Management of Linked Dyalog APL Workspaces (a work in progress)

Dr. Markos Mitsos
markos.mitsos@ergo.de

Deutsche Krankenversicherung AG DKV - ERGO, Actuarial Department

APL Germany — Berlin

ERGO

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

Necessary tools:

- file management (TortoiseSVN)
- test case management (DB2)

ERGO

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

Necessary tools:

- file management (TortoiseSVN)
- test case management (DB2)

About

Management of Linked Workspaces:

- schematic WS-structure
- function as WS builder
- automated tests

Necessary tools:

- file management (TortoiseSVN)
- test case management (DB2)

ERGO

Outline

ERGO

Outline

ERGO

Outline of section on framework and structure

In this section we outline:

Framework code management and WS structure

Build WS build for coding and debugging

Open open questions and problems

Outline of section on framework and structure

In this section we outline:

Framework code management and WS structure

Build WS build for coding and debugging

Open open questions and problems

Outline of section on framework and structure

In this section we outline:

Framework code management and WS structure

Build WS build for coding and debugging

Open open questions and problems

Outline of section on framework and structure

In this section we outline:

Framework code management and WS structure

Build WS build for coding and debugging

Open open questions and problems

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, “cooperating”
- coding
  - clear and clean distinction code vs. debug
  - decouple code, “save”, test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, "cooperating"
- coding
  - clear and clean distinction code vs. debug
  - decouple code, "save", test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, “cooperating”
- coding
  - clear and clean distinction code vs. debug
  - decouple code, “save”, test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Target WS framework

What is the framework for coding, debugging,. . . ?

- workspaces
  - Linked (text file based)
  - modular design, “cooperating”
- coding
  - clear and clean distinction code vs. debug
  - decouple code, “save”, test and deploy
- external tools
  - versioning in Tortoise SVN
  - test case management in DB2

ERGO

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns **<nsmn>**
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss **<nsfnX>**
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns **<nsmn>**
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss **<nsfnX>**
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns **<nsmn>**
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss **<nsfnX>**
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

Schematic workspace structure

What is the high level WS structure?

- objects
  - no vars/fns/ops under #
  - main ns <**nsmn**>
  - playground ns **test**
  - building instructions **build**
  - some reserved nss **globals**, **temp**, ...
  - foreign nss <**nsfnX**>
- sources
  - "own" nss primarily from working copy
  - "foreign" nss primarily from repository

WS structure
WS build structure

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

WS build for coding

How to create WS for coding?

- `)LOAD` special, small WS WS_BUILDER
- create temporal working copy of DIVERSES and Import it
- Link "own" nss bi-directionally
- create temporal working copies of "foreign" nss and Import them

setup for code

WS build for debugging

How to create WS for debugging?

- same as "code", Link "own" nss one-directionally (file → ws)

WS build for debugging

How to create WS for debugging?

- same as "code", Link "own" nss one-directionally (file → ws)

WS build for debugging

How to create WS for debugging?

- same as "code", Link "own" nss one-directionally (file → ws)

setup for debug
standrad setup

ERGO

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

ERGO

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

ERGO

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

Questions and problems

What can be done better?

- replace loading WS with scripting?
- wanted numbered WS logs, maybe named logs through scripting?
- occasionally code instability (auto format), especially when code+debug mixed
- occasionally listeners instability, possibly when coding while running obj for debug
- cannot grow warm with some aspects of Editor...

Outline of section on test and deployment

In this section we outline:

- **Framework** test construction, validation and repetition
- **Deployment** checking and "publishing" the workspace
- **Open** open questions and problems

Outline of section on test and deployment

In this section we outline:

Framework test construction, validation and repetition

Deployment checking and "publishing" the workspace

Open open questions and problems

Outline of section on test and deployment

In this section we outline:

Framework test construction, validation and repetition

Deployment checking and "publishing" the workspace

Open open questions and problems

Outline of section on test and deployment

In this section we outline:

Framework test construction, validation and repetition

Deployment checking and "publishing" the workspace

Open open questions and problems

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

Target test framework

What is the framework for debugging, testing,. . . ?

- debug and remember it
  - build test cases, try them out
  - when OK save arguments and results as target (SOLL)
- deploy if sure everything is OK
  - run all test cases, then `)SAVE` WS
  - document comparison between actual (IST) and target for IDP audit
- check when Dyalog or environment changes
  - run at least tests on base utilities when environment changes
  - document comparison for release audit

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

"Unit" tests and period system time tables

What kind of tests are used?

- "unit" test cases
  - want defined in- and output
  - but output too large to explicitly write down
- structure
  - use **build.check** to collect test case
  - one fn for each ns
  - one `:CASE` for each obj
  - remove/replace non-deterministic results prior to saving
- save in DB2
  - period system time very useful concept
  - implicitly hidden also useful
- (partly) show in Excel (for "externals")

ERGO

WS LATEX

Test interface between Dyalog and LaTeX

- ns **tex** is main ns of LATEX
- ns **tex.imp** contains objs for importing TeX files
- ARG_ANALYSIEREN parses arguments of an LaTeX macro
- CMD_ANALYSIEREN parses document concerning a list of LaTeX macros

WS LATEX

Test interface between Dyalog and LaTeX

- ns **tex** is main ns of LATEX
- ns **tex.imp** contains objs for importing TeX files
- ARG_ANALYSIEREN parses arguments of an LaTeX macro
- CMD_ANALYSIEREN parses document concerning a list of LaTeX macros

WS LATEX

Test interface between Dyalog and LaTeX

- ns **tex** is main ns of LATEX
- ns **tex.imp** contains objs for importing TeX files
- ARG_ANALYSIEREN parses arguments of an LaTeX macro
- CMD_ANALYSIEREN parses document concerning a list of LaTeX macros

WS LATEX

Test interface between Dyalog and LaTeX

- ns **tex** is main ns of LATEX
- ns **tex.imp** contains objs for importing TeX files
- ARG_ANALYSIEREN parses arguments of an LaTeX macro
- CMD_ANALYSIEREN parses document concerning a list of LaTeX macros

WS LATEX

Test interface between Dyalog and LaTeX

- ns **tex** is main ns of LATEX
- ns **tex.imp** contains objs for importing TeX files
- ARG_ANALYSIEREN parses arguments of an LaTeX macro
- CMD_ANALYSIEREN parses document concerning a list of LaTeX macros

Argument parsing ARG_ANALYSIEREN

Test parsing of arguments

- in- and output of fn small, could be written down
- contains range of checks on arguments (also to be tested)
- fn meant as subfunction
- needed because of recursion (`\section{Abschnitt}` vs. `\rbw[\alter[0]][np]`)

Argument parsing ARG_ANALYSIEREN

Test parsing of arguments

- in- and output of fn small, could be written down
- contains range of checks on arguments (also to be tested)
- fn meant as subfunction
- needed because of recursion (`\section{Abschnitt}` vs. `\rbw[\alter[0]][np]`)

Argument parsing ARG_ANALYSIEREN

Test parsing of arguments

- in- and output of fn small, could be written down
- contains range of checks on arguments (also to be tested)
- fn meant as subfunction
- needed because of recursion (`\section{Abschnitt}` vs. `\rbw[\alter[0]][np]`)

Argument parsing ARG_ANALYSIEREN

Test parsing of arguments

- in- and output of fn small, could be written down
- contains range of checks on arguments (also to be tested)
- fn meant as subfunction
- needed because of recursion (\section{*Abschnitt*} vs. \rbw[*alter[0]*][*np*])

Argument parsing ARG_ANALYSIEREN

Test parsing of arguments

- in- and output of fn small, could be written down
- contains range of checks on arguments (also to be tested)
- fn meant as subfunction
- needed because of recursion (`\section{`*Abschnitt*`}` vs. `\rbw[`*\alter[0]*`][`*np*`]`)

Document parsing CMD_ANALYSIEREN

Test parsing of documents

- needs list of macros, write down
- needs document in special form, better keep in TeXstudio and import on the fly
- result can be lengthy and complicated

Document parsing CMD_ANALYSIEREN

Test parsing of documents

- needs list of macros, write down
- needs document in special form, better keep in TeXstudio and import on the fly
- result can be lengthy and complicated

Document parsing CMD_ANALYSIEREN

Test parsing of documents

- needs list of macros, write down
- needs document in special form, better keep in TeXstudio and import on the fly
- result can be lengthy and complicated

Document parsing CMD_ANALYSIEREN

Test parsing of documents

- needs list of macros, write down
- needs document in special form, better keep in TeXstudio and import on the fly
- result can be lengthy and complicated

Questions and problems

What can be done better?

- need bigger argument/result fields, go to *VARCHAR(12000)* or directly to *CLOB(5M)*?
- need string, own formatting function or Array Notation (sometimes verbose)?
- compare strings or recreate structures (Adám: first, but re-format!)?
- also deploy other WS forms (runtime, dll,...)?

Questions and problems

What can be done better?

- need bigger argument/result fields, go to *VARCHAR(12000)* or directly to *CLOB(5M)*?
- need string, own formatting function or Array Notation (sometimes verbose)?
- compare strings or recreate structures (Adám: first, but re-format!)?
- also deploy other WS forms (runtime, dll,...)?

Questions and problems

What can be done better?

- need bigger argument/result fields, go to *VARCHAR(12000)* or directly to *CLOB(5M)*?
- need string, own formatting function or Array Notation (sometimes verbose)?
- compare strings or recreate structures (Adám: first, but re-format!)?
- also deploy other WS forms (runtime, dll,...)?

ERGO

Questions and problems

What can be done better?

- need bigger argument/result fields, go to *VARCHAR(12000)* or directly to *CLOB(5M)*?
- need string, own formatting function or Array Notation (sometimes verbose)?
- compare strings or recreate structures (Adám: first, but re-format!)?
- also deploy other WS forms (runtime, dll,. . .)?

Questions and problems

What can be done better?

- need bigger argument/result fields, go to *VARCHAR(12000)* or directly to *CLOB(5M)*?
- need string, own formatting function or Array Notation (sometimes verbose)?
- compare strings or recreate structures (Adám: first, but re-format!)?
- also deploy other WS forms (runtime, dll,. . .)?

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

ERGO

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

ERGO

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

ERGO

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

Conclusion

Open problems:

- scripting and multiple logs
- code stability and Editor behaviour
- bigger arguments/results in DB2
- format of test cases and Array Notation
- deployment of more WS variants

begin

Overview of examples and illustrations

- WS structure
- WS build structure
- setup for code
- setup for debug
- standrad setup

Schematic structure of workspace

#.<nsmn>	code proper of workspace
#.build	building instructions and tests cases
#.test	alternatives, ideas,. . .
#.<nsfnX>	imported foreign namespace[s]

#.temp	temporary, for example during build
#.globals	global object, for example COM

◂ Schematic workspace structure

ERGO

Schematic structure of namespace #.build.prms

nsl	source and target of relevant namespaces
wsn	target name of deployed workspace
lx	latent expression of deployed workspace
nsmn	main "own" namespace of workspace

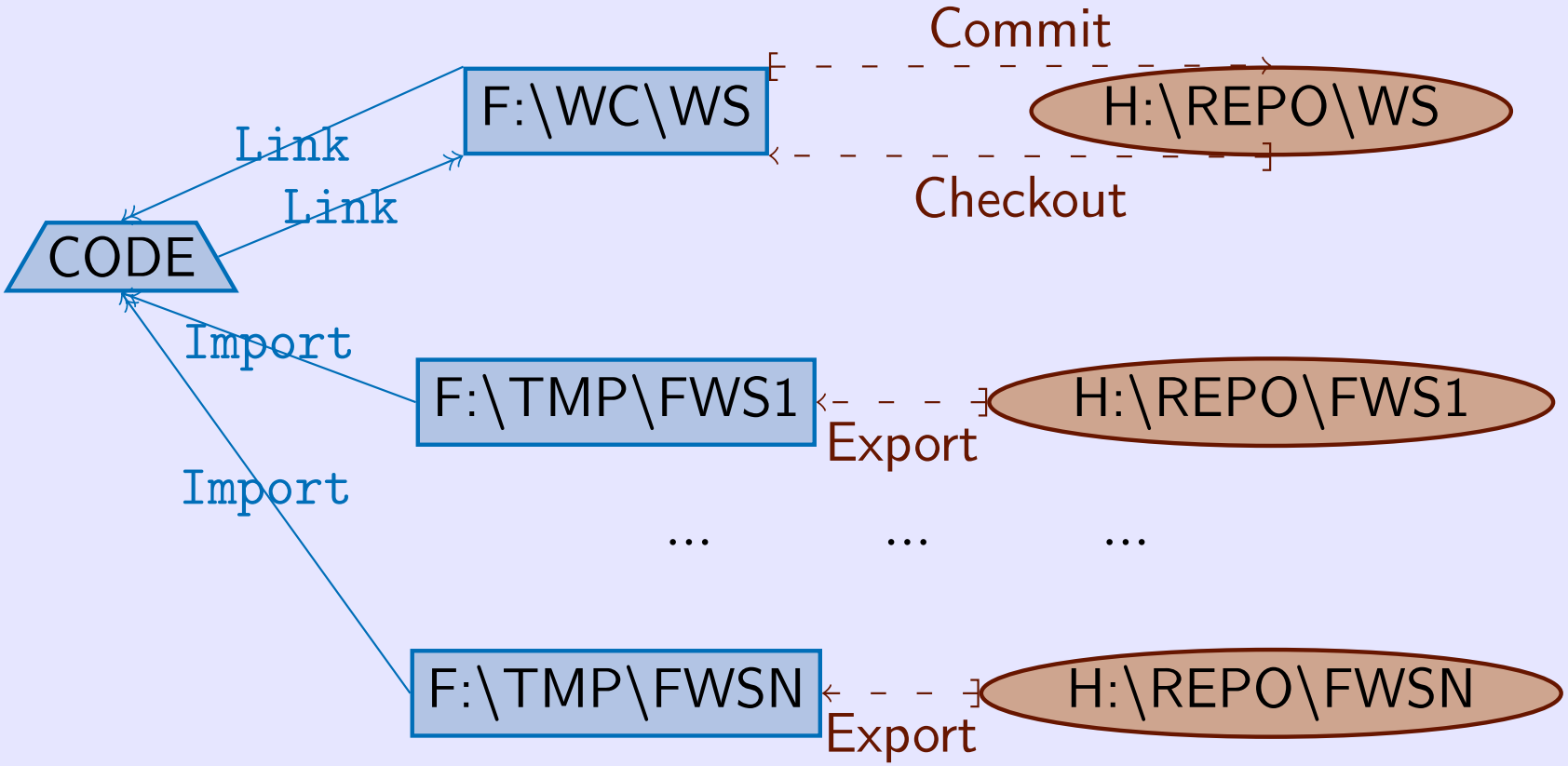
prvdr	DB2 provider for test case management
sqlid	DB2 SQLID for test case management
xln	Excel target for test validation documentation

◂ Schematic workspace structure

ERGO

Code workspace

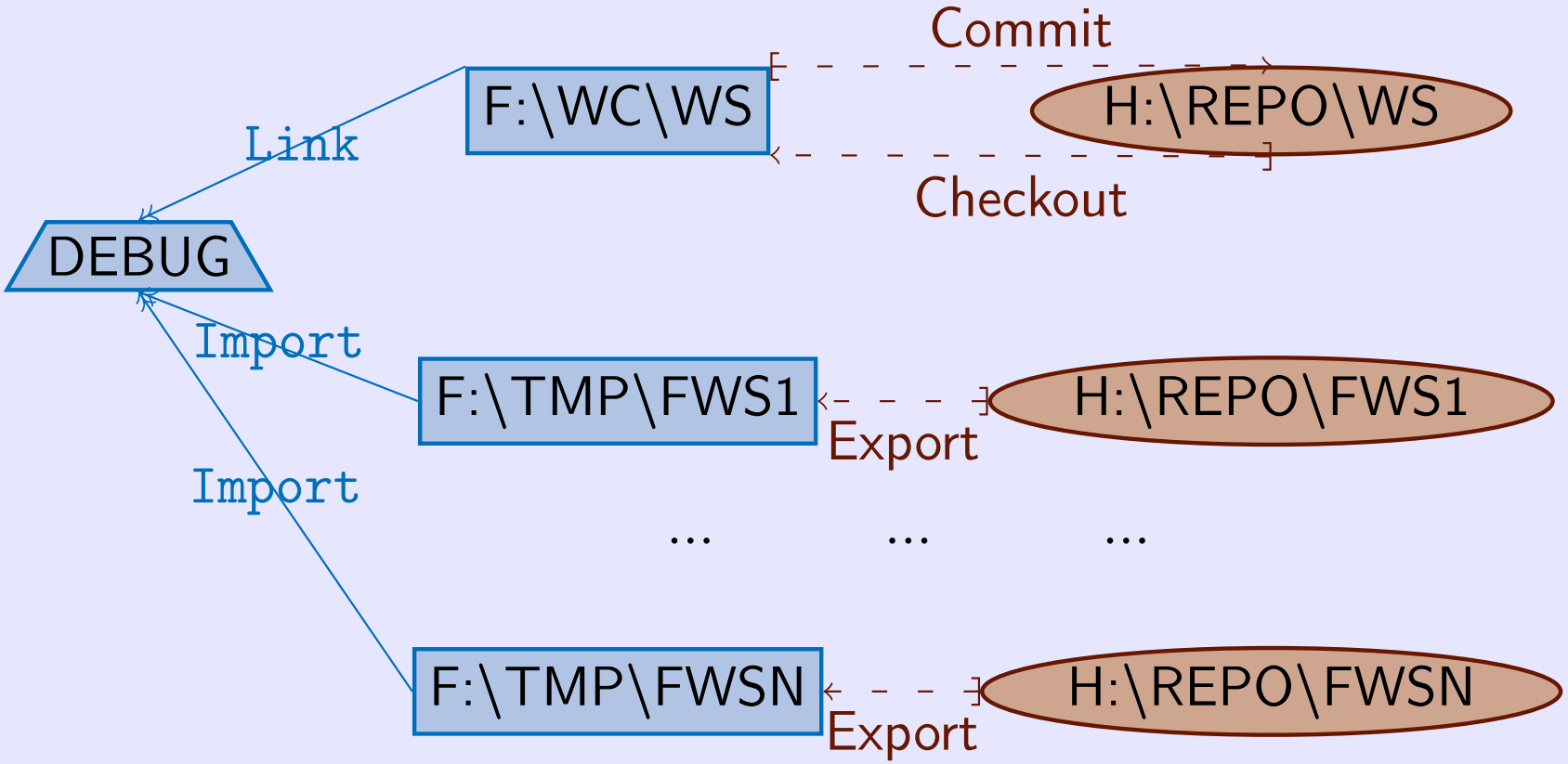
Own working copy is Linked bi-directionaly, Rest Imported:



◂ WS build for coding

Debug workspace

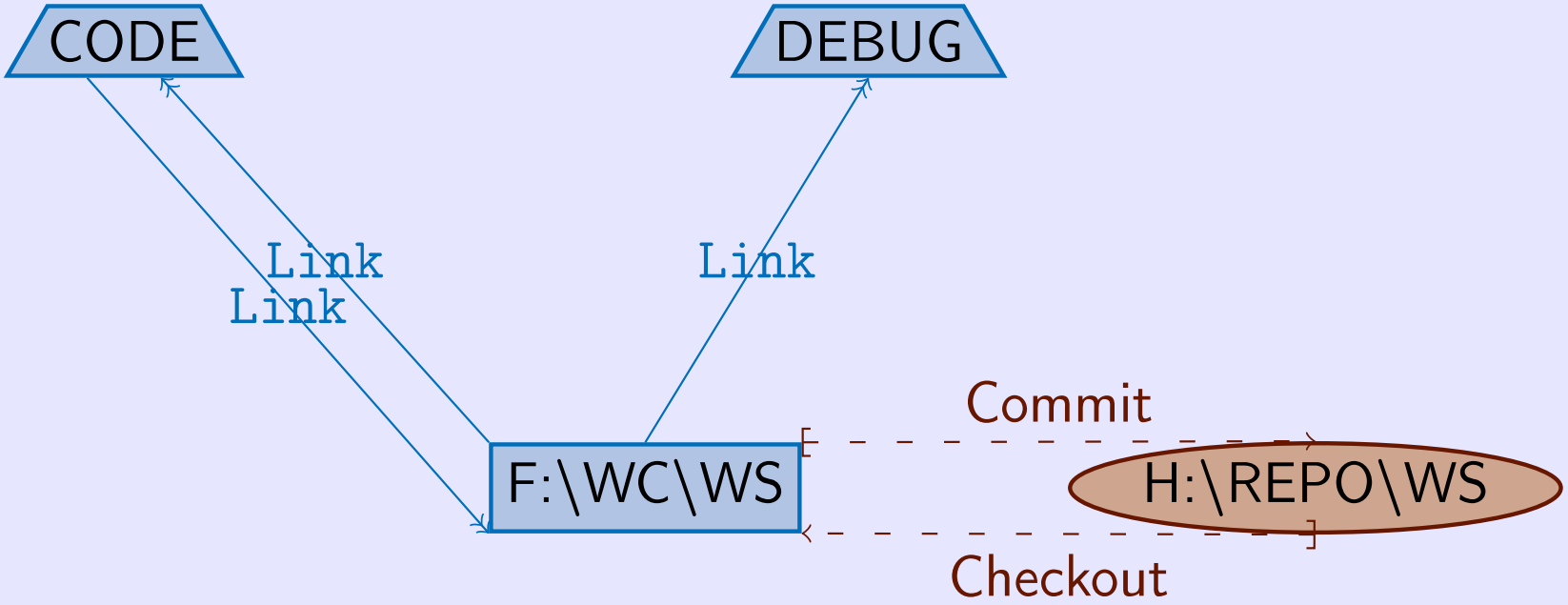
Own working copy is Linked uni-directionaly, Rest Imported:



◂ WS build for debugging

Code and debug workspace

Usual work setup, two WS in tandem:



◂ WS build for debugging