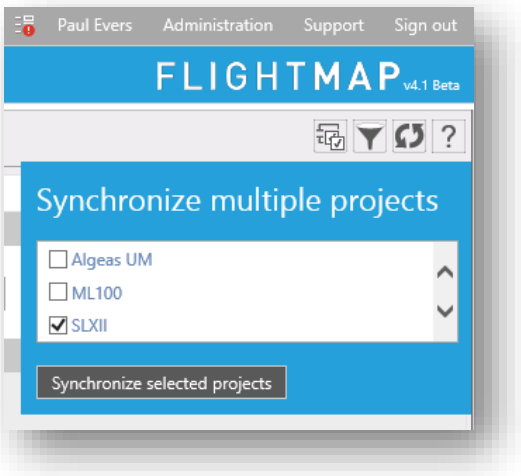
Introduction

This document describes FLIGHTMAP's connectivity architecture, as is available in FLIGHTMAP release 4.1. This is the first FLIGHTMAP release to fully implement the innovative FLIGHTHUB module that allows FLIGHTMAP to exchange information with a wide variety of external applications.

With the FLIGHTMAP connectivity architecture, a wide range of relevant uses cases is covered:

- Linking the FLIGHTMAP portfolio management portal to operational project management systems for aligning planning and tracking;
- Importing business and financial results from Enterprise Resource Planning (ERP) systems;
- Linking sources of innovative projects and trends to the portfolio in FLIGHTMAP (from sources such as idea management platforms and market report feeds);
- Connect to existing custom databases, spread sheets, and other file storage systems for migration or two-way file synchronization for off-line data management;
- Using FLIGHTMAP as the single dashboard across multiple different project and portfolio management solutions (as result of mergers, historical system differences, and practical tool variety);
- Linking multiple FLIGHTMAP portals together across the organization for consolidation.



Architectural vision and components

The architecture overview in Figure 1 shows the overall flow of information (in straight arrows) and control (in dashed arrows). External systems (destinations) are connected to FLIGHTMAP with a so-called Connector. All data exchange is channelled via these connectors.

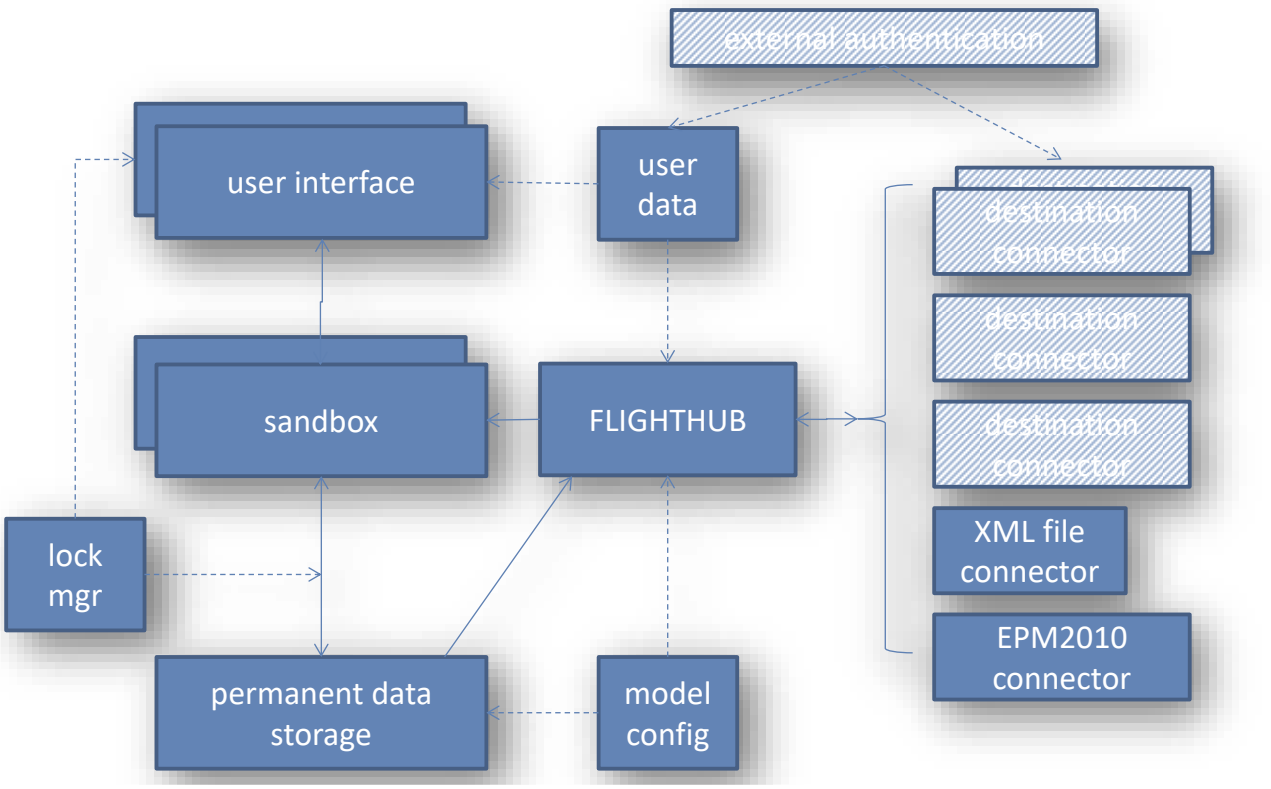


Figure 1 FLIGHTMAP Connectivity Architecture (shaded components require custom development).

New data that enters FLIGHTMAP from external sources is imported in the sandbox environment of the user that executes the import, so that the data can immediately be analysed and validated, before saving it to the underlying database. By making this a user decision, it is always clear who is accountable for the data in the FLIGHTMAP portal. For ease of data exchange from trusted sources, these two steps (import and save) can be combined in batch scripts that run periodically.

Data that is sent to an external system is taken from the permanent storage (not from the sandbox), so it is always traceable through the FLIGHTMAP logging mechanism which data is exchanged.

The architecture overview shows the central role of the FLIGHTHUB module and the other main components in the connectivity architecture.

- **FLIGHTHUB**: this central component maintains the list of connectors to external applications, and manages the links between data across the systems. FLIGHTHUB detects updates per project on both sides of the connector, and executes the actual synchronization of the data

(subject to the configuration of the link that indicates which data is synchronized in each direction). FLIGHTHUB communicates with all connectors via one standardized interface format (see later on), where the data is exchanged in XML format.

- The **Model Configuration** of the FLIGHTMAP portal also covers the mapping of FLIGHTMAP data fields to the FLIGHTHUB XML format (names, data types, and computations). FLIGHTHUB generates an XSD file based on this configuration which can be downloaded for use in the connectors.

- The **Connectors** actually collect data from and store data in the external destination systems, thereby translating the XML format into native data. Each Connector is a webservice that implements the FLIGHTHUB Interface via the standard WCF protocol (Windows Communication Foundation[1]).

- For authorized access to the external system, FLIGHTMAP can share a mapping of its **User Data** with the user account management of the destination system via the connector. This way the access rights of FLIGHTMAP users can be mapped on destination users; alternatively dedicated default user profiles can be set up as well (where the rights of access from FLIGHTMAP are set). In an IT landscape with federated or centralized identity management, the FLIGHTHUB and connector infrastructure can be integrated with this external authentication.

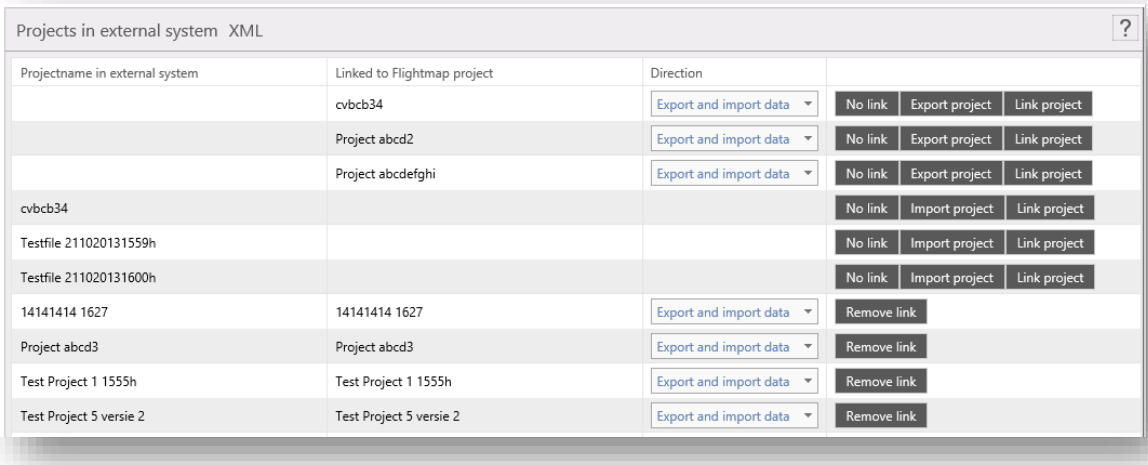


Figure 2 Linking projects via FLIGHTHUB

In the FLIGHTHUB data model, each FLIGHTMAP project links to one (or no) external project. This link is created by linking two existing projects, by creating a new FLIGHTMAP entry from a project in the

[1] See http://msdn.microsoft.com/en-us/library/ms731082.aspx

external system, or by creating a new entry in the external system from a FLIGHTMAP project. After this link is established, it is used to synchronize the selected project properties.

The user interface for managing the links between FLIGHTMAP and external projects is shown in Figure 2. The FLIGHTMAP help system explains how the FLIGHTMAP user can further use and edit this link, check updates, and synchronize projects.

Attaching a connector to the FLIGHTMAP portal

The steps to set up a connection between FLIGHTMAP and an external system are as follows (assuming a connector is available):

1. Choose connector deployment location
   FLIGHTMAP communicates with the connector via a WCF service interface, which is transparent for where the connector is actually hosted. This means it can be hosted on the FLIGHTMAP server, on the same destination server, or even on a separate server (where access to the destination is available).

2. Install and run connector at the right deployment location
   As for any WCF service, the connector must be available via its URL (e.g. http://ConnectorXML.flightmap.nl/FHConnectorService.svc). It is dependent on the connector technology how it must be deployed. For example, if the connector is developed as a WCF-service hosted in IIS (like the example connectors developed by Bicore), their code needs to be published to a dedicated folder (usually under inetpub\wwwroot) and added to IIS. We recommend to always use HTTPS over HTTP for secure communication.

3. Link connector to destination and FLIGHTMAP
   Linking the connector to the destination will be specific for each connector. The steps to activate the connector in FLIGHTMAP are straightforward, via the Administrator menu option External connectors.

   In order to configure the connector (see Figure 3):

   - Give the connector a meaningful name, set active to true;
   - Add the Webservice URL of the connector (see above) and its Client Token;
   - Illegal characters in projectname will be replaced by underscores (for the XML-connector this can be left empty; for the MS-EPM2010 the illegal characters are \/":;<>|,.'?*#);
   - The xml-tag for the projectname is ProjectName.

   In the second part of this page, you can configure which fields are available for this connector.

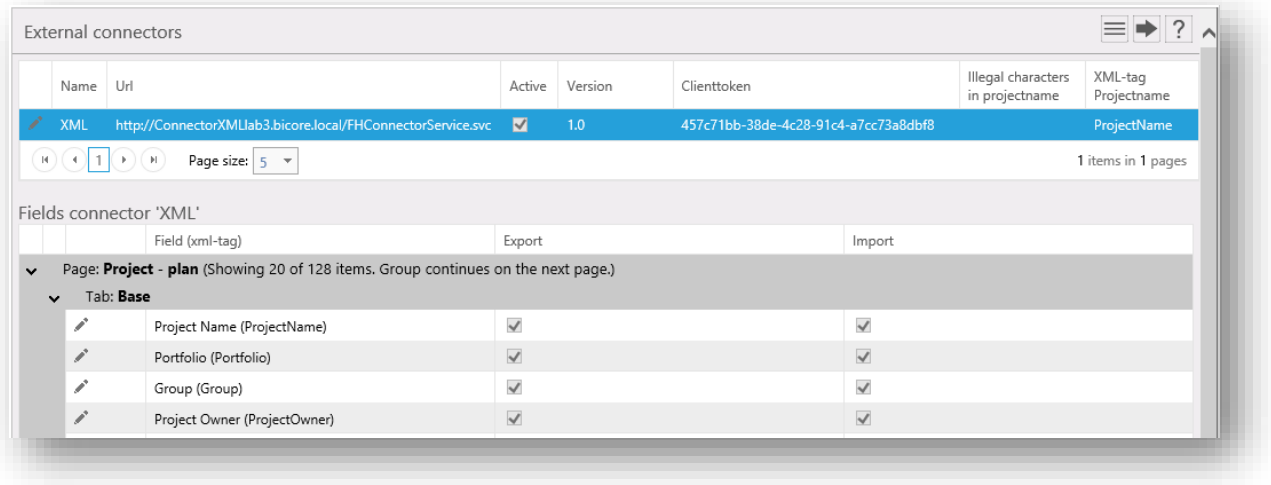


Figure 3 Configuring a connector in FLIGHTMAP

Developing FLIGHTMAP Connectors

The FLIGHTHUB architecture gives developers a lot of freedom to create a suitable connector for each external system. The connector only has to implement the IFHConnectorService interface in the WCF protocol (see Figure 4).

For future upgrades with backward compatibility, the interface includes a version number. FLIGHTHUB will use this version number to properly access each connector. The Client Token of the connector is a unique ID to identify Flightmap to the connector, to prevent unallowed calls to the connector's Webservice. This Client Token can be set via the connector configuration settings in FLIGHTMAP.

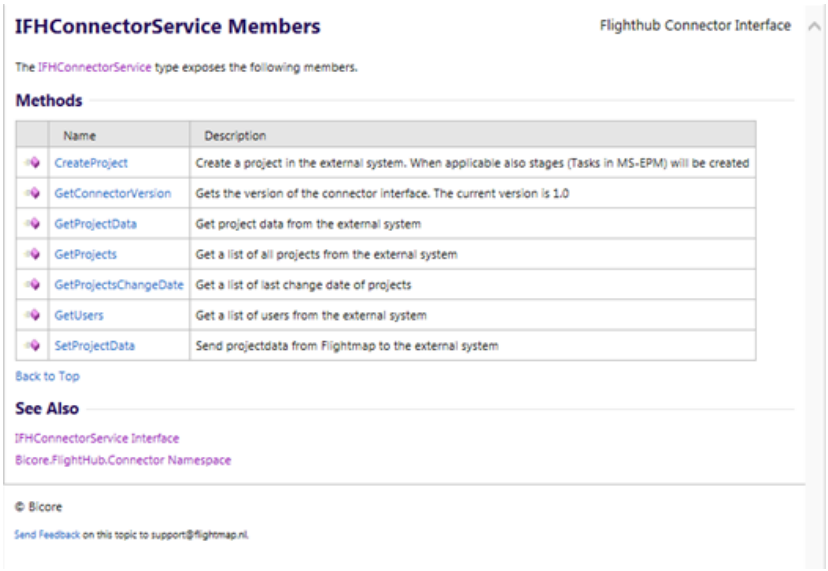


Figure 4 The IFHConnectorService interface

FLIGHTMAP supports connector developers by making available the following:

- XSD template generated by FLIGHTMAP for the XML data exchange format;
- Online documentation via http://connectorinterface.flightmap.nl (see Figure 4);
- Sample code (see Appendix A).

The FLIGHTHUB module in FLIGHTMAP release 4.1 comes bundled with an XML connector that offers two-way access to an XML file repository of your choice. It also features a working demonstrator version of the EPM2010 connector, which offers access to a typical hosted Microsoft Enterprise Project Management 2010 server environment.

Conclusion

With the FLIGHTHUB and Connector components in the FLIGHTMAP architecture, a wide range of connectivity requirements is solved. With a growing set of connectors for standard solutions and our connector development service for tailor-made connectors, all relevant management information from existing applications can be unlocked. In this way, the FLIGHTMAP connectivity architecture contributes to better data access, to better decisions, and hence to more value from the portfolio.

Bicore

Luchthavenweg 18C

5657 EB, Eindhoven, Netherlands

T +31 88 396 2777

info@flightmap.com

www.flightmap.com

Appendix A: Example implementation of the IHFConnectorService (in C#)

```
namespace Bicore.FlightHub.Connector.XML
{
    /// <summary>
    /// WCF service that exposes the methods for the Flighthub connector
    /// </summary>
    public class FHConnectorService : IFHConnectorService
    {
        /// <summary>
        /// Gets the version of the connector interface. The current version is 1.0
        /// </summary>
        /// <param name="data">input object</param>
        /// <returns>result object</returns>
        public ResultGetConnectorVersion GetConnectorVersion(DataGetConnectorVersion data)
        {
            ResultGetConnectorVersion tmpResult = new ResultGetConnectorVersion();
            string tmpError;
            Enums.ErrorLevel errorLevel;
            //check the received token
            if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
            {
                tmpResult.Error = tmpError;
                tmpResult.ErrorLevel = errorLevel.ToString();
            }
            else
            {
                tmpResult.Version = "1.0";
            }
            return tmpResult;
        }

        /// <summary>
        /// Create a project in the external system. When applicable also stages (Tasks in MS-EPM) will be created
        /// </summary>
        /// <param name="data">input object</param>
        /// <returns>result object</returns>
        public ResultCreateProject CreateProject(DataCreateProject data)
        {
            ResultCreateProject tmpResult = new ResultCreateProject();
            string tmpError;
            Enums.ErrorLevel errorLevel;
            //check the received token
            if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
            {
                tmpResult.Error = tmpError;
                tmpResult.ErrorLevel = errorLevel.ToString();
            }
            else
            {
                Connector connector = new Connector();

                Guid projectGuid = Guid.NewGuid();
                connector.CreateProject(projectGuid, data.ProjectName);

                tmpResult.InternalId = projectGuid.ToString();
                tmpResult.Error = connector.Error;
                tmpResult.ErrorLevel = connector.ErrorLevel.ToString();
            }
            return tmpResult;
        }

        /// <summary>
        /// Send projectdata from Flightmap to the external system
        /// </summary>
        /// <param name="data">input object</param>
        /// <returns>result object</returns>
        public ResultSetProjectData SetProjectData(DataSetProjectData data)
        {
            ResultSetProjectData tmpResult = new ResultSetProjectData();
            string tmpError;
```

```
    Enums.ErrorLevel errorLevel;
    //check the received token
    if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
    {
        tmpResult.Error = tmpError;
        tmpResult.ErrorLevel = errorLevel.ToString();
    }
    else
    {
        Connector connector = new Connector();
        connector.SetProjectData(data.Data);

        tmpResult.Error = connector.Error;
        tmpResult.ErrorLevel = connector.ErrorLevel.ToString();
    }
    return tmpResult;
}

/// <summary>
/// Get project data from the external system
/// </summary>
/// <param name="data">input object</param>
/// <returns>result object</returns>
public ResultGetProjectData GetProjectData(DataGetProjectData data)
{
    ResultGetProjectData tmpResult = new ResultGetProjectData();
    string tmpError;
    Enums.ErrorLevel errorLevel;
    //check the received token
    if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
    {
        tmpResult.Error = tmpError;
        tmpResult.ErrorLevel = errorLevel.ToString();
    }
    else
    {
        Connector connector = new Connector();
        tmpResult.Data = connector.GetDataProject(data.InternalIdList);
        tmpResult.Error = connector.Error;
        tmpResult.ErrorLevel = connector.ErrorLevel.ToString();
    }
    return tmpResult;
}

/// <summary>
/// Get a list of all projects from the external system
/// </summary>
/// <param name="data">input object</param>
/// <returns>result object</returns>
public ResultGetProjects GetProjects(DataGetProjects data)
{
    ResultGetProjects tmpResult = new ResultGetProjects();
    string tmpError;
    Enums.ErrorLevel errorLevel;
    //check the received token
    if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
    {
        tmpResult.Error = tmpError;
        tmpResult.ErrorLevel = errorLevel.ToString();
    }
    else
    {
        Connector connector = new Connector();
        tmpResult.Projects = connector.GetProjects();
        tmpResult.Error = connector.Error;
        tmpResult.ErrorLevel = connector.ErrorLevel.ToString();
    }
    return tmpResult;
}

/// <summary>
/// Get a list of last change date of projects
/// </summary>
/// <param name="data">input object</param>
/// <returns>result object</returns>
```

```
        public ResultGetProjectChangeDate GetProjectsChangeDate(DataGetProjectChangeDate data)
        {
            ResultGetProjectChangeDate tmpResult = new ResultGetProjectChangeDate();
            string tmpError;
            Enums.ErrorLevel errorLevel;
            //check the received token
            if (!Common.CheckToken(data.Token, out tmpError, out errorLevel))
            {
                tmpResult.Error = tmpError;
                tmpResult.ErrorLevel = errorLevel.ToString();
            }
            else
            {
                Connector connector = new Connector();
                tmpResult.ProjectsChangeDate = connector.GetProjectsChangeDate(data.InternalIdList);
                tmpResult.Error = connector.Error;
                tmpResult.ErrorLevel = connector.ErrorLevel.ToString();
            }
            return tmpResult;
        }
    }
}
```