07/08/2021 (Version 2.0.1)

Work with output data never been so simple!

WHAT IS METAOUTPUT?

MetaOutput is attempt to reinvent the most used tools of any **Integrated Development Environment** (***IDE***).

Functionality of all these tools is moved to single tool.

Result became simpler, smaller, faster and more functional than existing equivalents.

WHY METAOUTPUT WAS MADE?

Development of standard output tool is frozen, but it’s too week to be enough.

Nobody improve it and it was a reason why **MetaOutput** was created.

WHY METAOUTPUT IS DIFFER FROM ALL?

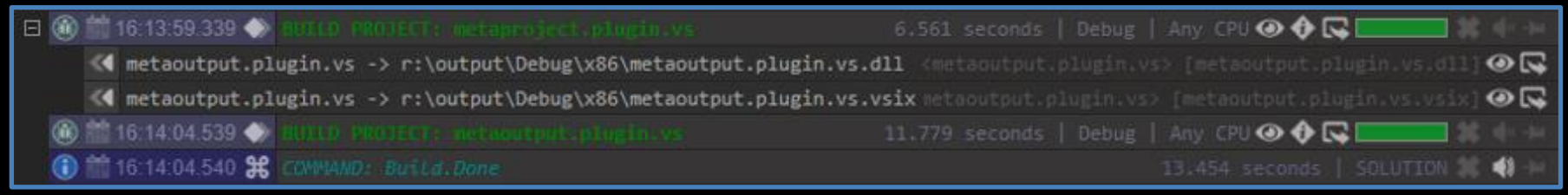
MetaOutput is based on this approach:

Output data / log files - IT'S NOT A TEXT!

It's look strange but seems almost nobody use this idea in own software products.

WHAT IS BASE OF METAOUTPUT?

Each trace message is complicated object with own visualization:



It consist of next blocks:

- Content (*main text*);
- Comment (*text on the right*);
- Message source and type (*icons on the left*);
- Timestamp (*date and time of on the left*);
- Operations with message (*buttons on the right*).

WHICH GOALS SHOULD BE ACHIEVED?

MetaOutput should solve only 3 problem: simplicity, simplicity and simplicity.

- Simply to learn it;
- Simply to use it;
- Simply to change it.

GOAL: SUPPORT ANY DATA COMPLEXITY

Doesn't matter which kind of data is used, it should be visualized in natural form.

Everything must be clear without any learning and any explanations.

GOAL: SUPPORT ANY DATA TYPES

Text messages, hierarchical data, audio, video, pictures or even **HTML** pages should be visualized as is directly from **MetaOutput**.

GOAL: VISUALIZE ANY DATA SOURCES

Data from different sources should be performed and visualized in one place simultaneously.

GOAL: EVERYTHING IS POSSIBLE TO SEE

All data should have visual representation and be represented in clear form.

Everything should be obvious from first look.

GOAL: EVERYTHING IS POSSIBLE TO HEAR

Everything that requires especial attention should have possibility to be played by **Text To Speech** mechanism.

GOAL: EVERYTHING IS POSSIBLE TO FIND

Any trace message should be possible to find by text substring.

Search initiating and switching to found result should be obvious and fast.

GOAL: EVERYTHING IS POSSIBLE TO FILTER

User should have possibility to change visibility of concrete messages or even whole sources.

Don’t want to see something – make it invisible…

GOAL: EVERYTHING IS POSSIBLE TO HIGHLIGHT

Everything that have special interest can be highlighted by desire of user.

Highlighted message or not highlighted should be obvious from first look.

GOAL: EVERYTHING SHOULD BE IN ONE PLACE

Many tools, much place for it, switching between it, rearrangement of it…

Forget it forever!

Only one place where all data will be collected and visualized.

GOAL: EVERYTHING IN REAL TIME

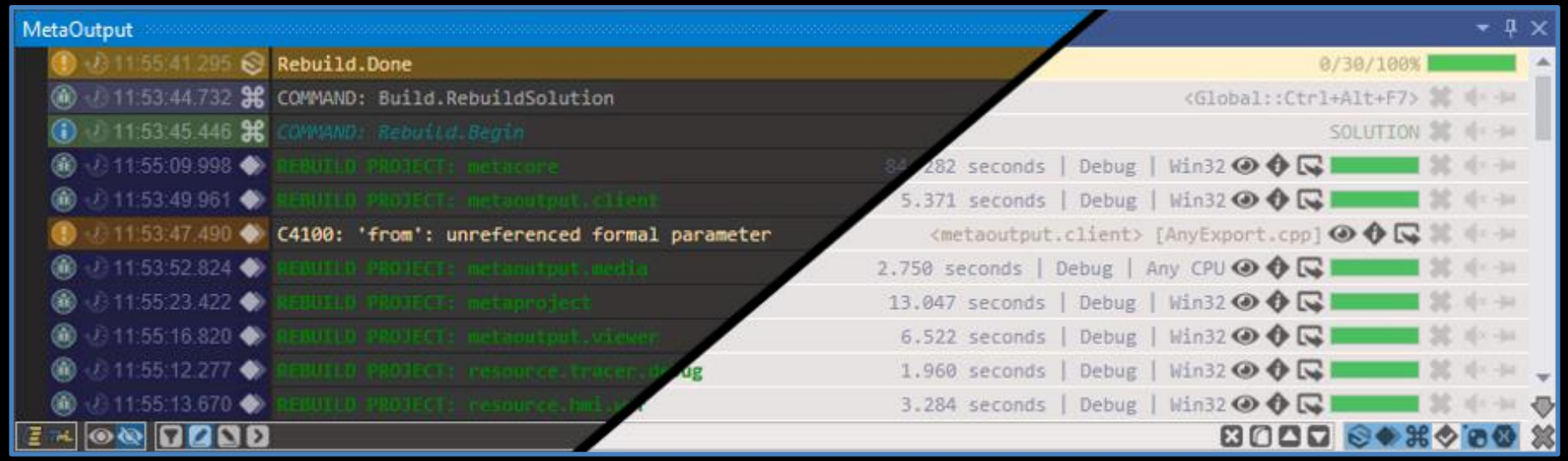
Data collecting, transforming, filtering, highlighting and visualization should work simultaneously in real time without any delays.

Compilation process in parallel or executing of debugging program should not have any influence on work of application.

Everything must work quickly!

ADAPTATION TO COLOR SCHEMA

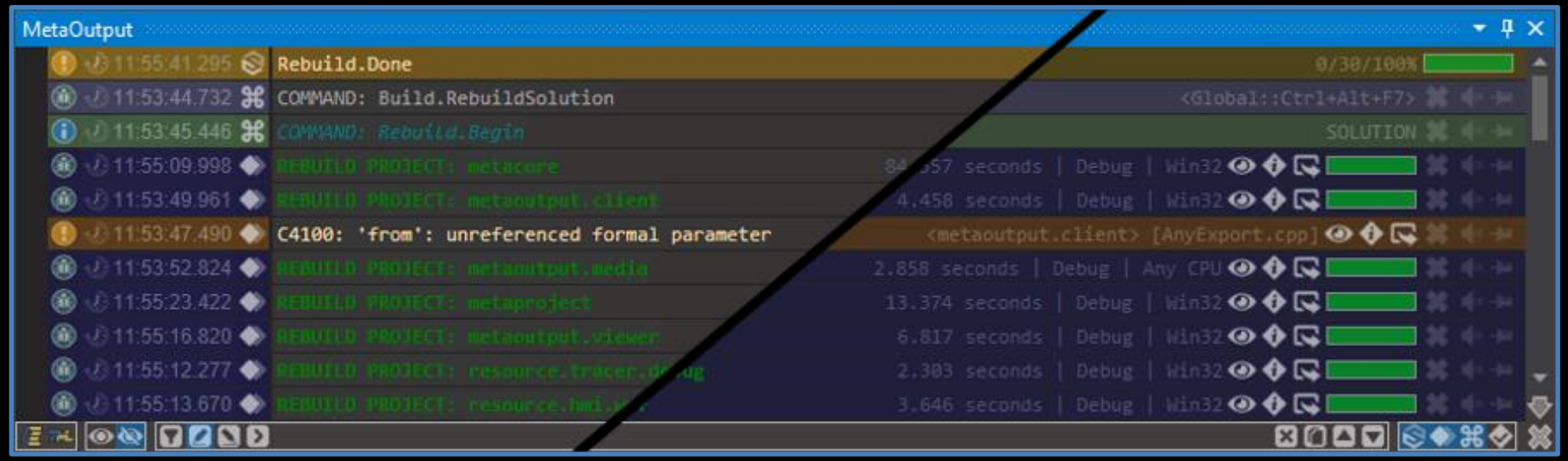
MetaOutput supports native for **IDE** color schemas:



This screenshot demonstrates **dark** and **light** color schemas.

ADAPTATION TO USER PREFERENCES

MetaOutput supports several message line styles (***disabled, half-line, full-line***):



This screenshot demonstrates **half-line** and **full-line** visualization.

VISUALIZATION OF DEFAULT SOURCES

MetaOutput supports these sources for all **IDE**s by default:

VISUALIZATION OF ADDITIONAL SOURCES

List of supported sources can be easily extended.

These sources are additionally available in **Visual Studio** by default:

Also, own sources can be added using **TML**.

This is example how to do it:

```
@@@SOURCE.APPEND <<<OWN_SOURCE_NAME>>> /some_folder/own_source_name.png
```

VISUALIZATION OF DEFAULT TYPES

MetaOutput supports these types for all **IDE**s by default:

VISUALIZATION OF SPECIFIC TYPES

List of supported types can be easily extended using **TML**.

This is example how to do it:

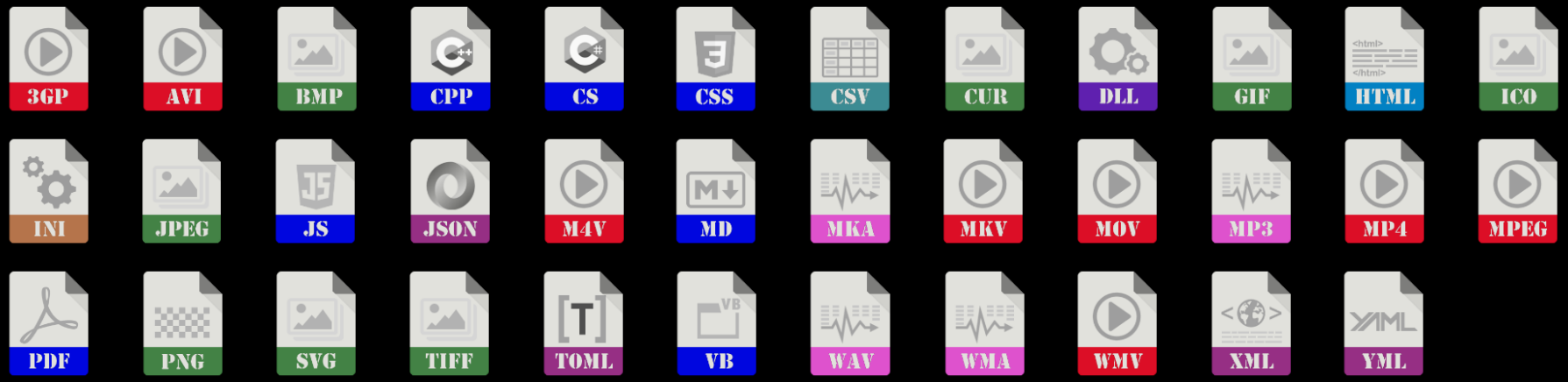
```
@@@TYPE.APPEND <<<OWN_TYPE_NAME>>> /some_folder/own_type_name.png
```

VISUALIZATION OF UNKNOWN FILE FORMATS

MetaOutput doesn’t know all existing file formats, it is impossible!

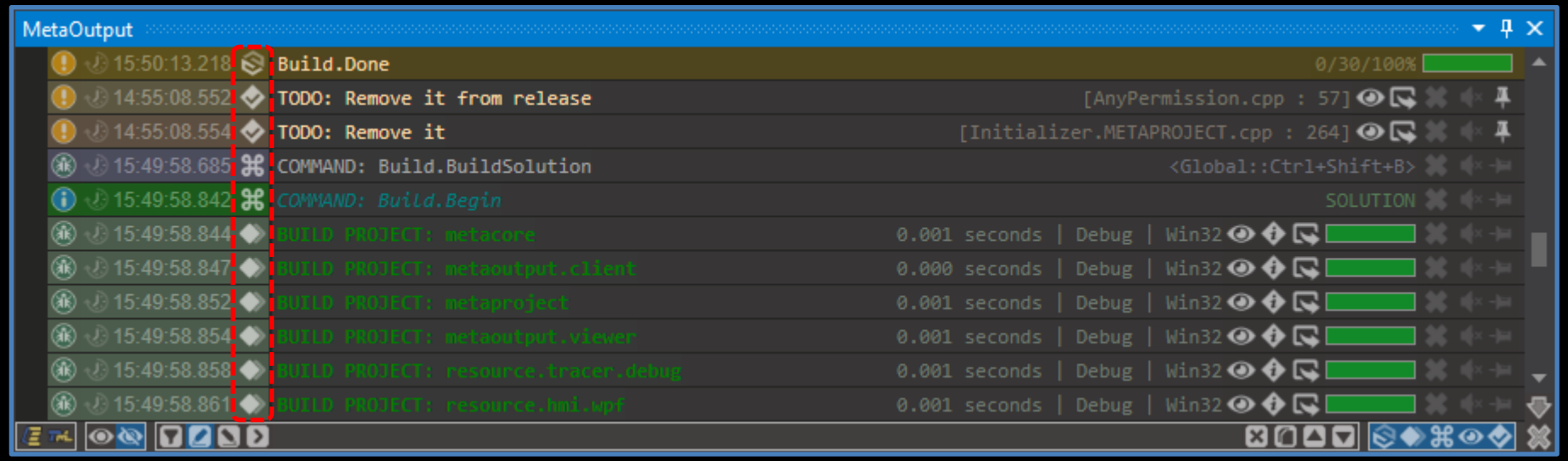
However, it became possible using extensions to this product.

Now is written support for next file formats:



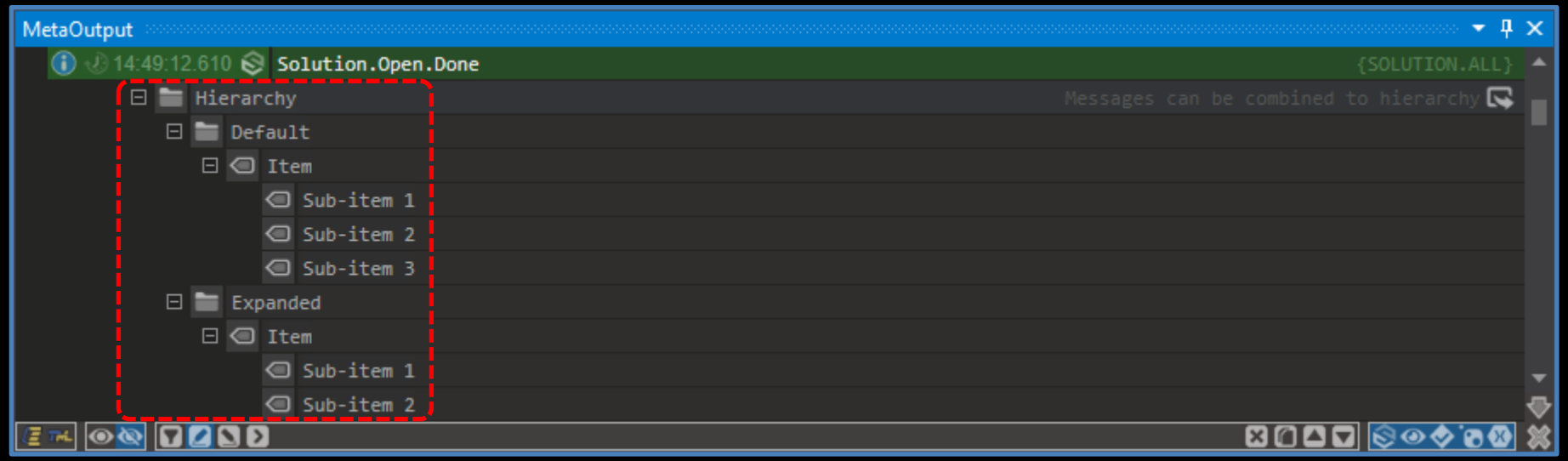
VISUALIZATION OF SEVERAL SOURCES

Trace messages can be visualized from several sources simultaneously:



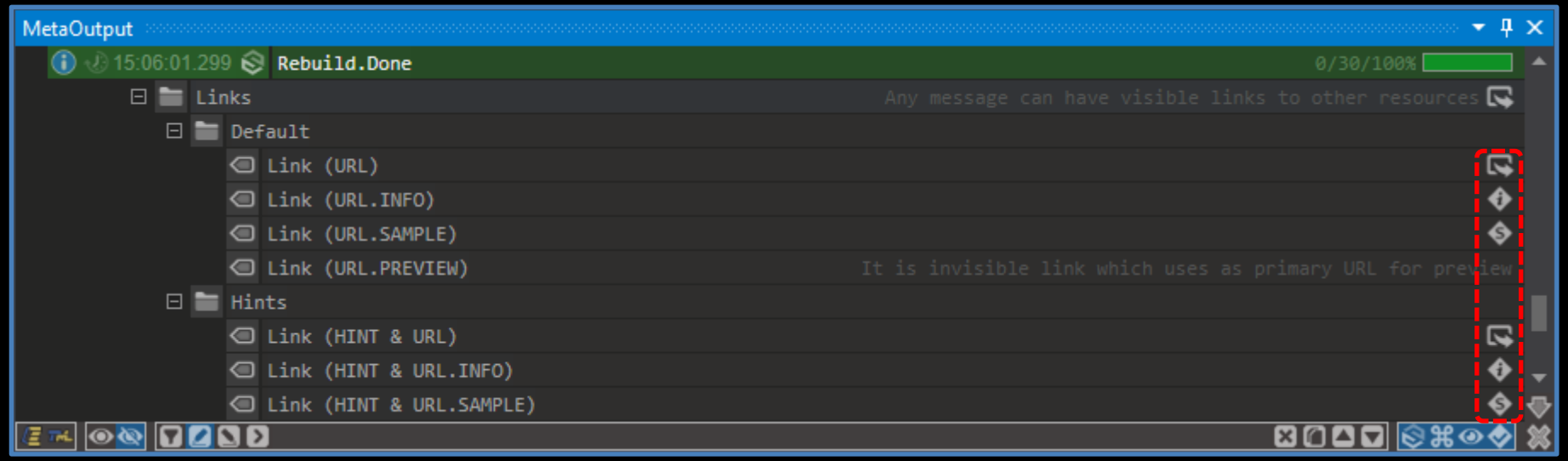
VISUALIZATION OF HIERARCHICAL DATA

Trace messages can be represented as hierarchical data with unlimited deep:



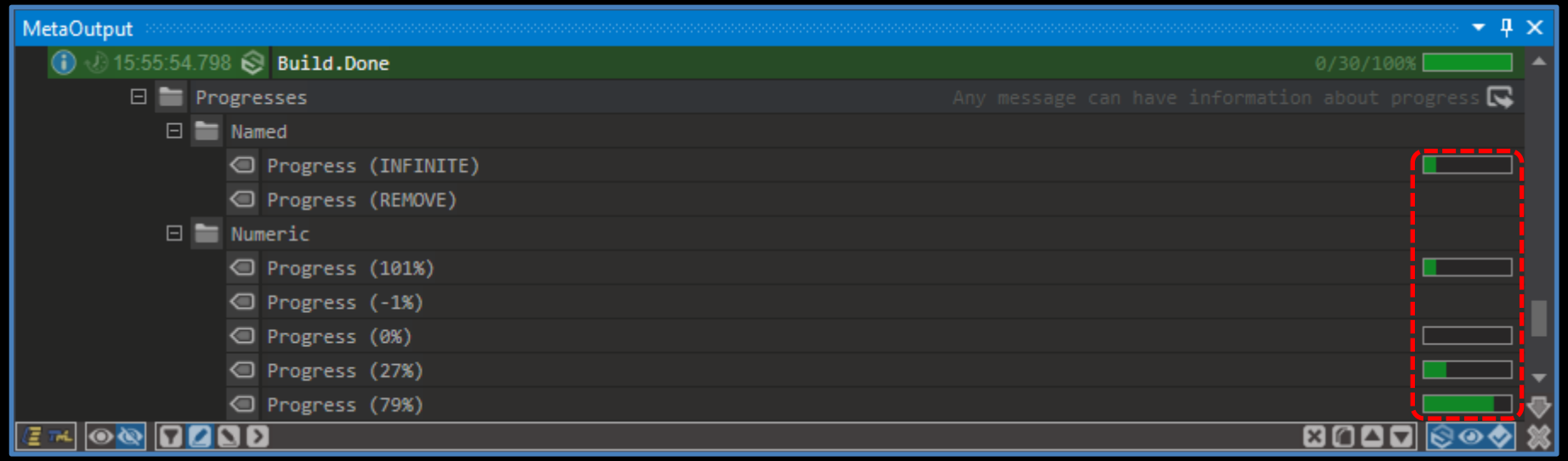
VISUALIZATION OF LINKS TO FILES

Trace messages can have different link buttons:



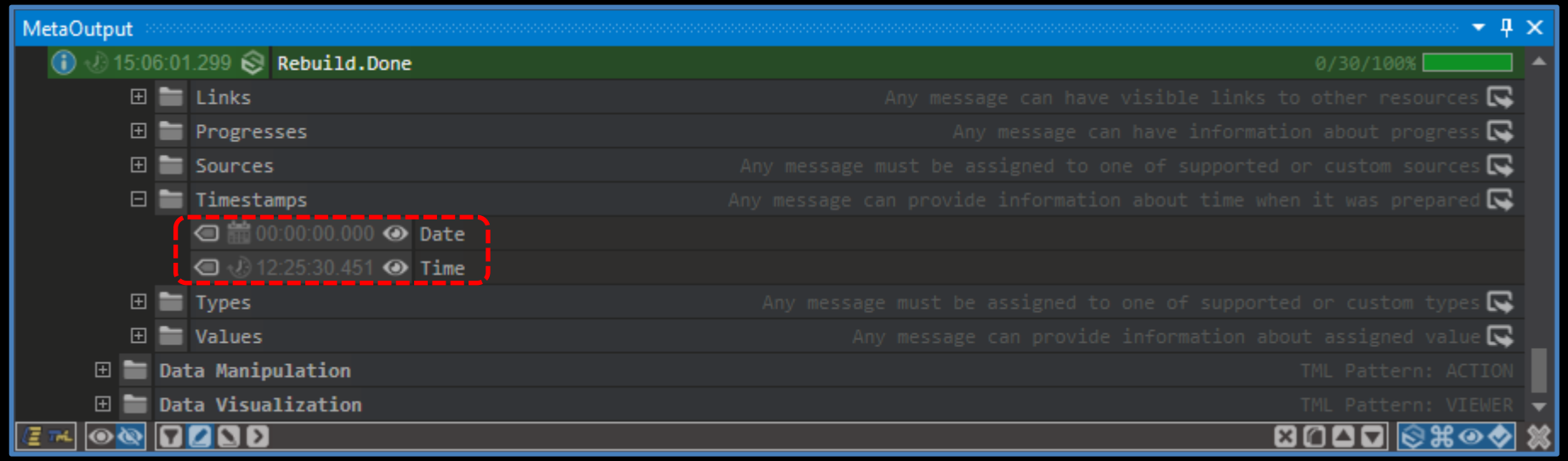
VISUALIZATION OF EXECUTION PROGRESS

Trace messages can visualize continuous operations:



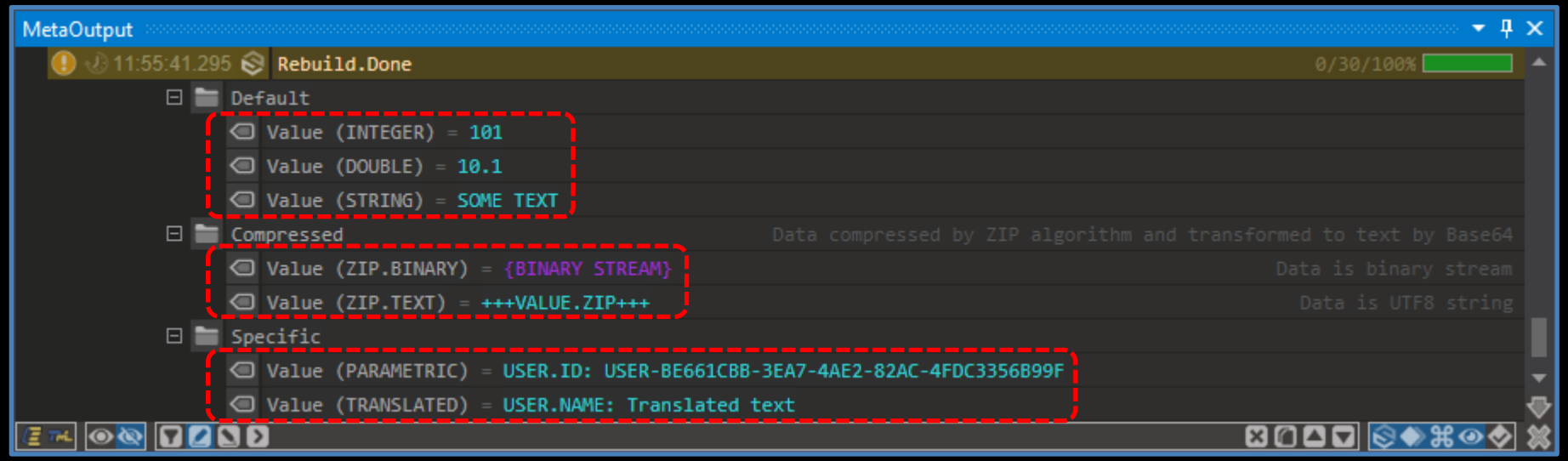
VISUALIZATION OF TIMESTAMPS

Trace messages can have timestamps with date and time:



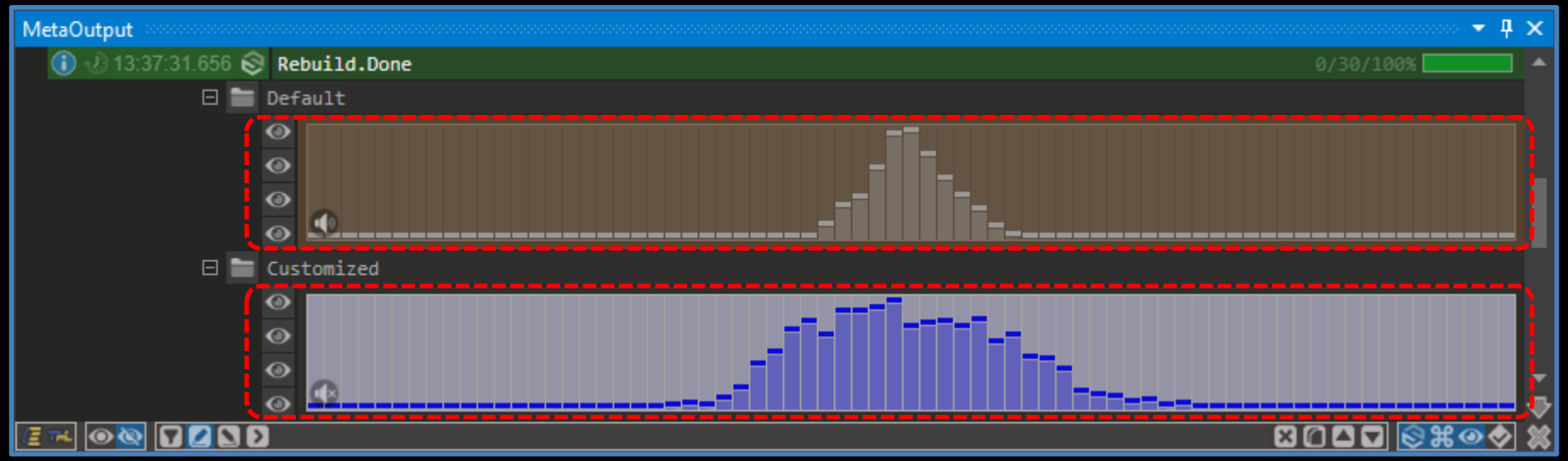
VISUALIZATION OF NAMED VALUES

Trace messages can be represented as **NAME** = **VALUE**:



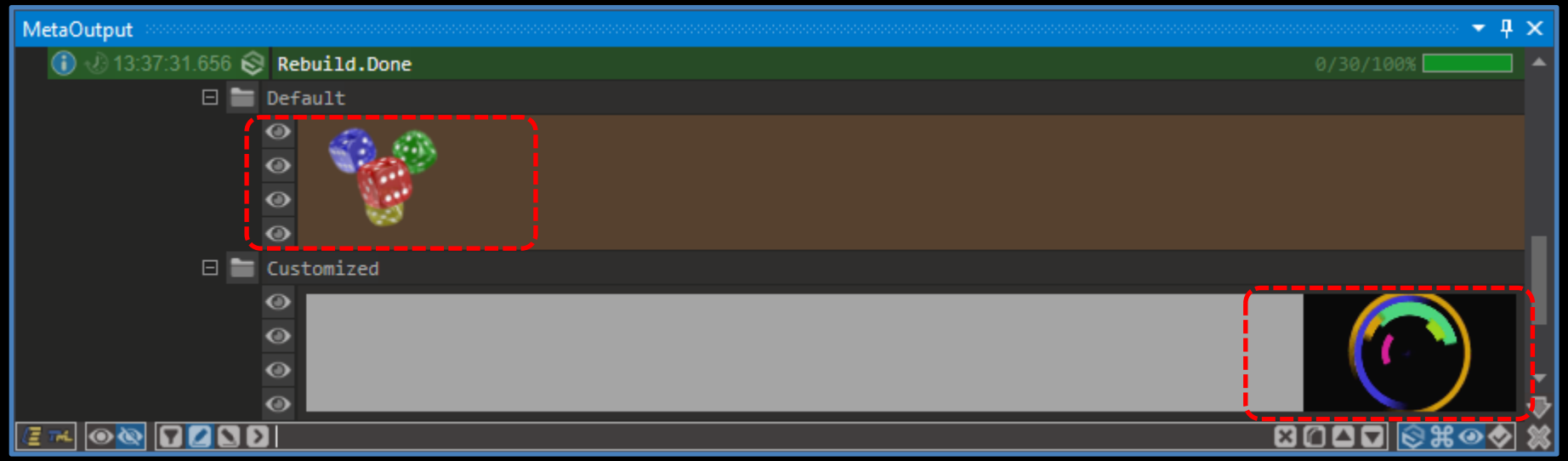
VISUALIZATION OF AUDIO FILES

Trace messages can visualize playing of audio files (*placed locally or remote*):



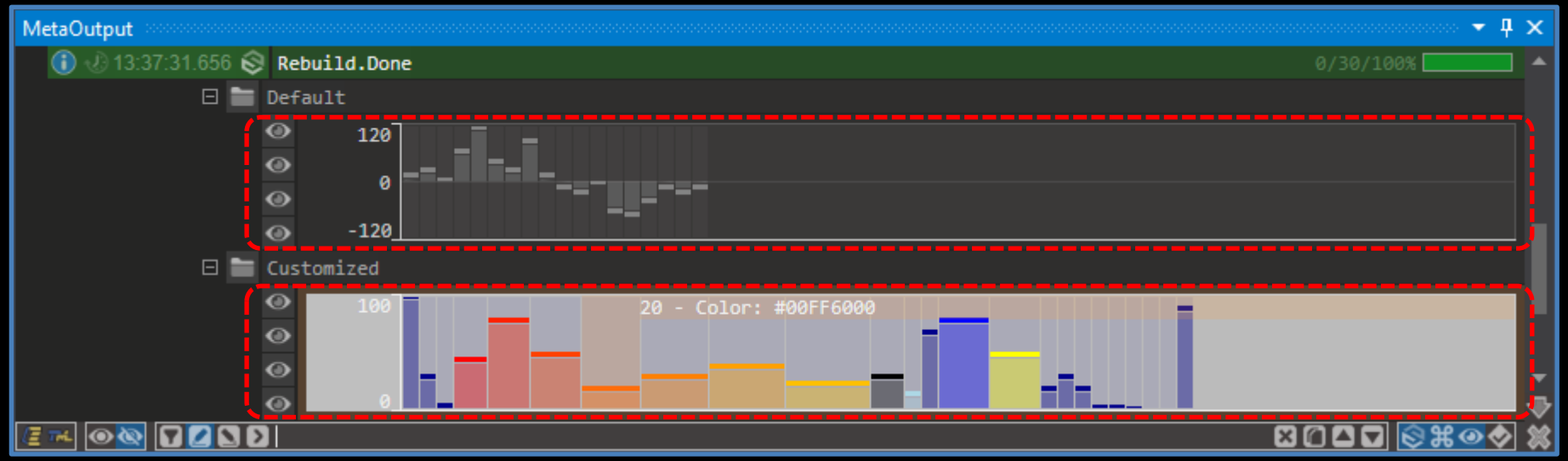
VISUALIZATION OF PICTURES

Trace messages can visualize picture files (*placed locally or remote*):



VISUALIZATION OF CHARTS

Trace messages can visualize chart diagrams:

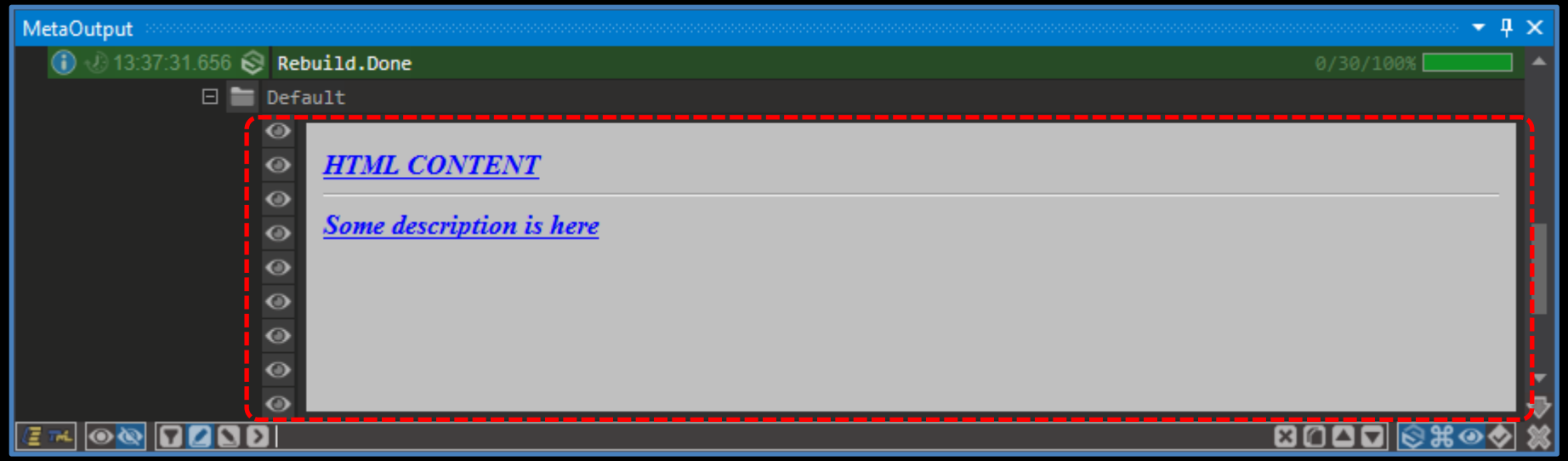


VISUALIZATION OF HTML FILES

Trace messages can visualize **HTML** files (*placed locally or remote*):

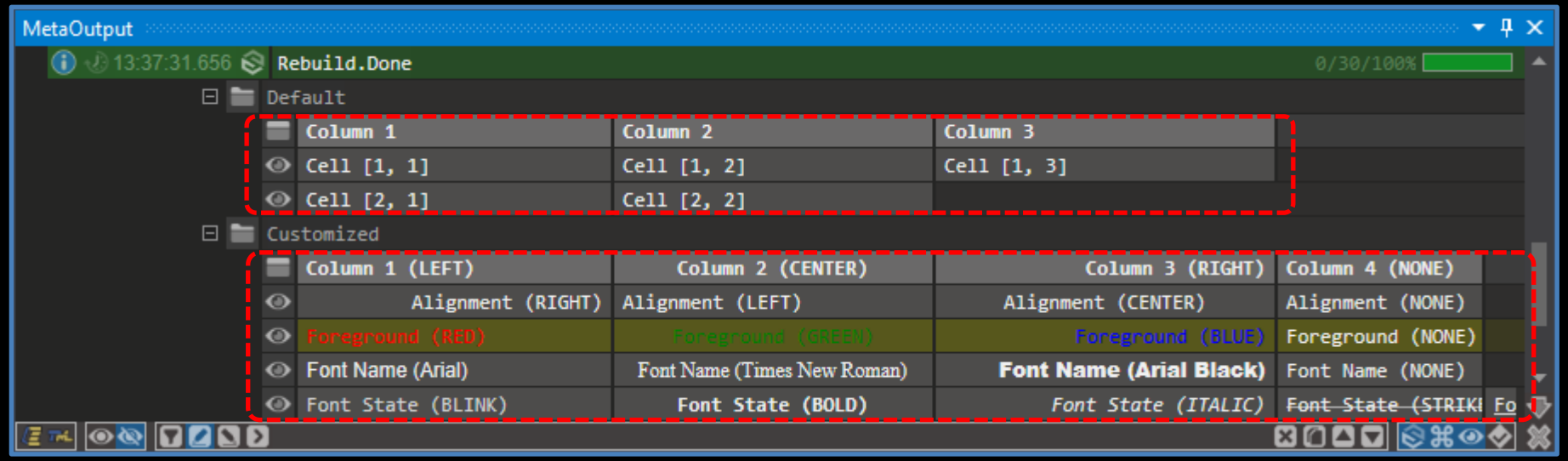
VISUALIZATION OF HTML CONTENT

Trace messages can visualize **HTML** content directly injected into the message:



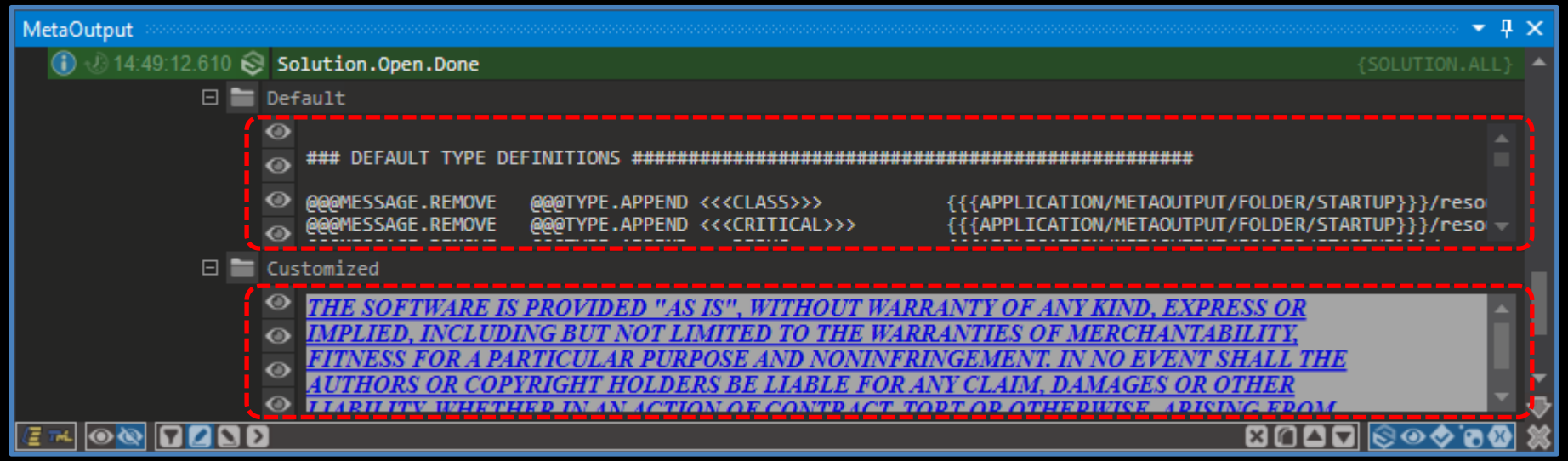
VISUALIZATION OF TABLES

Trace messages can visualize data tables:



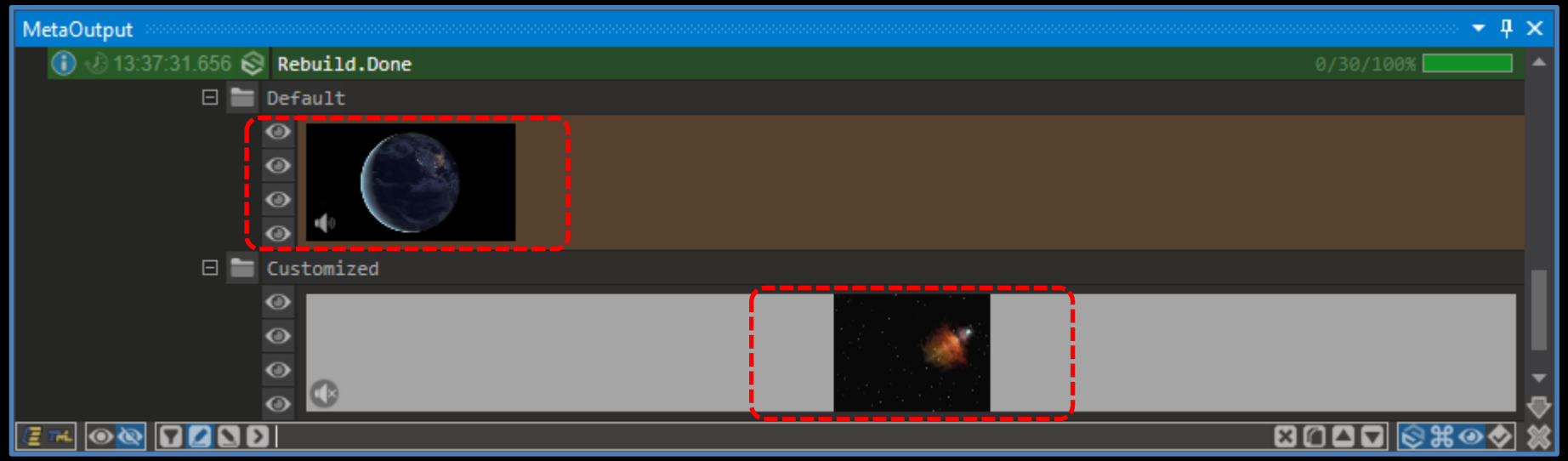
VISUALIZATION OF TEXT FILES

Trace messages can visualize text files (*placed locally or remote*):



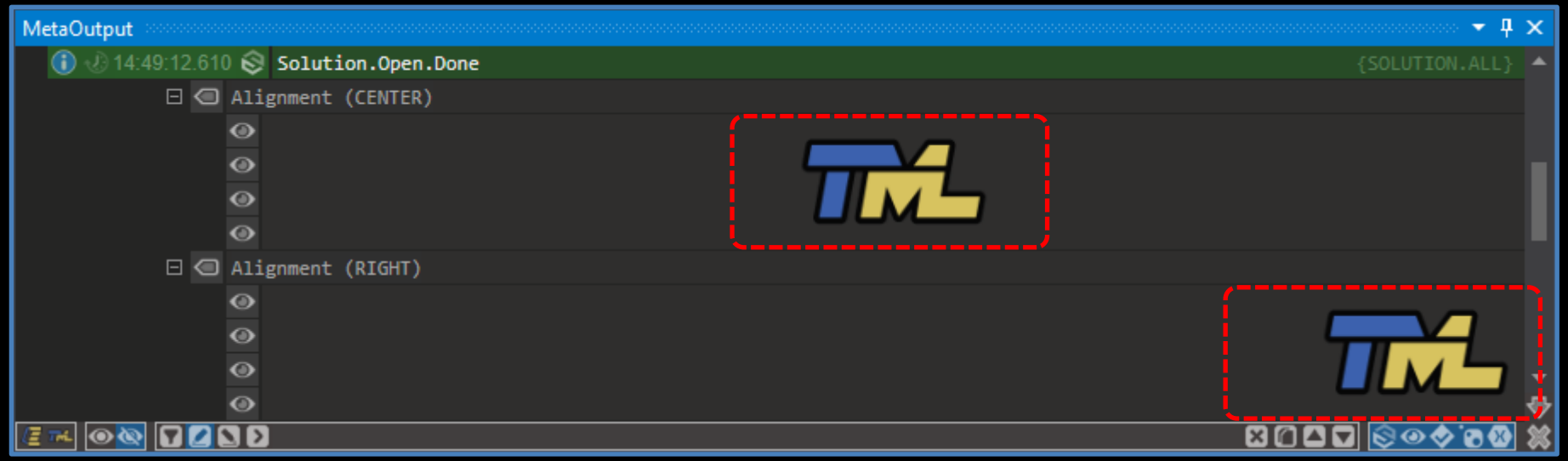
VISUALIZATION OF VIDEO FILES

Trace messages can visualize playing of video files (*placed locally or remote*):



VISUALIZATION ATTRIBUTES: ALIGNMENTS

Some complex controls can have different alignments:

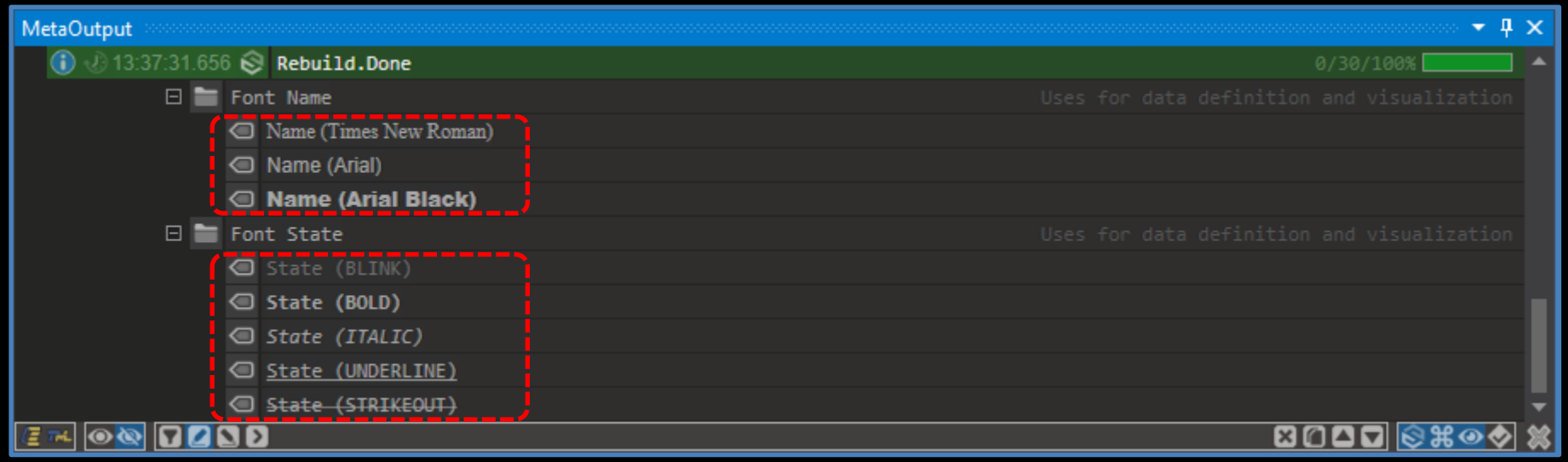


VISUALIZATION ATTRIBUTES: COLORS

Visual elements can have own background and foreground colors:

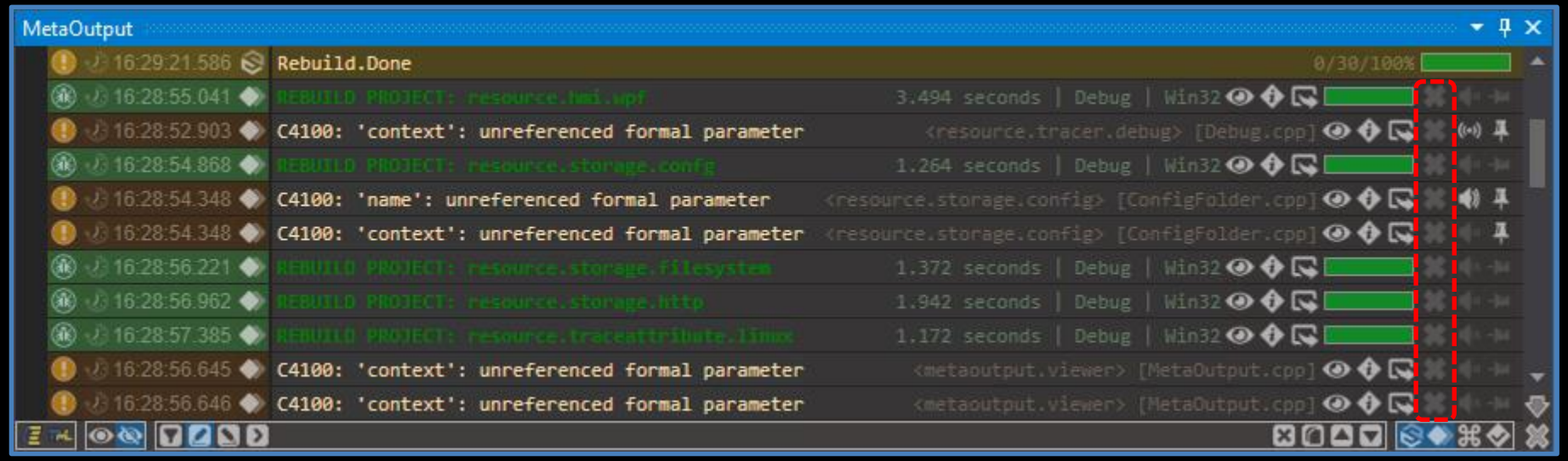
VISUALIZATION ATTRIBUTES: FONTS

Textual visual elements can use different font names and styles:



WHAT IS MESSAGE HIDING?

Any trace message can be hidden by desire of user:

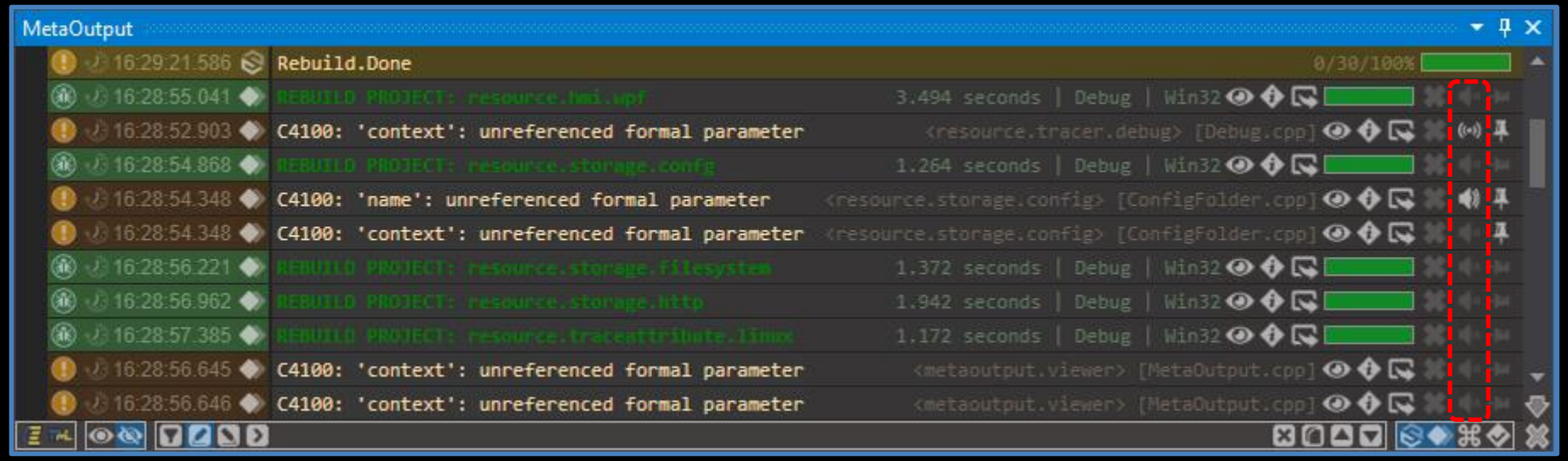


By pressing of close button will be hidden all similar messages.

Visibility of hidden messages is possible to restore.

WHAT IS MESSAGE PLAYING?

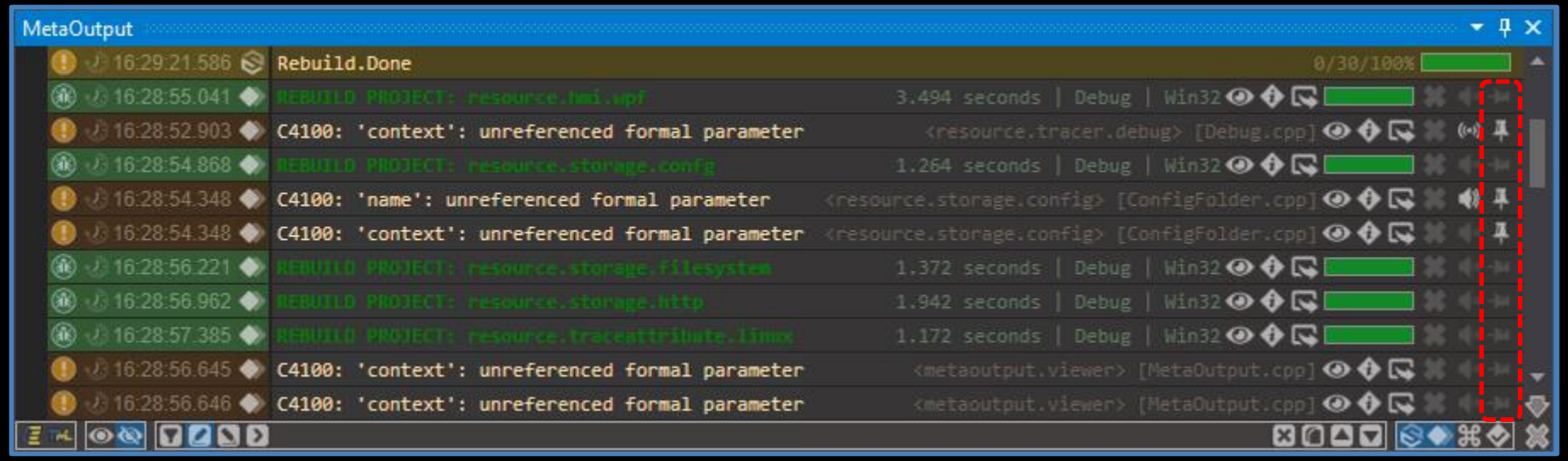
Any trace message can be played by desire of user:



Such messages will be played by **Text To Speech** mechanism.

WHAT IS PINNED MESSAGES?

Any trace message can be pinned by desire of user:



Such messages will not be removed on "**Clear all**" operation.

WHAT IS FILTERING?

Filtering its mechanism which simplify analyze of incoming data.

It can reduce amount of visible data and highlight some interesting messages.

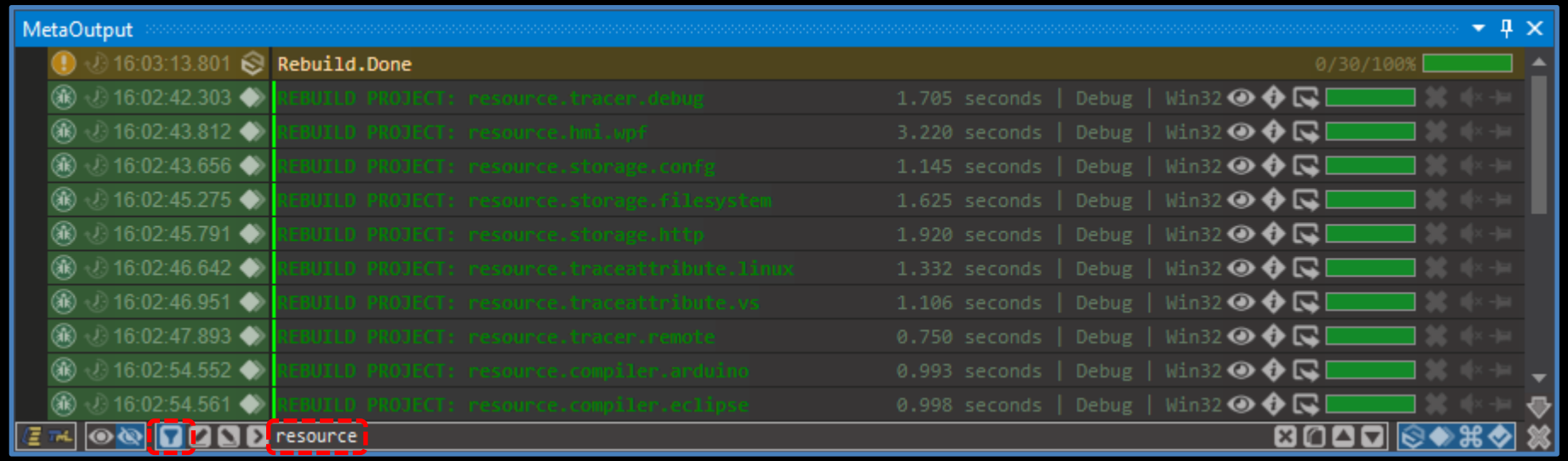
Reducing of data is possible:

- by hiding of not interesting messages;
- by hiding of not interesting sources;
- by hiding of all messages which don’t contain some substring (*like **grep***).

Everything is working in real time.

FILTER MODE: FILTER

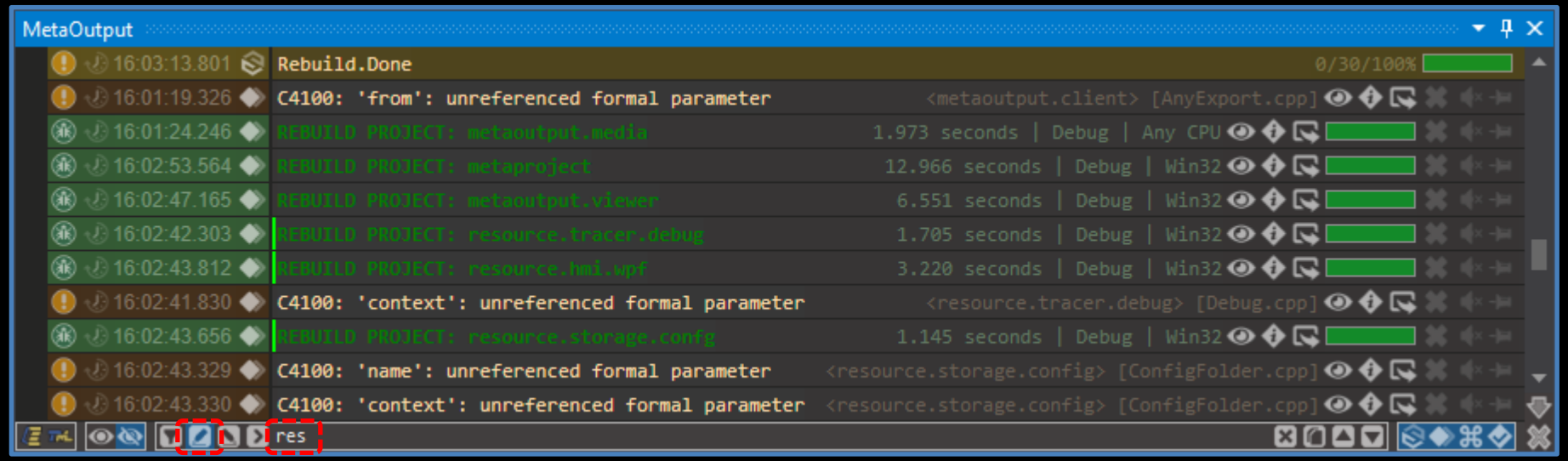
Showing only that messages which have required substring:



New messages will be also filtered.

FILTER MODE: CONTENT HIGHLIGHT

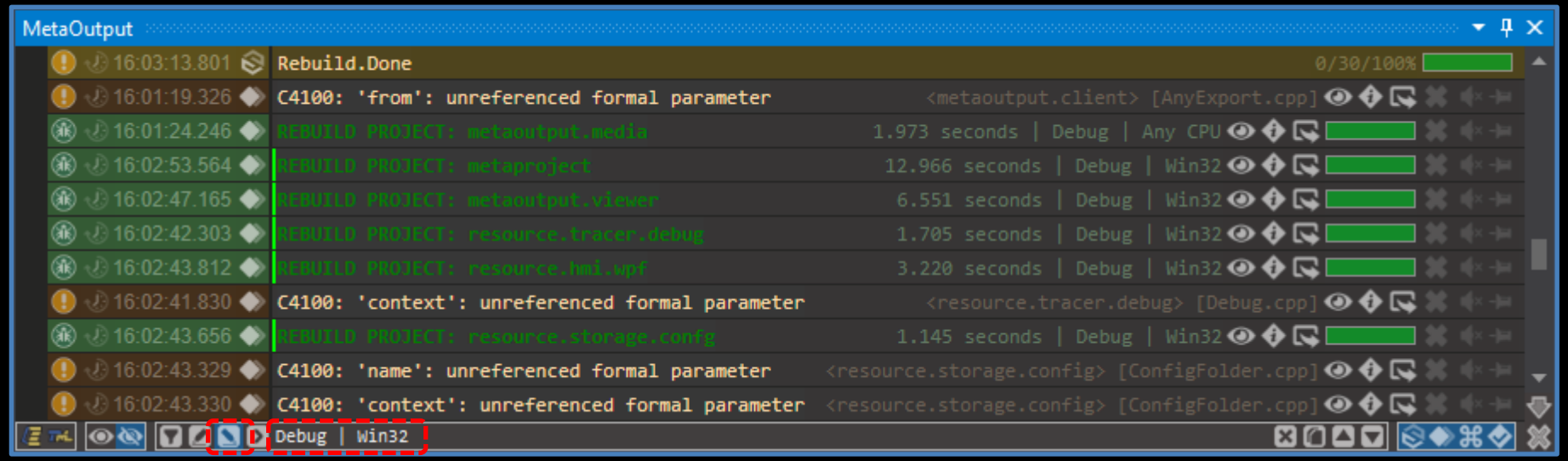
Highlighting all messages which have required substring in content:



New messages will be also highlighted.

FILTER MODE: COMMENT HIGHLIGHT

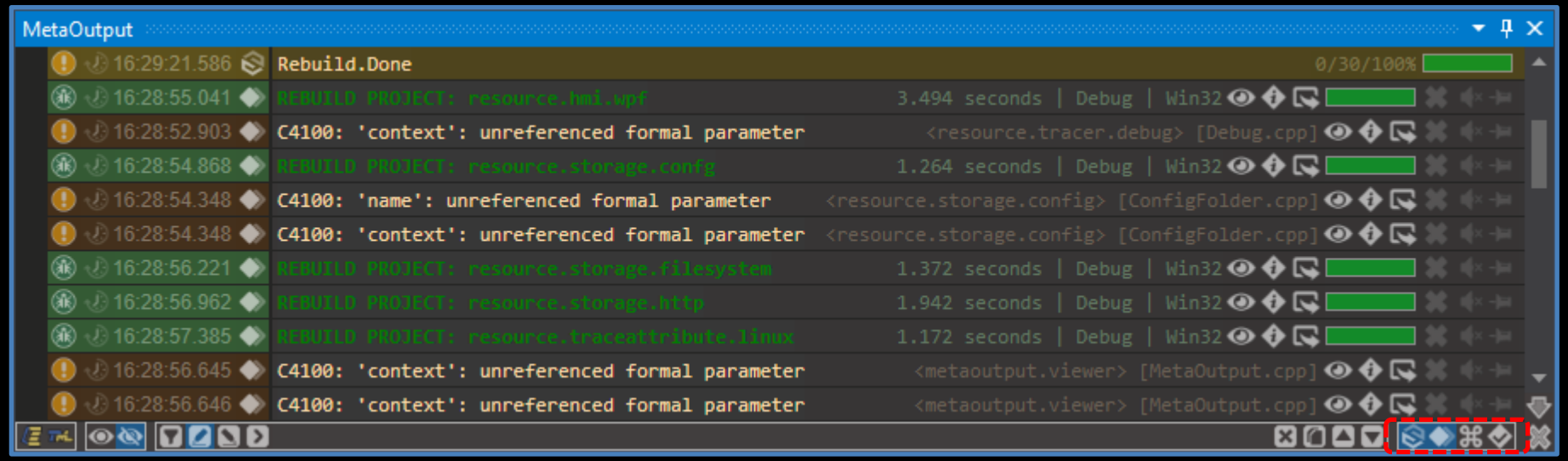
Highlighting all messages which have required substring in comment:



New messages will be also highlighted.

SOURCE FILTERING

User can show or hide any data source in one click:



WHAT IS METAOUTPUT BUILDING?

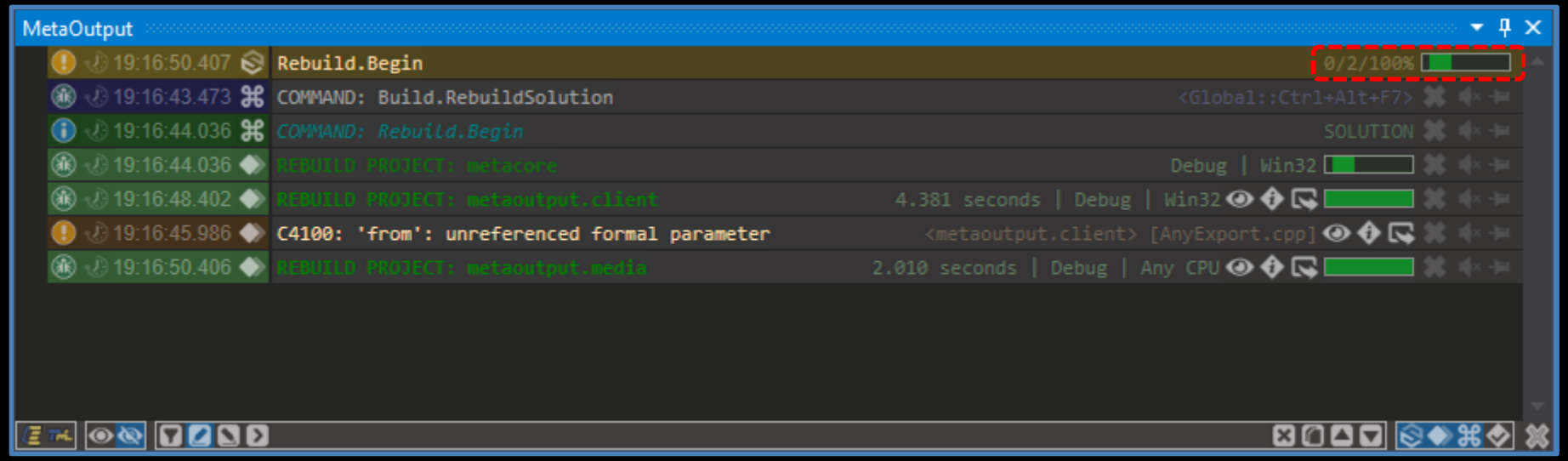
MetaOutput provides completely new user experience in project building.

It give possibility to see current building process together with compilation errors and warnings.

Also available compilation status for each project and overall status.

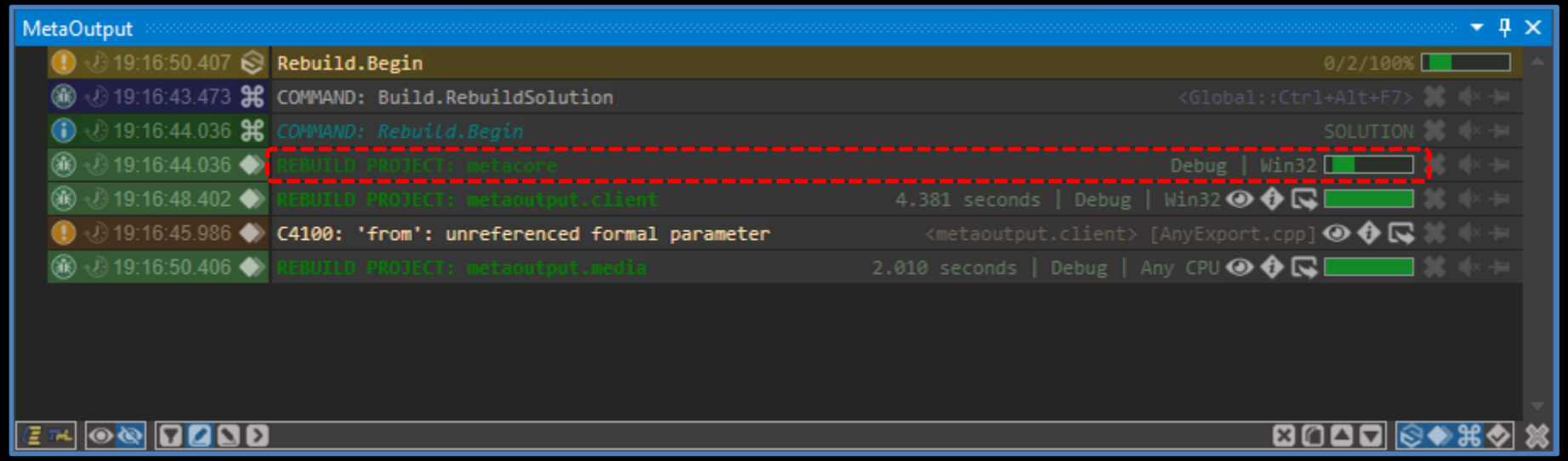
BUILDING: COMPILATION STATUS

Summary about how many projects compiled successfully and failed is here:



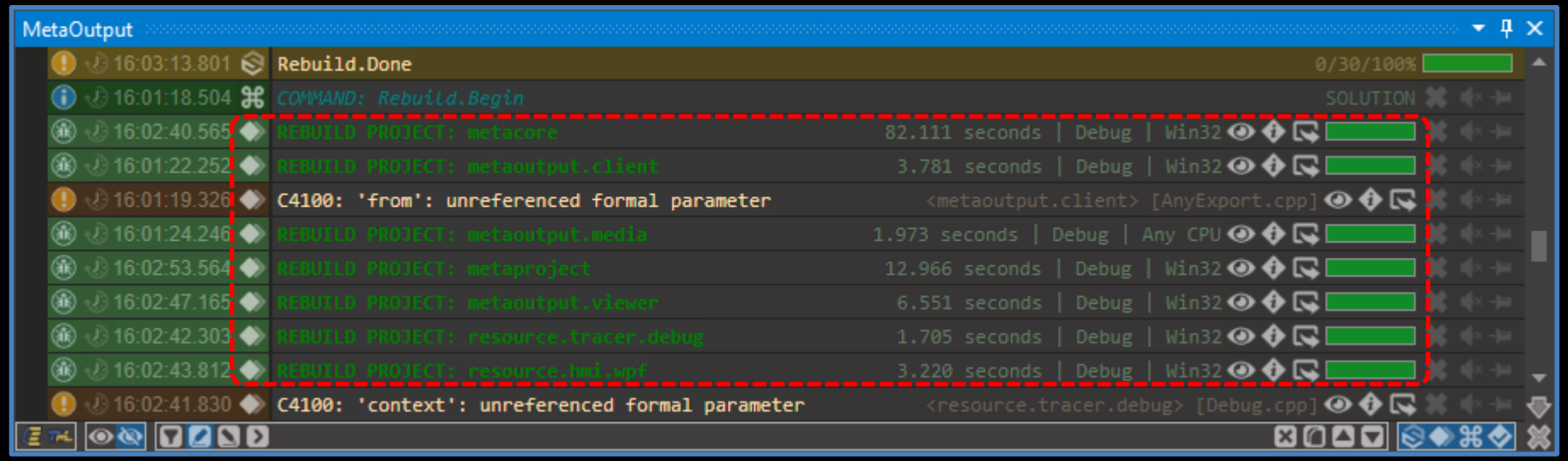
BUILDING: COMPILATION PROCESS

Detailed information about build process is available for each project:



BUILDING: COMPILATION RESULTS

Detailed information about compilation results is available for each project:



WHAT IS METAOUTPUT DEBUGGING?

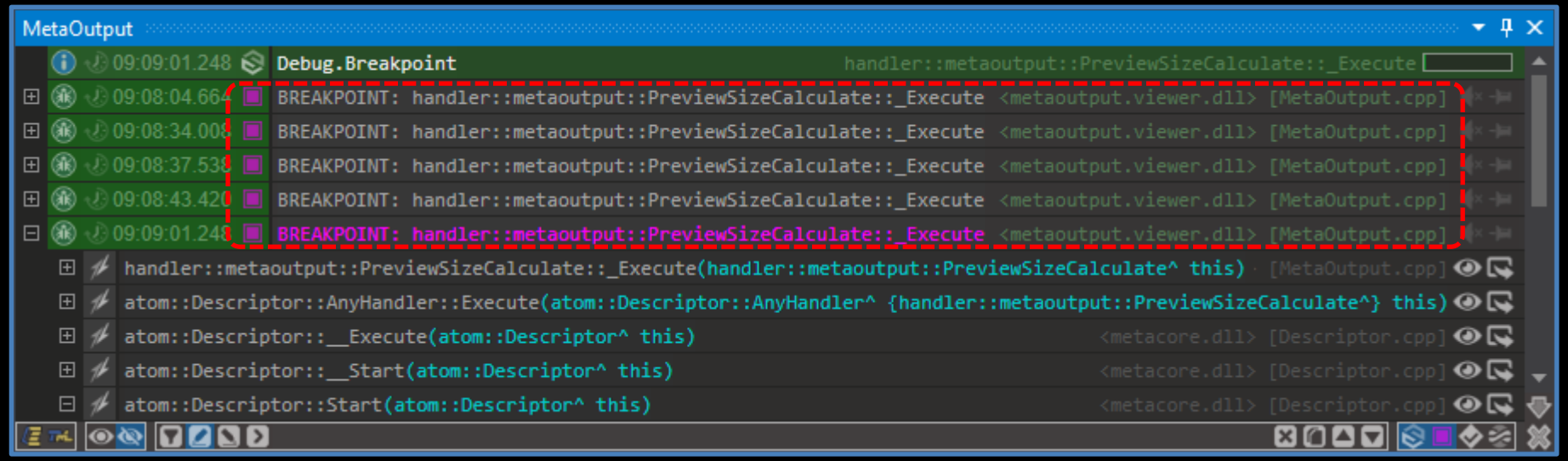
MetaOutput provides completely new user experience in software debugging.

It give possibility not only to see current state of application debugging, but to do it in the past.

It’s convenient mechanism, but also, it’s new vision of **Time Travel Debugging**.

DEBUGGING: BREAKPOINTS

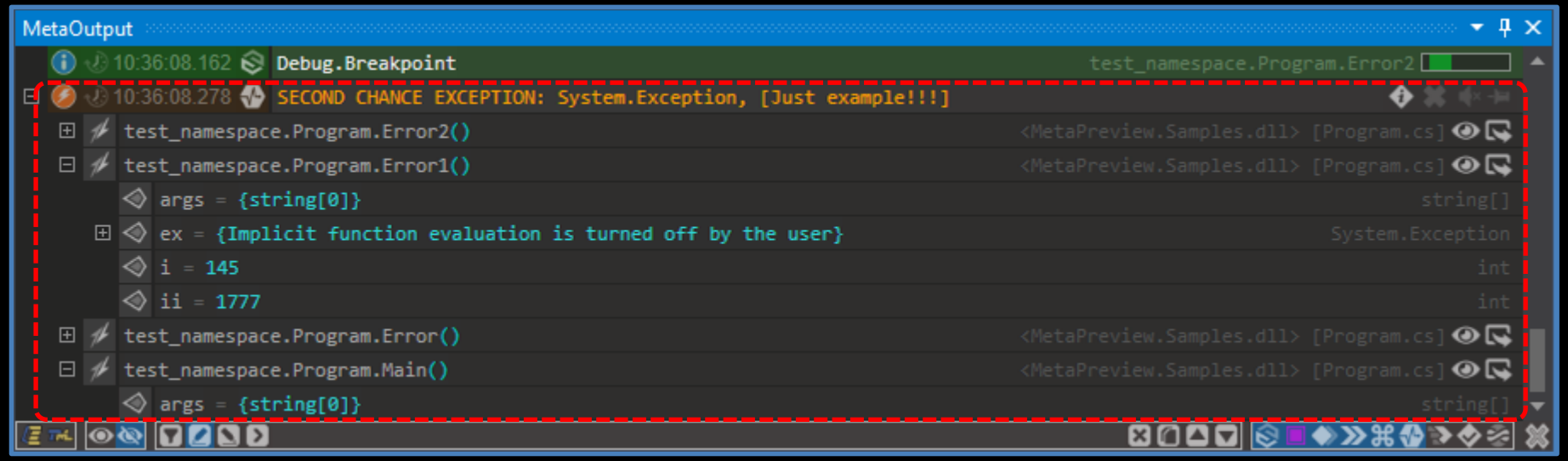
Detailed information is visualized for each application stops on breakpoints:



The latest breakpoint shows call stack automatically.

DEBUGGING: EXCEPTIONS

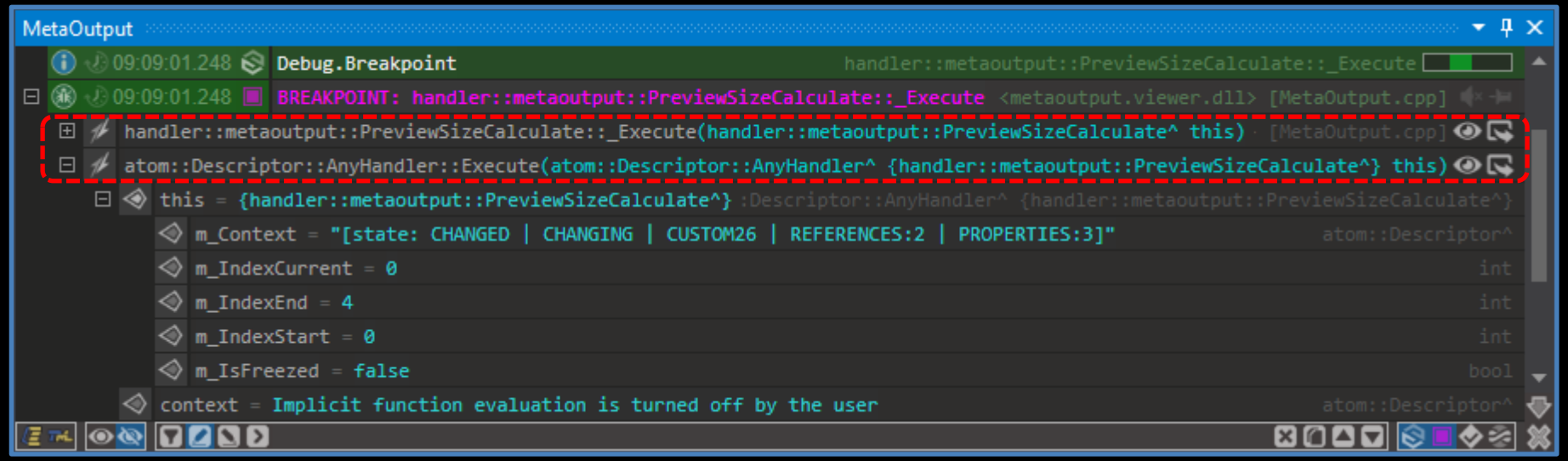
Detailed information is visualized for each thrown exceptions:



Stack trace with local variables is also available.

DEBUGGING: CALL STACKS

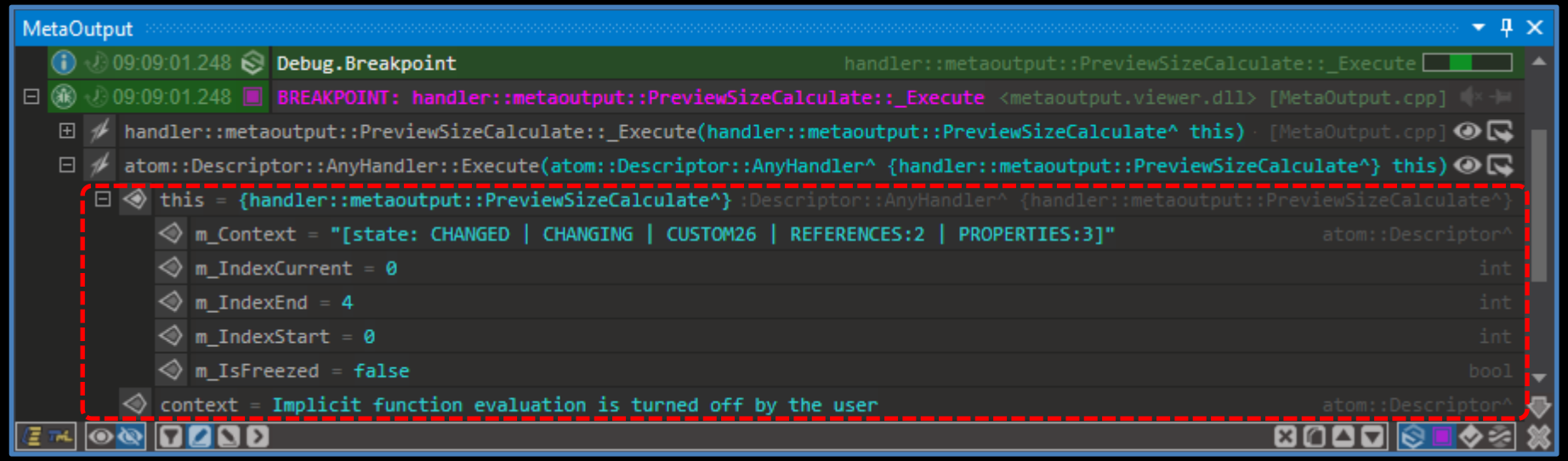
Call stack will be visualized for each breakpoint or exception:



Number of function calls can be restricted by user preferences.

DEBUGGING: VARIABLES

Local variables can be visualized for each function call:



Amount of data can be restricted by timeout and deep of extracted data.

DEBUGGING: NESTINGS

Collecting of data for classes and structures can have different deep.

It can be restricted by user preferences and timeouts.

DEBUGGING: TIMEOUTS

Some function calls can have too many variables.

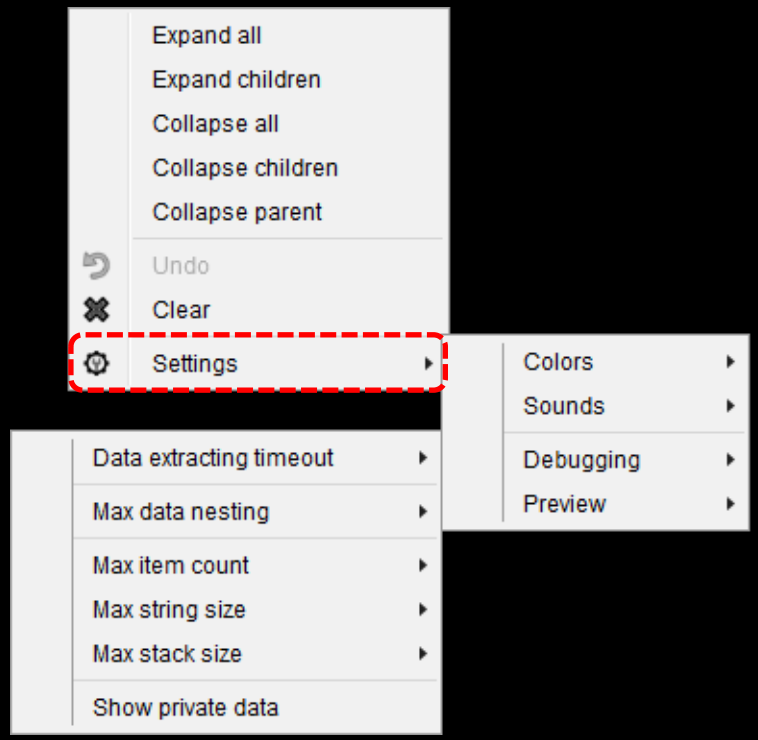
Their extraction can take a lot of time.

If to do it without any restrictions, this operation will make huge penalties for debugged process.

Therefore, data extracting will be stopped by ending of this operation or by exceeding of some interval of time.

DEBUGGING: SETTINGS

User have possibility to change any preferences by popup menu:



WHAT IS TRACE MESSAGE LANGUAGE (TML)?

TML is new data definition language integrated into **MetaOutput**.

- It can be used as substitution of **XML**, **JSON**, **YAML**, **INI** and other such languages;
- It can be used as new standard for logging;
- It can be used as new data container;
- It can be easily extended for many other goals...

IS TML ALREADY DEFINED AS STANDARD?

TML will be defined as **ISO** standard and specification will be available for all for free.

Any limitations with using, integrating to own products, commercial and noncommercial using will be absent.

However further development of this language **MUST BE COORDINATED** with author of this standard.

WHERE TO GET INFORMATION ABOUT TML?

Will be prepared additional presentation about **TML**.

HOW TO USE TML?

MetaOutput can visualize trace messages received from:

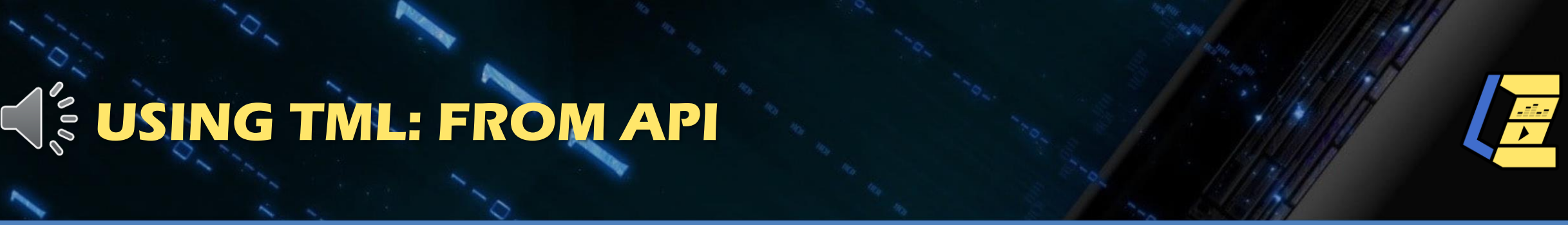
- From source code:
  - ***C#*** - *call function* *System.Diagnostics.Debug.WriteLine(tml_text);*
  - ***C++ CLI*** - *call function* *System::Diagnostics::Debug::WriteLine(tml_text);*
  - ***C++*** - *call function* *OutputDebugString(tml_text);*
- From **MetaOutput.Client** library;
- From **<CONSOLE MODE>** in **MetaOutput**.

USING TML: FROM SOURCE

This is example how to send **TML** message directly from source code:

```
System.Diagnostics.Debug.WriteLine(
    "[[[Some message]]] " +
    "@@@SOURCE COMMAND " +
    "@@@TYPE INFO " +
    "@@@COMMENT [[[Some comment]]] " +
    "@@@FONT.STATE BLINK ITALIC " +
    "@@@FOREGROUND TEAL");
```

USING TML: FROM API

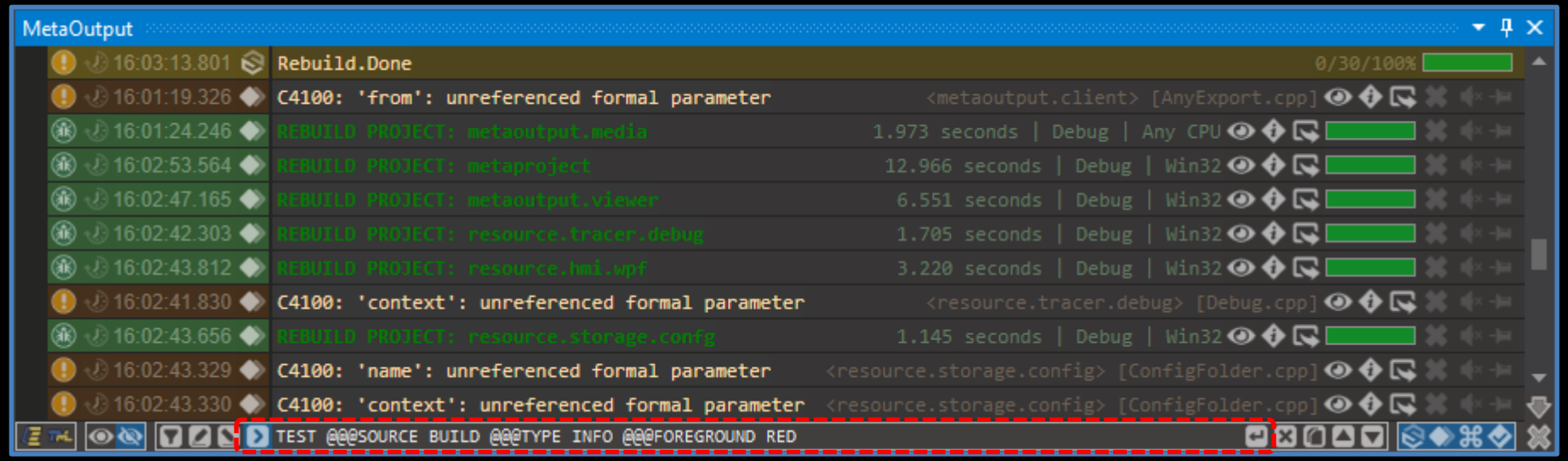


This is example how to send message using **MetaOutput.Client** library:

```
atom.Trace.GetInstance().
    SetFontState(atom.Trace.NAME.FONT_STATE.BLINK).
    SetFontState(atom.Trace.NAME.FONT_STATE.ITALIC).
    SetForeground(atom.Trace.NAME.COLOR.TEAL).
    SetComment("[[[Some comment]]]").
    Send(atom.Trace.NAME.SOURCE.COMMAND, atom.Trace.NAME.TYPE.INFO, 0, "[[[Some message]]]");
```

USING TML: FROM METAOUTPUT

New message is possible to send directly from **MetaOutput** using **TML**:



WHAT IS TML PATTERNS?

TML pattern is official extension of **TML**.

Each such pattern add to **TML** new attributes which are focused on using in some application area.

Some patterns, which can be commonly used will be accepted as standard.

TML PATTERN: TML.CORE

Pattern **TML.CORE** provides such functionality to **TML**:

- Comments
- Hierarchy
- Links
- Metadata
- Sources
- Timestamps
- Types
- Values

TML PATTERN: TML.ACTION

Pattern **TML.ACTION** provides such functionality to **TML**:

- Commands
- Conditions
- Transformations
- Translations
- Variables

TML PATTERN: TML.VIEWER

Pattern **TML.VIEWER** add next functionality to **TML**:

- Alignments
- Colors
- Controls
- Fonts

HOW TO WRITE NEW TML PATTERNS?

This library is in development.

COULD METAOUTPUT BE TRANSLATED?

MetaOutput uses **English** localization by default.

However, any string in user interface or any string in output can be translated to any other language using **TML**.

It can do any user which need it.

This is example demonstrates how to make it:

```
[[[TEST STRING]]] @@@TRANSLATION.APPEND <<<UA>>> [[[ТЕСТОВИЙ РЯДОК]]]
[[[TEST STRING]]] @@@TRANSLATION.APPEND <<<DE>>> [[[TESTSTRING]]]
[[[TEST STRING]]] @@@TRANSLATION.APPEND <<<IT>>> [[[STRINGA DI PROVA]]]
```

WHAT IS METAOUTPUT EXTENSIONS?

MetaOutput provides basic functionality, and mechanism how to expend it.

This mechanism is small and simple and it available using **MetaOutput.Client** library.

This library is implemented as **NuGet** package and available here:

https://www.nuget.org/packages/MetaOutput.Client

This library is open-source and source files are available here:

https://github.com/viacheslav-lozinskyi/MetaOutput

WHICH PLATFORMS ARE SUPPORTED?

Now implemented support only for **.NET Framework 4.5**+.

Support of the most popular platforms is planned.

WHICH LANGUAGES ARE SUPPORTED?

Now implemented support only for **C#** and **C++ CLI**.

Support of the most popular languages is planned.

METAOUTPUT EXTENSION: PREVIEW

This extension type is responsible for visualization of unknown for **MetaOutput** file formats.

For implementation is necessary to make inherited class:

```
class MyPreviewClass : extension.AnyPreview { … }
```

After that is necessary to make registration of this class:

```
extension.AnyPreview.Connect();
extension.AnyPreview.Register(".PNG", new MyPreviewClass());
```

METAOUTPUT EXTENSION: SOURCE

This extension type is responsible for sending to **MetaOutput** data from new sources.

For implementation is necessary to make inherited class:

```
class MySourceClass : extension.AnySource { … }
```

After that is necessary to make registration of this class:

```
extension.AnySource.Connect();
extension.AnySource.Register(new MySourceClass());
```

METAOUTPUT EXTENSION: EXPORT

This extension type is responsible for sending data from **MetaOutput** to some specific file format (for example log file).

For implementation is necessary to make inherited class:

```
class MyExportClass : extension.AnyExport { … }
```

After that is necessary to make registration of this class:

```
extension.AnyExport.Connect();
extension.AnyExport.Register(new MyExportClass());
```

METAOUTPUT EXTENSION: IMPORT

This extension type is responsible for sending to **MetaOutput** imported data in some specific file format (*for example log file*).

For implementation is necessary to make inherited class:

```
class MyImportClass : extension.AnyImport { … }
```

After that is necessary to make registration of this class:

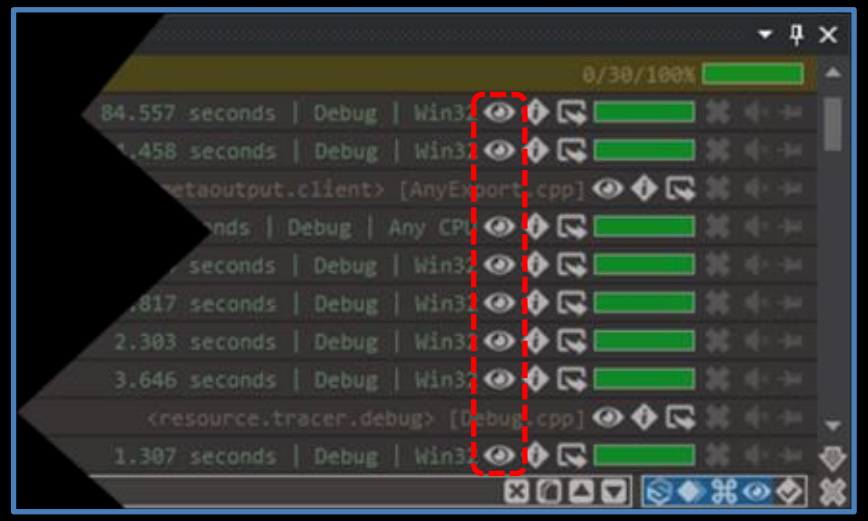
```
extension.AnyImport.Connect();
extension.AnyImport.Register(new MyImportClass());
```

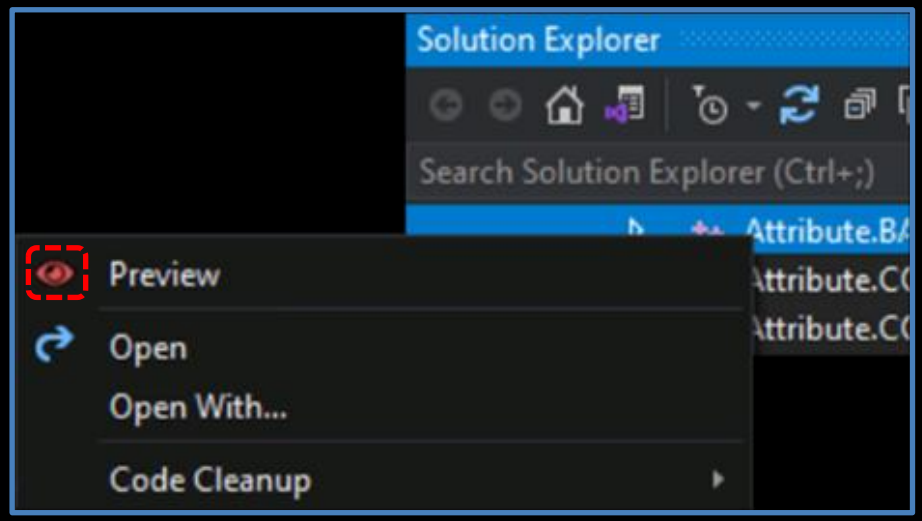
HOW TO USE EXTENSIONS?

Each file which have some **URL** can be previewed by installed extension.

If specialized extension is found, it will be responsible for visualization.

Otherwise, will be used generic mechanism.



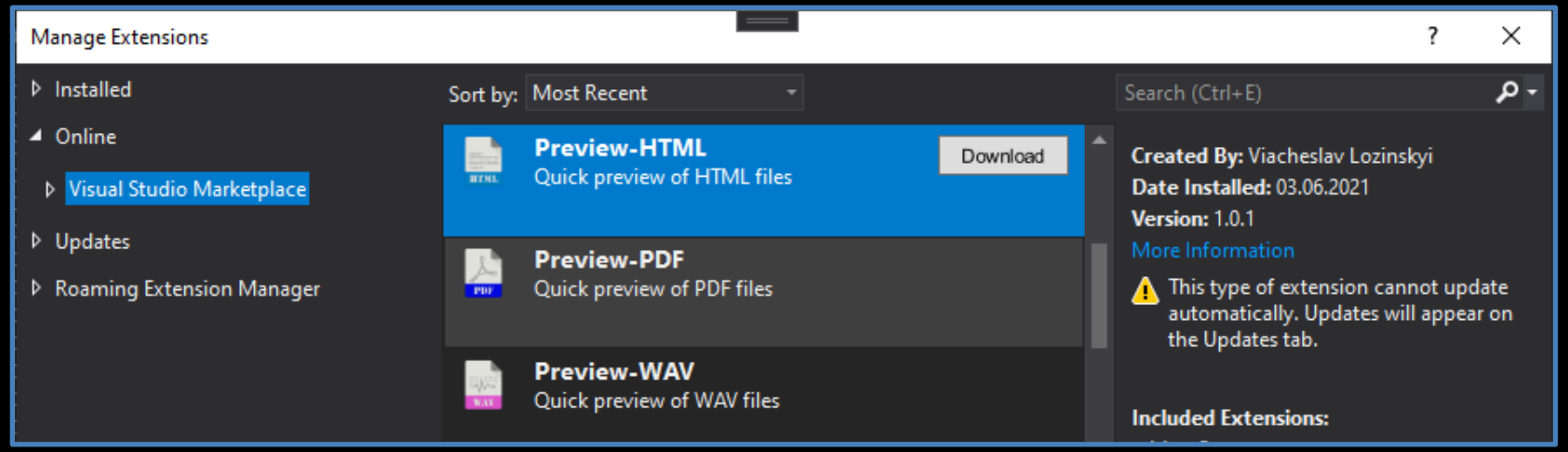


HOW TO INSTALL EXTENSIONS?

Installing of new extensions is possible from **Visual Studio Marketplace**:

https://marketplace.visualstudio.com/search?term=MetaOutput&target=VS&category=Tools

Also, it is possible to make directly from **IDE**:



HOW TO WRITE NEW EXTENSIONS?

The simplest way to write **MetaOutput** extensions is to use **MetaOutput.Client API**.

It provide enough mechanisms to write next types of extensions:

- **PREVIEW** (*visualization of required file formats*);
- **SOURCE** (*sending to **MetaOutput** data from some asynchronous source*);
- **IMPORT** (*import data from file in required format to **TML***);
- **EXPORT** (*export data from **TML** to file in required format*).

WHAT IS METAOUTPUT API?

MetaOutput provides several **API**s for next goals:

- Improvement of **MetaOutput**;
- Using of **MetaOutput** control in own applications;
- Using of **Trace Message Language** (***TML***) in own applications.

METAOUTPUT API: CLIENT

This **API** provides possibility to write **MetaOutput** extensions.

MetaOutput.Client library is available here:
https://github.com/viacheslav-lozinskyi/MetaOutput

Samples with using of this **API** are here:
https://github.com/viacheslav-lozinskyi?tab=repositories

METAOUTPUT API: SERVER

This **API** provides possibility to integrate **TML** language to own application.

It will support all available **TML** patterns and provide possibility to include all available extensions.

Implementation is planned.

METAOUTPUT API: VIEWER

This **API** provides possibility to integrate **MetaOutput** control to own application.

Implementation is planned.

INTEGRATION TO IDE

MetaOutput can be integrated into any **IDE** or editor using official plugin mechanism (*only for that products, who provide such **API***).

It doesn’t provide any influence on existing functionality, and it is working in parallel with standard tools and installed extensions from other vendors.

INTEGRATION TO IDE: VISUAL STUDIO

Plugin for **Visual Studio** is already implemented.

Supported versions are: **2015**, **2017**, **2019**, **2022**.

Implementation is here:
https://marketplace.visualstudio.com/items?itemName=ViacheslavLozinskyi.MetaOutput

INTEGRATION TO IDE: VISUAL STUDIO CODE

Plugin for **Visual Studio Code** is not implemented yet.

Implementation is planned.

INTEGRATION TO IDE: ECLIPSE

Plugin for **Eclipse** is not implemented yet.

Implementation is planned.

INTEGRATION TO IDE: XCODE

Plugin for **XCode** is not implemented yet.

Implementation is planned.

INTEGRATION TO IDE: OTHER IDE

Support for all popular **IDE**s will be implemented in any case.

Integration with other **IDE**s or editors is optional.

This is possible if owner of such product will finance this implementation and further support.

As alternative, development can be for free in case of significant promotion of **MetaOutput**.

If you are user of such **IDE** and want to use **MetaOutput** there, please send them this document. It will give possibility at least to start thinking about such project.

METAOUTPUT LICENSE

MetaOutput has proprietary license which provide next rights:

Permissions

- Commercial use
- Display
- Distribution
- Private use

Limitations

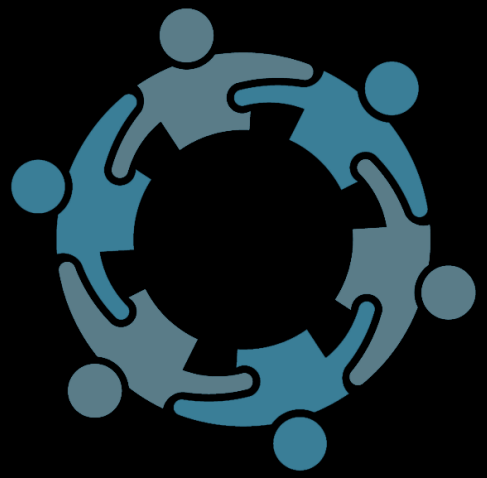
- Disclosure
- Liability
- Modification
- Warranty

METAOUTPUT PRICE

MetaOutput is freeware software.

In future it will have additional services which will be available by subscription.

ABOUT COMMUNITY



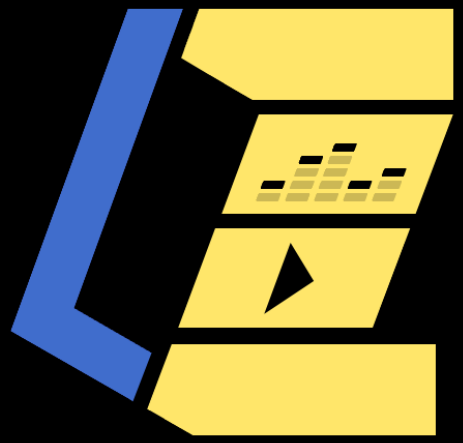
This community, it is about effectiveness.

Do more by less efforts!

- Any ideas is here…
- Any news is here…
- Any questions is here too…

Join us!

COMMUNITY: METAOUTPUT



Official community of **MetaOutput** is here:

Reddit: https://bit.ly/metaoutput-community-reddit

COMMUNITY: TML

Official community of **TML** is here:

Reddit: https://bit.ly/tml-community-reddit

ABOUT FEEDBACKS

Improvement of **MetaOutput** is impossible without feedbacks from real users.

It is very important part of development, and if you have some concerns or see any bug – don’t hesitate and report it!

All found bugs will be fixed (*often in the nearest release*).

All new feature requests will be performed (*the most useful of them will be implemented*).

FEEDBACKS: GIT

Primary place for reporting about any bugs and new features is here:

https://github.com/viacheslav-lozinskyi/MetaOutput/issues

Thanks in advance for any reported bug or feature request!

FEEDBACKS: VISUAL STUDIO MARKETPLACE

Visual Studio Marketplace supports review mechanism.

It is placed here:
https://marketplace.visualstudio.com/items?itemName=ViacheslavLozinskyi.MetaOutput&ssr=false#review-details

Will be great to see your rating there.
It will help to promote **MetaOutput** and save time for development.

IT IS VERY IMPORTANT, and thanks in advance for any comment there!

ABOUT REPOSTS

MetaOutput is new product.

By functionality it hasn't competitors but has one drawback - IT IS UNKNOWN.

Any repost is your contribution into this product.

IT IS VERY IMPORTANT, and thanks in advance for reposts!

ABOUT THIS DOCUMENT

This presentation was created by **Viacheslav Lozinskyi**.

Design of this document was made by **Maria Lozinskaya**.

The latest version of this document is available here:

https://bit.ly/metaoutput-presentation-pdf

The latest version of video presentation is available here:

https://bit.ly/metaoutput-presentation-youtube

CONTACTS

Author and owner of **MetaOutput** is **Viacheslav Lozinskyi**.

Contacts are next:

- **Email**: viacheslav.lozinskyi@gmail.com
- **LinkedIn**: https://www.linkedin.com/in/viacheslav-lozinskyi-a70750177/

THANKS FOR WATCHING