

ILOG CDE 2.0

Introduction

June 2000

© Copyright 2000 by ILOG

This document and the software described in this document are the property of ILOG and are protected as ILOG trade secrets. They are furnished under a license or non-disclosure agreement, and may be used or copied only within the terms of such license or non-disclosure agreement. No part of this work may be reproduced or disseminated in any form or by any means, without the prior written permission of ILOG S.A.

Printed in France

Chapter 1 Presentation7

Introduction7

 The answer is CDE8

Cartridge concept9

Cartridge Development Environment.10

Benefits.11

Chapter 2 Architecture.13

Overview13

 Data flow14

OPML.15

Data model16

 Memory management16

 Inverted relation management16

 Predefined containers16

 Data model access.17

 Implementation.17

Internal mapping17

External mapping.18

Algorithms18

Structure19

 Commands.19

 Configuration19

Chapter 3 Developing with CDE21

Development overview21

Programming languages.22

Internal data model**.22**

Design22

Implementation.....23

Test23

Maintenance.....23

Documentation.....24

Algorithm**.24**

Implementation.....24

Test24

External connection.....**.24**

Design24

Implementation.....25

Test25

CDE user profiles**.26**

Designer.....26

Developer.....26

Integrator27

Team composition27

CDE Features**.28**

Code generation.....28

Programming interface.....28

Predefined architecture28

Predefined connections28

Predefined configurations28

Test environment28

Documentation generation.....29

Presentation

This chapter describes the cartridge concept and how it provides a consistent and efficient way to address customer extension needs.

Introduction

ERP systems have proven very efficient to centralize the enterprise data. One of the interesting use of this data is to analyze the various enterprise processes and to add optimization and/or scheduling to make them more efficient.

Most ERP systems provide predefined optimization and/or scheduling. It happens nonetheless that those features are not always sufficient to answer very specific customer needs. It is then necessary to hook to those existing systems some external optimization features.

Adding such features to ERP systems can be achieved by doing the following tasks:

- managing the connection to those systems,
- implementing a data model that will be used to represent the external systems data in memory and
- implementing optimizations algorithms as efficient as possible to solve the given problem.

Those tasks may be achieved with two very different strategies:

- the generic approach: one builds a generic system and sell it to the largest audience as possible. Achieving a generic optimization module can only be achieved through a large number of iterations/modifications and only if one focuses on a specific problem type for aa specific industry.
- the specific approach: one builds a specific system for each customer. This solution is a winning one only if one can avoid the important development/maintenance costs associated to the number of developed applications.

The answer is CDE

CDE stands for Cartridge Development Environment. Its goal is to facilitate the development of optimization extensions on top of existing systems. The two main advantages provided by CDE are the following:

- it reduces the amount of development to the maximum,
- it offers a predefined architecture for optimization extensions.

As we saw in the previous sections, those two points are fundamental whatever the approach taken:

- Reducing the development tasks ensures that the amount of code that needs to be maintained, understood and enhanced is as small as possible, it also ensures that one can concentrate on the most important part of an extension: its algorithms.
- Similarly, a predefined proven architecture gives the assurance that the extension development will not fail because of wrong architecture choices.

With CDE, connecting to an external system is easy and relies on standard technics. Implementing the data model is reduced to a design problem because most of the implementation is generated. Organizing the extension parts and controlling the extension functions is easy to specify and to implement.

If the generic approach is chosen, CDE will make it easy to iterate on existing code thanks to the code generation features. It will also help moving from one external system to another thanks to the cartridge architecture.

If the specific approach is chosen, CDE will reduce the maintenance costs thanks to the code generation feature. Extensions produced with CDE use the CDE architecture that guarantees a successful development.

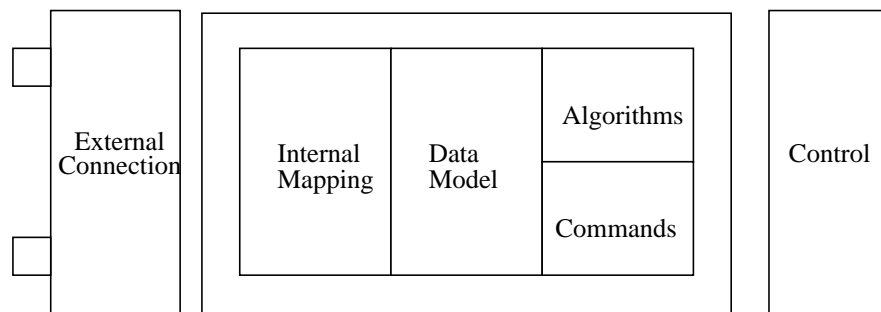
Furthermore, for each approach, CDE provides a scripting language that may be used to develop efficiently and easily the various customized parts of the generated code.

Cartridge concept

A CDE optimization extension is called a *cartridge*. It may be viewed as a black box with two interactions with the external world: one is the external data connection and the other is the end-user interface.

A cartridge is built from the following components:

- an external data connection part,
- an internal mapping part,
- an internal data model,
- an internal set of algorithms,
- a end-user control interface.



Data is loaded from the external system, it is then processed internally and stored in memory. The algorithms are run on the data and their results are stored in memory. The results may then be saved into the external system.

Those operations are triggered by the end-user using a control user interface¹. Depending on the cartridge configuration, this interface may be either graphic based, text based or batch oriented.

¹In CDE 2.0, the control user interface provides ways to activate the cartridge functions and to watch those functions execution. Data browsing is left to the external system user interface.

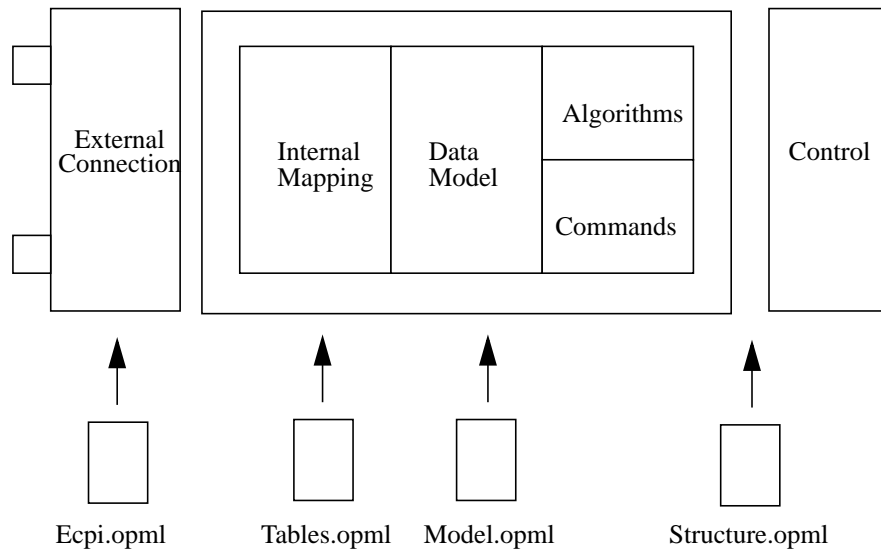
Cartridge Development Environment

CDE is built so that developing, maintaining and enhancing each part of a cartridge is as easy and as fast as possible.

A specification language (OPML) is used to specify:

- the object oriented internal data model,
- the mapping between this data model and an internal relational representation of this model,
- the bridges between the internal relational representation and the external data connection part,
- the cartridge end-user configuration.

This language is then processed and will generate most of the related cartridge code and the associated documentation. The generated code may then be customized using C++ language or the CDE scripting language.



This code is then assembled within a predefined architecture according to the chosen configuration (graphic user interface or batch-mode interface).

Benefits

Cartridge Development is made easier:

- A cartridge developer needs only to focus on the cartridge algorithms.
- External data connection can be implemented, tested independently from the data model and its algorithms.
- In the same way, testing the algorithms can be done without having to connect to the external system.

Cartridge maintenance is simplified:

- thanks to the OPML specification files, the hand-written code is greatly reduced compared to a standard development.
- documentation is generated from the OPML thus being always up to date with the real implementation.
- non-regression tests can be easily implemented.

And last but not least, Cartridge enhancement becomes possible:

- connecting to new external systems do not require rewriting code. One needs only to change the bridges specifications and this modification has no impact on the cartridge data model and thus has no impact on the algorithms.
- modifying the internal data model to cope with new algorithm strategies can be achieved without changing the external data connection.

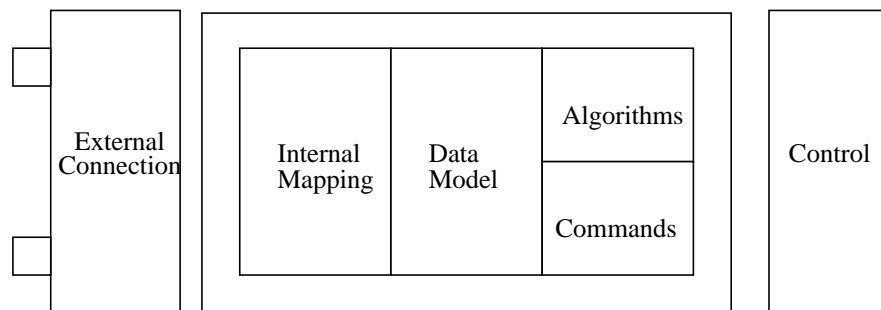
In the following chapter, we describe in detail the cartridge architecture.

Architecture

This chapter describes the cartridge architecture. For each part of the cartridge, an overview of its functions and a short description of the development work required to build it are given.

Overview

A cartridge is built from several components:



- the cartridge data model:

This part is used to represent all the business data in memory. This representation is object oriented. The different instances may be accessed using a C++ or a scripting programming interface. This data model provides easy memory management, sophisticated relationship management and a unified programming interface. This part is mostly generated.
- the cartridge internal mapping:

This part is used to manage the mapping between a relational representation of the data model and the data model itself. This representation is then used to connect the cartridge to the external systems. This part is mostly generated.
- the cartridge external bridges

This part is used to manage the connection between the external systems and the internal mapping component. Several bridges may be used at the same time to manage the data transfer between different external systems and the cartridge. This part is mostly generated and relies on predefined CDE bridges.
- the cartridge algorithm

This part is the heart of the cartridge. It uses a mathematical data model that is filled from the cartridge data model and implements some computation. The computation results are then put back into the cartridge data model so that they may be saved to the external systems.
- the cartridge structure:

This is the backbone of a cartridge, its role is to manage the cartridge configuration and the cartridge user-interface. It also gives access to all the other components and ensures that they all work together. It finally provides ways to formalize the cartridge functions so that they may be triggered by the end-user. This part is mostly generated.

All generated code may be customized and any such customization will be preserved in future code generations.

Data flow

Cartridges are meant to work in a batch like mode: a typical session using a cartridge would run as described below:

- data is loaded from the external system through the external bridges,
- data is mapped to the internal data model using the internal mapping component,
- data is then translated to a specific algorithm data model,
- the algorithm is run and simple feedbacks are sent to the end-user,
- results are stored in the internal data model,

- data is extracted from the internal data model using the internal mapping component and
- data is stored into the external system through the external data bridges.

Note that the load/save operations may not concern the whole data model: it is possible to describe partial load/save operations.

OPML

Most of the cartridge parts are described using OPML files. Those files are processed to generate the cartridge part code. The cartridge developer may then customize the generated code using preserved areas that will ensure that further code generation preserves the developer customizations.

An OPML file is a text file that describes the cartridge contents using UML concepts. Properties may be attached to the described entities. They will be used to guide the different code/documentation generation phases.

One will find below an example of an OPML file where a class, its attributes and operations are described.

```
class Message {
  /**
   * Message id.
   */
  id : String {
    before;
  }
  /**
   * Message Path.
   */
  path : MessagePath {
    before;
  }
  /**
   * Message type
   */
  type : MessageType = "ILG_UNKNOWN";
  /**
   * Indicates if the message is effectively activated.
   */
  <<derived>> activated: Boolean;
  /**
   * Message base where the message has been added.
   */
  role base : MessageBase {
    assocName= MessageBase_messages;
  }
}
```

```

/**
 * Helper function to compute an id from a path.
 */
<<static>> operation FromPathToId(path : MessagePath, inout id :String)
{
    jscript.visibility=protected;
}
}

```

Data model

The cartridge data model is used to represent the cartridge data in memory. This representation is object-oriented.

The data model is used as an intermediate representation between the external data and the algorithm mathematical model. Data may be accessed, transformed and/or checked easily using the generated programming interfaces and thus makes it possible to provide the cartridge algorithms with the best information formatted in the most optimal way.

The data model is implemented as a set of C++ classes that provide the following services:

- memory management,
- inverted relation management,
- predefined containers,
- scripting programming interface.

Memory management

All data model classes are relying on mechanisms that ensure a proper deletion mechanism. Instances may be automatically collected and moreover, relations may be declared as ownership relations. In such cases, when a owner object is deleted, its owned objects are also deleted. This makes it possible to very easily implement memory cleaning.

Inverted relation management

A relation may be described as an invert relation. Both relation attributes are then updated automatically whenever one of them is modified.

Predefined containers

Several containers are available to implement relations between classes. They are all based on the Standard Template C++ Library (STL), custom containers can be used too as long as they provide the STL programing interface.

- set: a set contains a non ordered number of instances,
- sequence: a sequence contains an ordered number of instances,
- map: a map contains a number of instances that may be accessed through a given key. Note that this key is one of the instance attributes.
- multimap: a multimap contains a number of instances that may be accessed through several keys. Those keys are chosen among the instance attributes.

All containers may be accessed using a uniform programming interface.

Data model access

All data model classes may be accessed using C++ and CDE scripting language. The code generation ensures that a standardized programming interface is available and makes it easy to use any generated class.

Scripting access may be selective so that internal classes may be hidden.

Implementation

To implement the cartridge data model, the following work has to be performed:

- describe the data model in the data model OPML file,
- generate the data model code using CDE generators,
- complete the generated classes with the data model operations code.

Internal mapping

The internal mapping is used to prepare the mapping between the external data and the cartridge data model. It defines a relational schema that is mapped with the cartridge data model.

To implement the cartridge internal mapping, the following work has to be performed:

- describe the schema in the internal mapping OPML file,
- describe how the data model maps on this schema in the data model OPML file,
- generate the internal mapping code using CDE generators.

External mapping

The external mapping goal is to transfer data between several *dataviews*. A *dataview* represents either a data source or a data collector. The external system and the cartridge internal mapping are viewed as *dataviews*.

Bridges are used to link *dataviews* and to transform data from an input *dataview* to an output *dataview*.

Several predefined *dataviews* are available:

- file oriented: such a view enables reading/saving from a text file in Microsoft Excel format (CSV).
- memory oriented: such a view enable reading/saving from objects in memory. It relies on a generated internal mapping (see previous section).
- relational database oriented: such a view enable reading/saving from relational databases.
- composite: such a view groups a set of elementary *dataviews*.

Bridges are used through named *requests*. A request activates a bridge to transfer data between two *dataviews*.

To implement the cartridge external mapping, the following work has to be performed:

- describe the different requests, the *dataviews* and the used bridges in the external mapping OPML file.
- generate the external mapping configuration file.

Note that defining and using the external mapping does not require any knowledge on the cartridge data model. It is even possible to implement and test that mapping without any data model implementation.

Algorithms

Algorithms are written using C++ language. They may use any C++ tools such as ILOG optimization libraries.

To implement the cartridge algorithms, the following work has to be performed:

- implement the algorithms,
- implement the transformation from the data model into the algorithm data model.

The generated programming interface provides an access to the data model instances making it easy to program any required transformations. Those transformations may be written using either C++ or CDE scripting language.

The clear separation between the data model and the external data sources make it possible to the algorithm developer to implement in a very short amount of time some simulated data sources. It is thus quite easy to test and debug the algorithms without having to cope with integration problems.

Structure

The cartridge structure describes the different commands that can be triggered by the end-user to execute the cartridge functions. It also contains configuration related information.

Commands

Each *command* is associated with a C++ or a scripting function, a set of parameters and a set of indicators.

- Parameters are used to provide end-user input to the cartridge functions. They could represent for example a time frame, a number of iterations, a database name.
- The C++ or scripting function will be executed when the command is triggered by the end-user. It will receive the parameters as arguments. The executed code is implemented by the developer.
- Indicators are used to provide information to the end-user during the command execution. They could represent for example how the solution quality evolve during the algorithm resolution.

To implement the cartridge commands, the following work has to be performed:

- describe the commands in the structure OPML file,
- generate the corresponding code using CDE generators,
- complete the generated code with the commands function code.

Configuration

A cartridge configuration describes the set of active commands within the cartridge, how the different components of the cartridge will be agenced and how the end-user will interact with the cartridge. It also provides initialization information such as the required script files, the directory paths where those files must be searched.

CDE 2.0 supports linked configurations: a cartridge is implemented as a unique process containing all the different cartridge components.

Two end-user interfaces are available:

- an alphanumerical console where one can use CDE scripting language to activate the cartridge commands. A console built with this interface may also be used in a batch mode.
- a graphic user interface displaying the various cartridge commands and providing ways to enter those commands parameters and visualize the commands indicators.

To implement the cartridge configuration, the following work has to be performed:

- describe the configuration in the structure OPML file,
- generate the configuration code using CDE generators,
- implement a `main` function that will at least call the generated functions that build and run the cartridge.

Developing with CDE

This chapter describes the various phases of a cartridge development. It describes CDE usefulness for each phase. It then gives a detailed description of the different developers profile and finally lists all CDE features.

Development overview

As we saw in the previous chapter, developing a cartridge means developing several components. Those components may be separated in three major parts: the algorithms, the internal data model and the external data connection.

Each of these parts will go through the standard phases of software development:

- design,
- implementation,
- test,
- maintenance,
- documentation.

CDE supports the cartridge developers during all those phases for each of these parts¹.

¹*CDE 2.0 focuses mostly on the internal data model and on the external data connection.*

Because those parts do not require the same technical skills, CDE makes it possible to develop those parts separately with different teams and defines how and when those teams have to work together.

Programming languages

CDE heES

Implementation

Most of the internal data model implementation relies on code generation or predefined code.

internal data model classes

The internal data model classes implementation is twofold: the first step is to use the CDE generators to have all C++ classes generated from the data model OPML file. The second step consists in adding within the generated code all specific code (specific operation code for example).

Some further development may be needed to implement specific operations such as data checking and/or data transformations. Implementing those operations can be achieved using either the C++ language or the CDE scripting language or both.

internal relational schema

No specific work is necessary to implement the internal relational schema: the OPML information is used by CDE when the connection to the external systems is implemented.

Test

Most of the internal data model code does not require any testing thanks to the fact that it is generated. It may nevertheless be useful to set up automated tests for specific operations, checking and/or transformations.

CDE provides a predefined mechanism that enables the cartridge developer to add and launch tests. This mechanism makes it possible to launch batches of tests automatically and get a corresponding report.

Tests may be written using either C++ or CDE scripting language or both.

Maintenance

The internal data model implementation results from a mixing of code generation and hand-written code. Most of the implementation is generated which means that maintenance is mostly done at the OPML file level.

CDE provides a documentation generator that helps maintaining an up-to-date documentation. This documentation is a great tool to have a good understanding of the internal data model.

If this internal data model requires modifications, the developer modifies the OPML file and only needs to go through the code generation phase to have an updated model. CDE ensures that all code that was added manually remain available whatever the modifications.

Documentation

CDE provides a documentation generator that takes as input the internal data model OPML files and generates a reference manual describing the different data model classes, their attributes and the way they relate to each other.

This documentation generation takes advantage of the OPML file comments.

Algorithm

Implementation

Algorithm implementation requires two different works from a developer: one concerns the algorithm development and the other the integration of this algorithm with existing data.

The first work may be performed using ILOG optimization tools.

The second work is mostly linked to data transformation: each algorithm is associated with a very specific data structure that has to be filled from available data. Transforming data from the CDE data model into this algorithm data structure is done using either the CDE scripting language or C++ programming language. The former makes it possible to develop such transformation in a fast manner using an interpreted language and the later provides an efficient final implementation.

Test

The CDE test environment can be used to set up non regression tests for the cartridge algorithms. Those tests may either be written using C++ code or CDE scripting language.

External connection

Design

Designing an external connection means describing the different requests that the cartridge will submit to the external systems to retrieve and to save its data. A request is fully described by the bridge it relies on and by the dataviews the bridge is linking.

CDE provides predefined dataviews to access supported external systems³.

³CDE 2.0 supports the following dataviews: CSV formatted file, Relational databases and Objects in memory

An external connection is described through the external connection OPML file. This file lists the various requests, bridges and dataviews. It also refers to the internal relational schema.

Implementation

An external connection is built from two parts: a configuration file that will describe the requests, bridges and dataviews and some system dependent code that glue the cartridge external connection part to the external systems.

Configuration

The external connection OPML file is used to generate an external connection configuration file. This file is written using CDE scripting language and is loaded within the cartridge to configure the connection mechanisms.

System dependent code

The final connection to the external systems depends of the targeted systems:

- no specific code is required to read/write from/to flat files,
- it might be necessary to build specific views on top of existing relational databases,
- no specific code is required to map to the internal data model objects.

Note that implementing the final connection is done using the standard tools of the targeted external systems.

Test

Testing the external connection is made easy thanks to the clear separation between the internal data model and the connection parts.

One can validate the external connection without having any knowledge on the internal data model using the following configurations:

- retrieve data from the external systems and store it in a supported format (flat file or relational database),
- save data from a supported format into the external systems.

Similarly, one can read/write data in the internal data model from/to any supported format.

It is thus easy to implement and validate an external connection on the targeted site and tests separately the cartridge internal parts on a distant site.

CDE user profiles

This section describes the different profiles required to work with CDE. It also gives an indicative description of a typical cartridge development team.

Designer

Cartridge designers need to have the following knowledge:

- business problem
Modeling an optimization algorithm to solve a given problem means having a very deep knowledge on the business problem.
 - optimization understanding
 - external systems
Precise knowledge on the available data is vital to a cartridge development: the targeted external systems have to be able to give access to the necessary data.
-

Developer

Cartridge developers have in fact two different profiles.

Data Model developer

The first profile is dedicated to the implementation of the internal data model and the internal relational schema. It requires the following skills:

- OPML and CDE scripting languages
OPML files are used to modelize most of the parts of the cartridge. The CDE scripting language is a JavaScript implementation and is used to accelerate the cartridge functions implementation and to ease the cartridge configuration. It may also be used to set up cartridge tests.
- Object model
The internal data model is described using UML concepts. Those concepts are tightly related to object oriented modeling. Note however that cartridge models do not rely on complex object oriented features such as multiple inheritance.
- C++ language
All cartridge internal parts are coded using the C++ language.
- Relational database concepts

The internal data model must be implemented according to certain rules so that it can be mapped to the internal relational schema. Setting up this schema and the mapping require knowledge on relational database concepts such as tables, keys.

Algorithm developer

The second profile is related to the algorithms development and requires the following skills:

- CDE scripting language and C++ language

To align the algorithm mathematical model with data, it is necessary to browse through the cartridge internal data model. This is achieved using either C++ or CDE scripting language. Note that no sophisticated object oriented knowledge is required for this task.

- Optimization tools

Algorithms have to be implemented using specific tools such as ILOG optimization libraries. It is thus required to know how to use those tools and of course how to implement the optimal algorithms using those tools.

Integrator

A cartridge integrator is in charge of connecting the external connection part of the cartridge to the external systems. This work requires the following skills:

- OPML specification of the external connection

The external connection part is built from an OPML description. This description does not require any knowledge on the cartridge internal model.

- Relational database concepts

The external connection part will heavily rely on the internal relational schema which defines the cartridge internal data using relational based structures.

- External system standard tools

When connecting the external connection part of a cartridge to external systems, it may be necessary to add some glue code. This work depends on the targeted systems but is done using the external system tools.

Team composition

A typical team composition would require one designer, one developer and one integrator. Note that the developer and integrator can work in parallel once the internal relational schema has been set up.

CDE Features

This section summarizes the CDE various features.

Code generation

Cartridge code is mostly generated from the different OPML files. The generated code is written in C++ (data model and structure) or in CDE scripting language (connection configuration).

Developer code may be added to it and is preserved by code generation.

Programming interface

The generated code provides a standardized programming interface. This interface is available in C++ and in CDE scripting language.

All generated classes may be handled in both languages.

Predefined architecture

Cartridges are built according to a predefined architecture. This reduces considerably the development time and the development risks.

The proposed architecture forces a clear separation of the different development tasks and thus enables separate developments and separate tests.

This architecture is based on ILOG experience in building optimisation extensions.

Predefined connections

CDE comes with a set of predefined connections (bridges). This reduces the development time and provides ways to move from one external system to another or to follow the various external systems releases.

Predefined configurations

Two predefined configurations are available. Both provides a predefined user interface that does not require any specific development.

Test environment

A test environment is available with various tools that makes it easy to implement non-regression tests.

Documentation generation

Documentation is generated from the OPML description files. This ensures that documentation and code are always up to date.

Documentation is available in HTML and RTF (Microsoft Word) formats.

