
</tr>
<tr>
<td></td>
<td>-p csm/client.p</td>
</tr>
<tr>
<td>work/csm/admin.r</td>
<td>Administrator functionality.</td>
</tr>
<tr>
<td>work/csm/client.r</td>
<td>Client-initiator - called from &quot;-pf client.pf&quot;</td>
</tr>
<tr>
<td>work/csm/client.log</td>
<td>Contains the set-up from the last update. First line states:</td>
</tr>
<tr>
<td></td>
<td>Change-log, TimeStamp</td>
</tr>
<tr>
<td></td>
<td>Following line the files retrieved from the server.</td>
</tr>
<tr>
<td></td>
<td>If this file is lost (or removed) a complete client-image will be downloaded.</td>
</tr>
<tr>
<td></td>
<td>Otherwise the change-log is used to determine which files needs to be installed to be in sync with the server.</td>
</tr>
<tr>
<td>work/csm/client.ok</td>
<td>“Client image ok” signal.</td>
</tr>
<tr>
<td>work/csm/client.err</td>
<td>List of errors from latest start-up.</td>
</tr>
<tr>
<td>work/csm/client.lst</td>
<td>List of files downloaded from the app-server. Every download is appended to this file as long as change-log is</td>
</tr>
</tbody>
</table>
### Appserver

List of files used by CSM.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>work/csm/apipro.ico</td>
<td>Icon for window.</td>
</tr>
<tr>
<td>work/csm/apilogo.bmp</td>
<td>Logo-image shown at start-up.</td>
</tr>
</tbody>
</table>

### Appserver

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>client/</td>
<td>Home of any site-specific 4gl</td>
</tr>
<tr>
<td>client/client.lst</td>
<td>List (1 of 3) of programs to be included in “compile all programs” – listing the context of client/. The other parts are work/api-comp.tot and work/custom.lst.</td>
</tr>
<tr>
<td>csm/backup/</td>
<td>Home of the backups made when a build in installed at the app-server.</td>
</tr>
<tr>
<td>csm/backup/number/</td>
<td>Root of change <strong>number</strong> providing a base for the standard directories: work/ xcode/ The <strong>number</strong> is formatted using 8-digits.</td>
</tr>
<tr>
<td>csm/backup/number/bacup.lst</td>
<td>List of changes done at time of deployment. If the build is rolled back “CREATED”-actions will be converted to “DELETE” in the deploy.act-file of the new build.</td>
</tr>
<tr>
<td>csm/banner.bmp</td>
<td>Optional “Banner” for the MSI client-initiator. Size should be: 493x58 pixels. Sample:</td>
</tr>
<tr>
<td>csm/batch.log</td>
<td>Running log from batch processes</td>
</tr>
<tr>
<td>csm/batch.ini</td>
<td>-ini-file for batch</td>
</tr>
<tr>
<td>csm/batch.pf</td>
<td>-pf-file for batch</td>
</tr>
<tr>
<td>csm/build/</td>
<td>Home of the builds</td>
</tr>
<tr>
<td>csm/build/number/</td>
<td>Root of build <strong>number</strong> providing a base for the standard directories: client/ custom/ work/ xcode/ The <strong>number</strong> is formatted using 8-digits.</td>
</tr>
<tr>
<td>csm/build/number/compile.do</td>
<td>Trigger compile after load has finished.</td>
</tr>
<tr>
<td>csm/build/number/compile.err</td>
<td>Last compile had error(s). The fill will be deleted if the build is re-compiled.</td>
</tr>
<tr>
<td>Filename</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>csm/build/number/compile.lst</td>
<td>List of programs to be compiled</td>
</tr>
<tr>
<td>csm/build/number/compile.log</td>
<td>Log for last compile Note: This file lists the set-up of the compilation, including the FONTs used. For the application to look as intended the layout of DEFAULT and font-12 should list these values (the screen resolution would change this!)...</td>
</tr>
</tbody>
</table>
|                               | "FONT-SIZE (DEFAULT):" "APIPRO(50x16)"
|                               | "1234567890(70x16)"
|                               | ...
|                               | "FONT-SIZE (12):" "APIPRO(40x13)"
|                               | "1234567890(60x13)"
|                               | ...
<p>|                               | The fill will be deleted if the build is re-compiled.                                                                                                                                                      |
| csm/build/number/compile.ok    | Last compile went OK                                                                                                                                                                                       |
| csm/build/number/compile.tot   | Make a “total recompile”. This file is created by the package-loading if the message file states: “…total recompile…”                                                                                     |
| csm/build/number/client/       | On-site 4gl                                                                                                                                                                                                |
| csm/build/number/custom/       | Xcode customizations                                                                                                                                                                                        |
| csm/build/number/deploy.act    | Special actions at deploy-time: “DELETE” “file” will move the file to the back-up area.                                                                                                                   |
| csm/build/number/deploy.do     | Trigger deployment of the build after compile. NOTE: Should the compile end with errors, this “trigger” is ignored.                                                                                           |
| csm/build/number/deploy.err    | Build couldn’t be deployed due to some error.                                                                                                                                                              |
| csm/build/number/deploy.ok     | Build have been deployed.                                                                                                                                                                                   |
| csm/build/number/deploy.tot    | Trigger deployment of the build after compile regardless if any error occurred unlike deploy.do.                                                                                                          |
| csm/build/number/missing.lst   | Programs which did not produce a r-code. Can be used as source for a new compile.lst                                                                                                                                 |
| csm/build/number/package.dir   | Optional reference to a local directory loaded.                                                                                                                                                           |
| csm/build/number/package.txt   | Description of the build in plain text.                                                                                                                                                                    |
| csm/build/number/wo            | The resulting changes.                                                                                                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rk/</td>
<td>Xcode delivered from a fix-pack?</td>
</tr>
<tr>
<td>csm/build/number/xcode/</td>
<td>Locked(?) for making builds. Rejects makebuild() to make new areas. Unlock using unlockbuild(0).</td>
</tr>
<tr>
<td>csm/build.log</td>
<td>The lastest build reserved.</td>
</tr>
<tr>
<td>csm/build.lck</td>
<td>Locked(?) for making builds. Rejects makebuild() to make new areas. Unlock using unlockbuild(0).</td>
</tr>
<tr>
<td>csm/change/</td>
<td>Home of the changes documented</td>
</tr>
<tr>
<td>csm/change.log</td>
<td>High-water mark from last change</td>
</tr>
<tr>
<td>csm/change/number.list</td>
<td>List of changes going to change \textit{number}. The \textit{number} is formatted using 8-digits.</td>
</tr>
<tr>
<td>csm/check.cmd</td>
<td>Command for updating through ‘at’ \texttt{...proenv...} \texttt{cd install \texttt{check.cmd}}</td>
</tr>
<tr>
<td>csm/client.cmd</td>
<td>Part of the client-initiator. Should this file be missing the standard work/client.cmd is used as default.</td>
</tr>
<tr>
<td>csm/client.ico</td>
<td>Part of the client-initiator (MSI-version) used for making desktop-shortcut and menu-item in the start-up folder. Should this file be missing the standard work/image/api7.ico is used as default.</td>
</tr>
<tr>
<td>csm/client.ini</td>
<td>Part of the client-initiator Should this file be missing work/api7o.ini is used as default.</td>
</tr>
<tr>
<td>csm/client.pf</td>
<td>Part of the client-initiator. Should this file be missing the standard work/client.pf is used as default.</td>
</tr>
<tr>
<td>csm/compile.ini</td>
<td>-ini-file for compile (in batch mode)</td>
</tr>
<tr>
<td>csm/compile.pf</td>
<td>-pf-file for compile (in batch mode)</td>
</tr>
<tr>
<td>csm/dialog.bmp</td>
<td>Optional “Dialog” for the MSI client-initiator. Size should be: 493x312 pixels. Sample:</td>
</tr>
<tr>
<td>Filename</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>csm/event.cmd</td>
<td>If this command is present, it will receive “events” from the CSM-workflow. The command will be executed with current-directory set to “work/” and parameters: Unquoted character string in lowercase telling which event is launched: createbuild compilebegin compileend deploybegin deployend deletebuild createchange deletechange Number of the build or change being processed. Quoted path for the home of the build-workspace or filename of the changelog.</td>
</tr>
<tr>
<td>csm/event.log</td>
<td>Standard output of all csm/event.cmd calls made.</td>
</tr>
<tr>
<td>csm/init.cmd</td>
<td>Script to export files for the CSM-initiator</td>
</tr>
<tr>
<td>csm/launch.log</td>
<td>Log from launching batch-processes at the app-server</td>
</tr>
<tr>
<td>csm/server.lst</td>
<td>Last known status of work/. Used by “checkwork” to find the files that have been created, modified or changed.</td>
</tr>
<tr>
<td>csm/work.lst</td>
<td>List of files where the default logic to determine if a file is “CLIENT” or “SERVER” only can be overruled. The format of each line is &lt;filename&gt; &lt;action&gt;. Both values are in IMPORT-format (optional placed in quotes). Actions are one of: “CLIENT”: Send this file to the client. “SERVER”: Don’t send this file to the clients. Delete the file at the clients but don’t raise an error if it doesn’t exist.</td>
</tr>
<tr>
<td>custom/csttrans.db</td>
<td>Translation database.</td>
</tr>
<tr>
<td>Filename</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>init/</td>
<td>Home of exported CSM-initiator files.</td>
</tr>
<tr>
<td>xcode/apitrans.db</td>
<td>Translation database.</td>
</tr>
<tr>
<td>work/api-comp.tot</td>
<td>List (1 of 3) of programs to be included in “compile all programs” – listing the context of xcode/. The other parts are work/custom.lst and client/client.lst.</td>
</tr>
<tr>
<td>work/autocomp.lng</td>
<td>Configuration (text-file). Line 3 states languages for the translation.</td>
</tr>
<tr>
<td>work/client.cmd</td>
<td>Default client.cmd command file. If needed the file can be copied to csm/client.cmd and modified (the client-initiator will then use this).</td>
</tr>
<tr>
<td>work/client.pf</td>
<td>Default client.pf command file. If needed the file can be copied to csm/client.pf and modified (the client-initiator will then use this).</td>
</tr>
<tr>
<td>work/csm/compile.r</td>
<td>The CSM compiler, running in batch-mode at the appserver.</td>
</tr>
<tr>
<td>work/csm/noconnect-appserver.html</td>
<td>1. priority file which will be shown if connect fails during start-up for the specific AppServer called: appserver in the default file-viewer for the html-file-suffix – Typical InternetExplorer or Firefox Using “iframe” or “location” the actual status/reason could be broadcasted to the clients trying to connect. If this file is found the loop stops here – otherwise the client-initiator will look for the 2. priority.</td>
</tr>
<tr>
<td>work/csm/noconnect-appserver.htm</td>
<td>2. priority file which will be shown if connect fails during start-up. Only difference from 1. priority is the spelling of the suffix. If this file is found the loop stops here – otherwise the client-initiator will look for the 3. Priority.</td>
</tr>
<tr>
<td>work/csm/noconnect-appserver.txt</td>
<td>3. priority file which will be shown if connect fails during start-up. Now the default file-viewer for the txt-file-suffix (typical notepad) will be used. If this file is found the loop stops here – otherwise the client-initiator will look for the 4. Priority.</td>
</tr>
<tr>
<td>work/csm/noconnect.html</td>
<td>4. priority file which will be shown if connect fails during start-up for any appserver. If this file is found the loop stops here – otherwise the client-initiator will look for the 5. Priority.</td>
</tr>
<tr>
<td>work/csm/noconnect.htm</td>
<td>5. priority file which will be shown if connect fails during start-up. Only difference from 4. priority is the spelling of the suffix. If this file is found the loop stops here – otherwise the client-initiator will look for the 6. Priority.</td>
</tr>
<tr>
<td>work/csm/noconnect.txt</td>
<td>6. priority file which will be shown if connect fails during start-up. Run by work/maint/server/ttcsm.r and work/csm/client.r providing the logic at the appserver.</td>
</tr>
<tr>
<td>Filename</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>work/csm/server.log</td>
<td>States the current installed build.</td>
</tr>
<tr>
<td></td>
<td>This file is placed here (rather than in csm/) to capture a restore of</td>
</tr>
<tr>
<td></td>
<td>work/.</td>
</tr>
<tr>
<td>work/custom.lst</td>
<td>List (1 of 3) of programs to be included in “compile all programs” –</td>
</tr>
<tr>
<td></td>
<td>listing the context of custom/.</td>
</tr>
<tr>
<td></td>
<td>The other parts are work/api-comp.tot and client/client.lst.</td>
</tr>
<tr>
<td>work/maint/client/tcsm.r</td>
<td>The CSM admin-tool (Client logic).</td>
</tr>
<tr>
<td>work/maint/server/tcsm.r</td>
<td>The CSM admin-tool (Server logic).</td>
</tr>
</tbody>
</table>
Two *Note*-records are created to ensure single-user access to two critical flows.

**Build-lock**

Relates_to  LOCK  
Relate_Key  csm/server.p|build  
Type  1  

**Change-lock**

Relates_to  LOCK  
Relate_Key  csm/server.p|change  
Type  1
Command-line MSI-installation

The default value for the Destination folder can be given at command-line:

\texttt{msiexec /i DKK004.msi INSTALLDIR=C:\temp\abc}

Adding option /qn makes the installation go completely “silent”:

\texttt{msiexec /i DKK004.msi /qn INSTALLDIR=C:\temp\abc}

If you specify a restricted folder the installation will fail:
The installer has insufficient privileges to access this directory: C:\Program Files (x86)\AOS8999. The installation cannot continue. Log on as administrator or contact your system administrator.
Enhancements are related fix-packages

The table beneath lists the enhancements made to CSM over time and which fix-packages are available.

<table>
<thead>
<tr>
<th>Date</th>
<th>Enhancement</th>
<th>Fix-package</th>
<th>Required files</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.02.2013</td>
<td>Built-in unzip</td>
<td>A11837</td>
<td>define ttcsm</td>
</tr>
<tr>
<td>22.02.2013</td>
<td>Recommended fix for SP05</td>
<td>A11844</td>
<td>csm</td>
</tr>
<tr>
<td>27.02.2013</td>
<td>Deal with different drives when call OS-commands.</td>
<td>None</td>
<td>csm define ttcsm</td>
</tr>
<tr>
<td>19.03.2013</td>
<td>Bug-fix dealing with changelog/build number mismatch. New command-line: 'CheckClient' for Citrix?</td>
<td>A11925</td>
<td>csm</td>
</tr>
<tr>
<td>16.04.2013</td>
<td>Builds may contain new apitrans and/or csttrans which is then used for the compilation</td>
<td>A12027</td>
<td>csm</td>
</tr>
<tr>
<td>30.04.2013</td>
<td>Run fix-db in separate process during deploy. Setup in CSM/batch.ini and CSM/batch.pf</td>
<td>A12045</td>
<td>csm fix-db</td>
</tr>
<tr>
<td>04.06.2013</td>
<td>Trigger CSM/event.cmd</td>
<td>A12167</td>
<td>csm</td>
</tr>
<tr>
<td>19.06.2013</td>
<td>Trap STOP during load and leave. Previously the initiator would try to re-run (endlessly?) when the AppServer is down.</td>
<td>None</td>
<td>csm</td>
</tr>
<tr>
<td>Date</td>
<td>Description</td>
<td>Ticket</td>
<td>Author</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>02.07.2013</td>
<td>A few more files are considered “server-only”.</td>
<td>None</td>
<td>csm</td>
</tr>
<tr>
<td>02.07.2013</td>
<td>Keeping data and focus in administrator.</td>
<td>None</td>
<td>ttcsm</td>
</tr>
<tr>
<td>19.07.2013</td>
<td>Only one smaller bar in APIPRO71</td>
<td>A12272</td>
<td>csm</td>
</tr>
<tr>
<td>30.07.2013</td>
<td>Missing quotes for some events added.</td>
<td>A12284</td>
<td>csm</td>
</tr>
<tr>
<td>31.07.2013</td>
<td>Accumulative package with all changes (except api-comp.tot)</td>
<td>A12300</td>
<td>ALL</td>
</tr>
<tr>
<td>22.10.2013</td>
<td>Show a “noconnect”-file if connection fails</td>
<td>A12526</td>
<td>csm</td>
</tr>
<tr>
<td>21.02.2014</td>
<td>Skip &quot;Schema holder&quot;-files, 'empty.txt' and introduce 'CSM/work.lst'.</td>
<td>A12936</td>
<td>csm</td>
</tr>
<tr>
<td>24.03.2014</td>
<td>&quot;DELETED&quot; files from 'CSM/work.lst' are appended to &quot;download all&quot;.</td>
<td>A13001</td>
<td>csm</td>
</tr>
<tr>
<td>02.04.2014</td>
<td>‘StartClient’ and ‘ResetClient’ from login</td>
<td>(A13034)</td>
<td>csm</td>
</tr>
<tr>
<td>11.04.2014</td>
<td>'CSM/delete.lst'.</td>
<td>(A13112)</td>
<td>csm</td>
</tr>
<tr>
<td>15.04.2014</td>
<td>Shared MSI-packages</td>
<td>(A13153)</td>
<td>ttcsm</td>
</tr>
<tr>
<td>24.04.2014</td>
<td>Accumulative package with all changes since A12300</td>
<td>A13224</td>
<td>ALL</td>
</tr>
</tbody>
</table>

*Note: All changes since A12300.*
“Issues”

CSM/ have been deleted (but WORK/server.log survived).
CSM/build/00000001/compile.log looks like:

Compile start : 28/02/2014 09:15
Compile build  : 1
Server deployed: 219
Compile set-up failed

Or (older version installed):

Compile start : 28/02/2014 09:15
Session info:
  GUI MS-WIN95
  Screen resolution : 768 by 1024 pixels = 32 by 146,29 characters
  Session code Page : UTF-8
  Stream code Page  : UTF-8
  Numeric format    : EUROPEAN = thousand delimiter '
  ' decimal point ','
  Date format       : dmy
  Font (DEFAULT):   : APIPRO(50x16)1234567890(70x16)
  Font (0):         : APIPRO(42x14)1234567890(70x14)
  Font (1):         : APIPRO(40x13)1234567890(60x13)
  Font (2):         : APIPRO(42x14)1234567890(70x14)
  Font (3):         : APIPRO(42x14)1234567890(70x14)
  Font (4):         : APIPRO(40x13)1234567890(60x13)
  Font (5):         : APIPRO(50x16)1234567890(70x16)
  Font (6):         : APIPRO(46x13)1234567890(70x13)
The compile fails when the server already has a higher build deployed (219) than the new one (1).

The strange text “Builds2201” is from the code trying to build the PROPATH through the CSM-workarea from server+1 to build...

A simple solution could be to use the open “Reset server”.